

Thomas Much



Java

für Mac OS X



Galileo Computing 

Liebe Leserin, lieber Leser,

was macht die Faszination von Apples Betriebssystem Mac OS X aus, so dass in letzter Zeit auch immer mehr Anwender und Entwickler auf diese Plattform umsteigen oder sie zumindest als zusätzliche Plattform nutzen? Neben Design und einfacher Bedienung spielt hier sicherlich der UNIX-Kern des Systems die größte Rolle. Dadurch können viele bekannte Open Source-Werkzeuge auch unter der Mac-Oberfläche eingesetzt werden. Und: Apple betrachtet Java als eine der Standard-Programmierungsumgebungen von Mac OS X.

Ob Sie bereits Mac-Anwender sind und nun Java programmieren möchten, oder als Java-Entwickler jetzt auf Mac OS X umsteigen möchten, dieses Buch ist auf jeden Fall die richtige Lektüre für Sie. Aber auch, wenn Sie hauptsächlich für andere Systeme entwickeln und Ihre Programme nun optimal an Mac OS X anpassen möchten, finden Sie hier wertvolle Informationen. Thomas Much ist als erfahrener Softwareentwickler und passionierter Mac-Nutzer bestens gewappnet, um die beiden Welten Mac OS X und Java zusammenzubringen.

Dieses Buch wurde mit großer Sorgfalt geschrieben, begutachtet, lektoriert und produziert. Sollte dennoch etwas nicht so funktionieren, wie Sie es erwarten, dann scheuen Sie sich nicht, sich mit mir in Verbindung zu setzen. Ihre freundlichen Anregungen und Fragen sind jederzeit willkommen.

Und nun viel Spaß bei der Lektüre

Ihr Stephan Mattescheck

Lektorat Galileo Computing

stephan.mattescheck@galileo-press.de

www.galileocomputing.de

Galileo Press • Gartenstraße 24 • 53229 Bonn

Auf einen Blick

Vorwort	11
1 Grundlagen	19
2 Entwicklungsumgebungen	73
3 Grafische Benutzungsoberflächen (GUI)	117
4 Ausführbare Programme	173
5 Portable Programmierung	285
6 Mac OS X-spezifische Programmierung	307
7 Grafik und Multimedia	351
8 Werkzeuge	383
9 Datenbanken und JDBC	421
10 Servlets und JavaServer Pages (JSP)	443
11 J2EE und Enterprise JavaBeans (EJB)	461
12 J2ME und MIDP	481
A Kurzeinführung in die Programmiersprache Java	509
B Java auf Mac OS 8/9/Classic	531
C Java 1.5 »Tiger«	541
D System-Properties	547
E VM-Optionen	565
F Xcode- und Project Builder-Einstellungen	577
G Mac OS X- und Java-Versionen	585
H Glossar	589
I Die Buch-CD	595
Index	597

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

ISBN 3-89842-447-2

© Galileo Press GmbH, Bonn 2005

1. Auflage 2005

Der Name Galileo Press geht auf den italienischen Mathematiker und Philosophen Galileo Galilei (1564–1642) zurück. Er gilt als Gründungsfigur der neuzeitlichen Wissenschaft und wurde berühmt als Verfechter des modernen, heliozentrischen Weltbilds. Legendär ist sein Ausspruch **Eppur se muove** (Und sie bewegt sich doch). Das Emblem von Galileo Press ist der Jupiter, umkreist von den vier Galileischen Monden. Galilei entdeckte die nach ihm benannten Monde 1610.

Das vorliegende Werk ist in all seinen Teilen urheberrechtlich geschützt. Alle Rechte vorbehalten, insbesondere das Recht der Übersetzung, des Vortrags, der Reproduktion, der Vervielfältigung auf fotomechanischem oder anderen Wegen und der Speicherung in elektronischen Medien.

Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, können weder Verlag noch Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen.

Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Lektorat Stephan Mattescheck

Korrektur Harriet Gehring

Einbandgestaltung Barbara Thoben

Herstellung Iris Warkus

Satz SatzPro, Krefeld

Druck und Bindung Koninklijke Wöhrmann, Niederlande

Inhalt

Vorwort	11
1 Grundlagen	19
1.1 Historisches	22
1.2 Aktuelles	24
1.3 Den Mac richtig bedienen	25
1.4 Eingeben, übersetzen, ausführen	35
1.5 Entwicklerwerkzeuge und -dokumentation	42
1.6 Up to date bleiben	44
1.7 Mac OS X erkennen	47
1.8 Konfigurationsdateien speichern	52
1.9 Aufbau des Apple-Java-Systems	55
1.9.1 Java-Struktur anderer Systeme	57
1.9.2 Java-Struktur von Mac OS X	60
1.10 Kleine Fallen	69
1.11 Literatur & Links	70
2 Entwicklungsumgebungen	73
2.1 Project Builder und Xcode	76
2.1.1 Projekttypen	76
2.1.2 Projektstruktur	79
2.1.3 Eingeben und übersetzen	82
2.1.4 Ausführen und debuggen	87
2.1.5 Build-System	90
2.1.6 Versionsverwaltung	94
2.2 Eclipse	98
2.3 NetBeans und Sun Java Studio Creator	103
2.4 IntelliJ IDEA	105
2.5 OmniCore CodeGuide	106
2.6 Borland JBuilder	107
2.7 Borland Together Control Center und Together Solo	108
2.8 Oracle JDeveloper	108
2.9 Metrowerks CodeWarrior	110

2.10	jEdit	111
2.11	Jext	112
2.12	JJEdit	113
2.13	BlueJ	113
2.14	DrJava	115
2.15	Literatur & Links	116

3 Grafische Benutzungsoberflächen (GUI) 117

3.1	Das AWT und die Java 1.3-Apple-Erweiterungen	121
3.1.1	Eine einfache AWT-Anwendung	122
3.1.2	Fensterhintergrund	134
3.1.3	Kontext-Popup-Menüs	135
3.1.4	Dateiauswahl-Dialoge	136
3.1.5	Portables Einbinden der Java 1.3-Apple-Erweiterungen	139
3.2	Swing und die Java 1.4-Apple-Erweiterungen	142
3.2.1	Konfiguration des Aussehens (Look & Feel)	144
3.2.2	Ein Swing-Beispiel mit mehreren Dokument-Fenstern	146
3.2.3	Portables Einbinden der Java 1.4-Apple-Erweiterungen	163
3.2.4	Hinweise zu diversen Swing-Komponenten	165
3.3	Fenster mit »Brushed Metal«-Aussehen	167
3.4	Drag & Drop	169
3.5	»Headless«-Applikationen ohne Benutzungsoberfläche	170
3.6	Literatur & Links	171

4 Ausführbare Programme 173

4.1	Shell-Skripte im Terminal	176
4.2	Doppelklickbare JAR-Archive	182
4.3	Mac OS X-Applikationen	187
4.3.1	Programmsymbole	188
4.3.2	Programmpakete	190
4.3.3	Jar Bundler	193
4.3.4	Project Builder, Xcode und Eclipse	203
4.3.5	Programmpakete von Hand erzeugen	206
4.4	Installationswerkzeuge	219
4.4.1	Mac OS X-Hausmittel	220
4.4.2	Kommerzielle Installer	227
4.5	Java Web Start	234
4.6	Applets	250
4.6.1	Applets programmieren und einsetzen	251
4.6.2	Web-Browser	265
4.6.3	Kommunikation mit dem Browser	271
4.7	Literatur & Links	282

5 Portable Programmierung 285

5.1	Strategien zur portablen Programmierung	288
5.2	Häufige Problembereiche	292
5.2.1	Datei- und Pfadtrennzeichen	292
5.2.2	Zeilenenden	294
5.2.3	Zeichenkodierung	295
5.2.4	Runtime.exec()	297
5.3	Oft benötigte Lösungen	298
5.3.1	PDF-Dokumente	298
5.3.2	Web-Browser und HTML-Dokumente	299
5.3.3	Zugriff auf serielle Schnittstellen	301
5.3.4	Hochauflösende Zeitmessung	301
5.3.5	Rendezvous	302
5.4	Literatur & Links	305

6 Mac OS X-spezifische Programmierung 307

6.1	JNI	309
6.1.1	JNI-Bibliotheken mit Xcode erzeugen	310
6.1.2	Installationsverzeichnisse für JNI-Bibliotheken	317
6.1.3	JNI-Bibliotheken mit gcc erzeugen	318
6.1.4	Abhängigkeiten von anderen Bibliotheken	319
6.1.5	Laufzeitumgebungen (JVMs) erzeugen	321
6.2	JDirect und JManager	324
6.2.1	JManager	325
6.2.2	JDirect	325
6.3	Zugriff auf Datei-Metadaten	327
6.4	Cocoa Java	330
6.5	AppleScript	340
6.5.1	AppleScript mit Cocoa Java	340
6.5.2	AppleScript mit Shell-Kommandos	342
6.6	Speech & Spelling Frameworks	343
6.6.1	Speech Framework	343
6.6.2	Spelling Framework	345
6.7	Weitere systemabhängige Bibliotheken	347
6.7.1	Posix-Aufrufe mit dem Suite/P Toolkit	347
6.7.2	Authorisation Toolkit	348
6.8	Literatur & Links	349

7 Grafik und Multimedia 351

7.1	Java Advanced Imaging (JAI) und Java Image I/O (JIO)	353
-----	--	-----

7.2	Java 3D	356
7.3	OpenGL/JOGL	358
7.4	Java Media Framework (JMF)	362
7.5	QuickTime for Java (QTJava)	365
	7.5.1 QTJava 6.0 (QuickTime bis Version 6.3)	367
	7.5.2 QTJava 6.1 (QuickTime ab Version 6.4)	371
7.6	Sound/Musik	374
7.7	Drucken	375
7.8	Literatur & Links	380

8 Werkzeuge 383

8.1	Ant und Maven	385
	8.1.1 Ant in Xcode	388
	8.1.2 Ant in Xcode 1.5	393
	8.1.3 Ant in Eclipse	394
	8.1.4 Programmpakete mit Ant erzeugen	394
	8.1.5 Maven	396
8.2	JUnit	398
	8.2.1 JUnit in Xcode	401
	8.2.2 JUnit in Eclipse	402
8.3	Decompiler	403
8.4	Obfuscators	404
8.5	Bytecode Viewer (Disassembler)	406
8.6	Profiler	408
8.7	JavaBrowser	410
8.8	Jikes	412
8.9	Groovy	413
8.10	UML-Modellierung	416
8.11	<oxygen/> XML-Editor	417
8.12	Literatur & Links	418

9 Datenbanken und JDBC 421

9.1	SQL	424
9.2	JDBC	426
	9.2.1 Datensätze lesen	427
	9.2.2 Datensätze schreiben	430
9.3	Datenbanken	431
	9.3.1 MySQL	432
	9.3.2 PostgreSQL	438
	9.3.3 Firebird	439

9.3.4	HSQLDB	440
9.3.5	Oracle	440
9.3.6	Sybase Adaptive Server Enterprise	441
9.3.7	Berkeley DB Java Edition	441
9.4	ODBC	441
9.5	Literatur & Links	442

10 Servlets und JavaServer Pages (JSP) 443

10.1	Servlets	446
10.2	Tomcat	448
10.3	JavaServer Pages (JSP)	453
10.4	Webapplikationen	455
10.5	Literatur & Links	459

11 J2EE und Enterprise JavaBeans (EJB) 461

11.1	JBoss	464
11.2	Enterprise JavaBeans (EJB)	467
11.2.1	Eine einfache Enterprise-Applikation	468
11.2.2	XDoclet	477
11.3	Literatur & Links	478

12 J2ME und MIDP 481

12.1	MIDlets entwickeln und testen	485
12.2	Einsatz im mobilen Endgerät	496
12.2.1	Lokale Installation	497
12.2.2	Installation vom Web-Server	502
12.3	Benutzungsoberflächen und Grafik	503
12.4	Literatur & Links	508

A Kurzeinführung in die Programmiersprache Java 509

A.1	Programme, Klassen, Pakete	509
A.2	Methoden und Anweisungen	513
A.3	Variablen und Datentypen	515
A.4	Fallunterscheidungen und Schleifen	516
A.5	Klassen und Objekte	518

A.5.1	Vererbung	520
A.5.2	Assoziationen	522
A.5.3	Polymorphismus	523
A.5.4	Abstrakte Klassen und Interfaces	524
A.5.5	Innere und anonyme Klassen	526
A.6	Fehlerbehandlung mit Exceptions	528
A.7	Literatur & Links	530

B Java auf Mac OS 8/9/Classic 531

C Java 1.5 »Tiger« 541

D System-Properties 547

D.1	Vordefinierte System-Properties	547
D.2	Konfigurierbare System-Properties	549

E VM-Optionen 565

F Xcode- und Project Builder-Einstellungen 577

F.1	Projekteinstellungen	578
F.2	Installationseinstellungen	578
F.3	Target-Einstellungen	579
F.4	Java-Einstellungen	580
F.5	Programmpaket-Einstellungen	582
F.6	Standardpfade	583

G Mac OS X- und Java-Versionen 585

H Glossar 589

I Die Buch-CD 595

Index 597

Vorwort

*»Für die meiste Entwicklungsarbeit verwende ich mein PowerBook. Ich halte es für beträchtlich effizienter als ein Desktop-System, denn zum einen hat es alle Möglichkeiten eines vollständigen UNIX-Desktops, zum anderen kann ich es mitnehmen, weil es alle Laptop-Eigenschaften besitzt. Ich kann im Flugzeug arbeiten, zu Hause oder in irgendeiner Ecke, während ich in einer langweiligen Besprechung sitze. Und man kann damit nicht nur e-mailen und surfen, es besitzt voll funktionale, professionelle Entwicklungswerkzeuge.«
(James Gosling, Erfinder der Java-Technologie)¹*

Liebe Leserin, lieber Leser!

Wenn der Erfinder der Java-Technologie so lobende Worte über ein Apple-System verliert, wird man hellhörig – was macht die Faszination an Apples Betriebssystem Mac OS X aus, dass in letzter Zeit immer mehr Entwickler auf diese Plattform umsteigen oder sie zumindest als zusätzliche Plattform nutzen? Und das bei einem System, dem vor ein paar Jahren kaum ein Außenstehender mehr zugetraut hat, als ein paar nette Grafiken zu bearbeiten – und dessen Maus sowieso (mindestens) eine Taste fehlt?

Zum einen bestechen Apple-Rechner seit Jahren durch gutes Design, das viele kleine Raffinessen bietet. Dadurch, dass Apple sowohl die Computer als auch das Betriebssystem und wichtige Anwendungen entwickelt, sind Hardware und Software perfekt aufeinander abgestimmt. Die einfache Bedienbarkeit des klassischen Mac OS ist legendär, und aktuelle Mac OS X-Versionen reichen an diese Benutzerfreundlichkeit wieder heran. Durch den UNIX-Kern von Mac OS X erreicht das System nicht nur eine hervorragende Stabilität, es ist auch in der Grundeinstellung bereits gut gegen Angriffe aus dem Internet geschützt – Sicherheitslücken sind selten und werden schnell behoben, Trojaner und Würmer können dementsprechend wenig anrichten, Viren sind nahezu unbekannt.

Der UNIX-Kern ist für Entwickler besonders interessant, schließlich können dadurch viele bekannte Open-Source-Werkzeuge unter Mac OS X genutzt werden – entweder werden diese bereits standardmäßig mitgeliefert, oder Sie können sie sich selbst kompilieren. Und falls Sie die grafische Oberfläche bevorzugen und Programme wie Adobe Acrobat oder Microsoft Office verwenden wollen, ist dies auch kein Problem.

¹ <http://www.apple.com/pro/science/gosling/>

Der wichtigste Punkt aber, warum Sie überhaupt dieses Buch aufgeschlagen haben: Java ist fest ins System integriert – Apple betrachtet Java als eine der Standard-Programmierungsumgebungen für Mac OS X!

Für wen ist dieses Buch geschrieben?

Dieses Buch spricht sehr unterschiedliche Zielgruppen an: vom langjährigen Mac-Anwender, der nun Java programmieren möchte, bis hin zum langjährigen Java-Entwickler, der auf Mac OS X umsteigen oder zumindest seine Anwendungen daran anpassen will.

Damit trotz der verschiedenen Voraussetzungen alle zumindest das Zielsystem verstehen und mit den gängigen Begriffen umgehen können, bietet das erste Kapitel die Grundlagen sowohl für die Mac-Bedienung als auch für die Java-Entwicklung mit den Standard-Kommandozeilenwerkzeugen. Danach wird dann angenommen, dass Sie entweder Java schon beherrschen, gerne Beispiele ausprobieren (die lauffähig auf der Buch-CD beiliegen) oder nebenbei die Sprache mit einem weiteren Buch lernen (falls Ihnen das Anhang-Kapitel zu den Java-Grundlagen nicht ausreicht).

Dennoch finden sich in den Kapiteln immer wieder Hinweise, die für Profis vermutlich nichts Neues sind, die aber Einsteigern über viele kleine Fallen hinweghelfen. Die ausführliche Beschreibung der Beispiele richtet sich gerade in den ersten Kapiteln an die Einsteiger – Profis werden sich hier wohl eher an den abgedruckten Listings orientieren.

Was dieses Buch nicht ist: eine allgemeine Einführung in die Programmierung oder eine komplette Java-Referenz. Sie sollten bereits ein wenig Programmiererfahrung in irgendeiner Sprache besitzen, damit Sie etwas mit Begriffen wie »Variable« oder »Schleife« anfangen können. Und den kompletten Umfang von Java zu berücksichtigen, kann nicht Ziel dieses Buches sein – es konzentriert sich darauf, die Software-Entwicklung für die verschiedenen Java-Bereiche (J2SE, J2EE, J2ME) auf Mac OS X grundlegend vorzustellen und dabei vor allem die Mac-Besonderheiten ausführlich zu beschreiben.

Entscheidungshilfe für potentielle »Switcher«?

Wenn Sie schon lange einen Mac besitzen oder auch kürzlich erst umgestiegen sind, stellt sich für Sie die Frage nach der Entwicklungsplattform nicht mehr – Sie möchten Java auf Mac OS X programmieren und portabel auf den diversen Java-Systemen einsetzen. Lesen Sie einfach beim nächsten Abschnitt weiter!

Wenn Sie hauptsächlich für andere Systeme entwickeln und Ihre Java-Anwendungen einfach nur möglichst gut an Mac OS X anpassen möchten: Seien Sie

beruhigt – Sie benötigen nicht zwingend einen Apple-Rechner, um die Informationen dieses Buches verwerten zu können. Die meisten Anpassungen für MacOS X lassen sich auch mit einer Entwicklungsumgebung unter Windows oder Linux realisieren. Allerdings sollten Sie bedenken, dass ein vernünftiger Softwaretest immer nur auf dem System stattfinden kann, wo die Software später auch eingesetzt wird. Daher sollten Sie sich zumindest das Betriebssystem selbst zulegen und dann mit einer geeigneten Emulationssoftware (beispielsweise »PearPC« von <http://pearpc.sourceforge.net/> bzw. <http://www.pearpc.net/>) auf einem schnellen PC einsetzen.

Sollten Sie schließlich zu der immer größer werdenden Gruppe der Java-Entwickler gehören, die sich gerade einen Umstieg auf MacOS X überlegen, kann Ihnen dieses Buch eine gute fachliche Entscheidungshilfe sein. Suchen Sie dazu alle Java-Themen zusammen, die Sie derzeit für Ihre Projekte benötigen, und klären Sie anhand der jeweiligen Kapitel, ob und wie diese auf MacOS X genutzt werden können – in der Regel wird es dabei kaum oder keine Probleme geben. Eine Frage, die in diesem Zusammenhang immer wieder gestellt wird: Ist die Java-Entwicklung mit den Mac-Entwicklungsumgebungen wirklich so viel langsamer als unter Windows? Dies kann ich Ihnen leider nicht pauschal beantworten. Wenn man beide Welten kennt, merkt man bei der Benutzungsoberfläche definitiv einen Unterschied. Aber was für die einen unerträglich und ein K.o.-Kriterium ist, stört die anderen wenig oder gar nicht. Ganz am Anfang haben Sie bereits James Gosling kennen gelernt, den Erfinder der Java-Technologie, der nach eigenem Bekunden für fast alle Entwicklungsaufgaben seinen portablen Apple-Rechner verwendet. Wenn Sie also die in diesem Buch gezeigten Java-Möglichkeiten auf MacOS X interessieren, gehen Sie doch einfach mal zu einem Mac-Händler in Ihrer Nähe und probieren Sie dort Eclipse oder NetBeans aus.

Wie sollten Sie dieses Buch lesen?

Als **Einsteiger** sollten Sie zunächst die Java-Kurzeinführung im Anhang lesen, falls Sie Java nur wenig kennen oder Ihr Grundlagenwissen auffrischen möchten. Dann können Sie mit Kapitel 1 loslegen. Wenn Sie den Umgang mit MacOS X noch nicht lange gewohnt sind, erfahren Sie am Anfang des Kapitels, wie Sie den Mac richtig bedienen. Daran schließen sich die absoluten Grundlagen von Java auf MacOS X an. Das Ende des Kapitels über den Aufbau der Apple-Java-Implementierung ist zunächst vermutlich nicht so interessant für Sie. Lesen Sie dann auf jeden Fall noch Kapitel 2 über die diversen Entwicklungsumgebungen und Kapitel 4 über die unterschiedlichen Möglichkeiten, wie Sie ein Java-Programm als ausführbares Programm verteilen können. Die restlichen Kapitel lesen Sie dann bei entsprechendem Interesse.

Als **Profi** wird Sie zunächst der letzte Teil von Kapitel 1 über die Struktur des Apple-Java-Systems interessieren. Gerade wenn das Apple-Betriebssystem neu für Sie ist, kann es aber auch nicht schaden, wenn Sie sich mit den grundlegenden Bedienweisen des Macs am Anfang des Kapitels vertraut machen. Kapitel 2 über die Entwicklungsumgebungen können Sie kurz überfliegen, ebenso Kapitel 8 über die diversen Werkzeuge, die Ihnen zum Teil vielleicht schon von anderen Systemen her bekannt sind. Kapitel 4 über ausführbare Programme und die Formen der Programmdistribution sowie das kurze Kapitel 5 über portable Programmierung sollten Sie auf jeden Fall lesen; wenn Sie Anwendungen mit grafischen Benutzungsoberflächen entwickeln, gilt dies auch für Kapitel 3. Zur Feinoptimierung Ihrer Applikationen finden Sie im Anhang Listen der System-Properties und der VM-Optionen, ansonsten lesen Sie die restlichen Kapitel bei Bedarf.

Und wenn Sie sich **irgendwo zwischen Einsteiger und Profi** einschätzen, arbeiten Sie das Buch ganz einfach von vorne nach hinten durch!

Da dieses Buch aber nicht zwingend von vorne nach hinten durchgelesen werden muss, sondern Sie sich einzelne Kapitel nach Interesse herausuchen können, finden Sie die jeweils relevanten Literaturangaben und Internet-Links am Ende jedes Kapitels. Zusammen mit den Links aus dem Fließtext finden Sie die Webadressen auch auf meiner Webseite zum Buch (s. u.), wo sie regelmäßig aktualisiert werden. Die Links sind trotzdem im Text enthalten, damit Sie einen Anhaltspunkt haben, wonach Sie bei Google suchen könnten. Und damit Sie beim Lesen einzelner Kapitel nicht ständig blättern müssen, werden einige grundlegende Informationen vereinzelt mehrfach aufgeführt.

Welche Versionen werden behandelt?

Der Fokus des Buchs liegt auf dem derzeit aktuellen Mac OS X 10.3 (»Panther«) mit Java 1.3 und Java 1.4. Bei allen Themen sind wichtige Unterschiede zu Mac OS X 10.2 (»Jaguar«) dokumentiert, damit Sie ältere Programme verstehen und gegebenenfalls auf den neuesten Stand bringen können. Hinweise für noch ältere Systeme tauchen nur vereinzelt auf, da die Versionen bis einschließlich Mac OS X 10.1 im praktischen Einsatz kaum noch von Bedeutung sind.

So weit wie derzeit möglich ist bei den Angaben sichergestellt, dass die Informationen auch für das kommende Mac OS X 10.4 (»Tiger«) gültig sind, das Java 1.5 (bzw. J2SE 5.0, wie die Version neuerdings auch heißt) enthalten wird. Im Anhang finden Sie eine kurze Einführung zu Java 1.5.

Beispielprogramme

Die Quelltexte sind nicht immer vollständig abgedruckt, beispielsweise wenn es sich um Standardcode handelt, der nichts Besonderes zum jeweiligen Beispiel beiträgt. Die entsprechenden Stellen sind mit `// . . .` gekennzeichnet, aber auf der beiliegenden CD sind die Quelltexte natürlich komplett enthalten. Ebenso wurden im Buch die Quelltext-Kommentare entfernt, da die Beispiele im Fließtext beschrieben werden.

Im Buch wird bewusst kein durchgehendes Beispielprogramm verwendet, und aufbauende Beispiele werden nur dort eingesetzt, wo es einen wirklichen Mehrwert bringt. Am häufigsten finden Sie in den Abschnitten also unabhängige Beispiele, wodurch aber auch der Struktur des Buches Rechnung getragen wird und Sie die Kapitel weitestgehend unabhängig voneinander lesen können.

Die Beispiele sind bewusst einfach gehalten. Als Profi kennen Sie sowieso größere Projekte und möchten nur den Mac-relevanten Code kennen lernen, und als Einsteiger würden Sie bei größeren Beispielen zu leicht den Überblick verlieren. Daher verwenden die meisten Programme z.B. auch keine aufwändigen Oberflächen (abgesehen natürlich von denen im Kapitel über Benutzungsoberflächen), damit Sie schnell das Wesentliche erkennen und einfach aus dem Quelltext heraus kopieren können.

Wo Sie die Beispiele auf der beiliegenden Buch-CD finden und was darauf sonst noch enthalten ist, wird im Anhang kurz beschrieben. Alternativ lassen Sie sich einfach die Datei `index.html` aus dem CD-Wurzelverzeichnis in einem Web-Browser anzeigen.

Schriftdarstellung

Dieses Buch verwendet folgende Konventionen für die Schriftdarstellung der unterschiedlichen Textbestandteile:

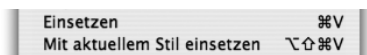
Textbestandteil	Darstellung
Programmquelltext (Listings)	<code>/* Schrift mit fester Zeichenbreite */</code>
Java-Bezeichner (Variablen, Methoden, Klassen)	Schrift mit fester Zeichenbreite: <code>meinObjekt</code> <code>indexOf()</code> <code>String</code>
Dateinamen und Pfadangaben	Schrift mit fester Zeichenbreite: <code>/Programme/TextEdit</code>

Verwendete Schriftkonventionen

Textbestandteil	Darstellung
Befehle und Programm- ausgaben	Schrift mit fester Zeichenbreite: java -version
Programmbeispiele auf der CD, hier »HalloWelt« in Kapitel 1	Schrift mit fester Zeichenbreite (es handelt sich um Pfadangaben auf der CD): //CD/examples/ch01/hallo/HalloWelt.java
Web-Adressen (URLs)	<i>Kursivschrift:</i> <i>http://www.muchsoft.com/java/</i>
Menüs und Menüeinträge, Texte auf dem Bildschirm	Menütitel • Menüeintrag • Untermenüeintrag usw.: Bearbeiten • Kopieren Ablage • Sichern unter... Einzelne Menüeinträge und Texte auf dem Bildschirm werden auch in Anführungszeichen gesetzt.
Tastaturkürzel (Shortcuts)	Die Umschalttasten werden ausgeschrieben. Einzelne Buch- staben werden großgeschrieben, \square ist dabei aber nur zu drücken, wenn dies auch in der Tastenkombination steht. Sonstige Aktionen werden normal dargestellt: $\square + \square$ $\square + \square$ So angegebene Tastaturkürzel sind zusammen auszuführen (bzw. erst die Umschalttasten, dann der Rest).

Verwendete Schriftkonventionen (Forts.)

Der Mac besitzt diverse Umschalttasten mit verschiedenen Bezeichnungen: Shift, Ctrl (Steuerung), Alt (Wahl bzw. Option), Apfel (Command / Befehl bzw. Propeller). In Abbildung 0.1 sehen Sie beispielsweise den Menüeintrag »Einsetzen« mit dem Tastaturkürzel $\square + \square$, darunter den Eintrag »Mit aktuellem Stil einsetzen« mit dem Kürzel $\square + \square + \square + \square$. Die Umschalttaste \square wird beim Mac eher selten bei Menüeinträgen benutzt, dort würde dann das Dach (^) auftauchen.



Symbole der Umschalttasten

Danksagung

Ein herzliches Dankeschön gilt dem gesamten Team von Galileo – ohne sie würden Sie dieses Buch schließlich nicht in den Händen halten! Ganz besonderem Dank bin ich dabei meinem Lektor Stephan Mattescheck verpflichtet – für die gute Betreuung und die unendliche Geduld, die ein Lektor wohl immer bei solchen Projekten aufbringen muss. Bedanken möchte ich mich auch beim Mac-Stammtisch Karlsruhe (<http://www.mac-ka.de/>), bei den Schwarzwäldern

und bei vielen Verwandten und Bekannten für ihr beständiges Interesse am Fortschritt (»Na, was macht das Buch? Immer noch nicht fertig?« ;-)) – durch ihre Fragen wurde mein Augenmerk immer wieder auf kleine, wichtige Details gelenkt, die es zu berücksichtigen galt. Und nicht zuletzt ein ganz liebes Dankeschön an Gesine, die das Buch von Anfang an begleitet und mich mit Kreativität und Motivation unterstützt hat!



Kontakt

Wenn Sie Fragen, Anregungen oder Verbesserungsvorschläge haben, schreiben Sie mir an macjava@muchsoft.com oder schicken Sie Ihre Post an den Verlag Galileo Press. Schauen Sie aber am besten vorher auf meiner privaten Webseite zum Buch (<http://www.muchsoft.com/java/>) vorbei, ob Ihre Fragen und Wünsche dort vielleicht schon beantwortet sind. Und natürlich können Sie auch die Verlagsseite <http://www.galileocomputing.de/> besuchen, wo gegebenenfalls Online-Aktualisierungen für das Buch zur Verfügung stehen.

Zum Schluss bleibt mir nur noch, Ihnen viel Spaß beim Lesen und Ausprobieren zu wünschen – viel Erfolg mit der Java-Programmierung auf und für Mac OS X!

Thomas Much

Karlsruhe, im Oktober 2004

1 Grundlagen

1.1	Historisches	22
1.2	Aktuelles	24
1.3	Den Mac richtig bedienen	25
1.4	Eingeben, übersetzen, ausführen	35
1.5	Entwicklerwerkzeuge und -dokumentation	42
1.6	Up to date bleiben	44
1.7	Mac OS X erkennen	47
1.8	Konfigurationsdateien speichern	52
1.9	Aufbau des Apple-Java-Systems	55
1.10	Kleine Fallen	69
1.11	Literatur & Links	70

1 Grundlagen

2 Entwicklungsumgebungen

3 Grafische Benutzungsoberflächen (GUI)

4 Ausführbare Programme

5 Portable Programmierung

6 Mac OS X-spezielle Programmierung

7 Grafik und Multimedia

8 Werkzeuge

9 Datenbanken und JDBC

10 Servlets und JavaServer Pages (JSP)

11 J2EE und Enterprise JavaBeans (EJB)

12 J2ME und MIDP

A Kurzeinführung in die Programmiersprache Java

B Java auf Mac OS 8/9/Classic

C Java 1.5 »Tiger«

D System-Properties

E VM-Optionen

F Xcode- und Project Builder-Einstellungen

G Mac OS X- und Java-Versionen

H Glossar

I Die Buch-CD

1 Grundlagen

*»Beginne am Anfang«, sagte der König ernst, »und fahre fort, bis du ans Ende kommst: Dann höre auf.«
(Lewis Carroll)*

In diesem Buch treffen zwei Welten aufeinander, die zwar perfekt zueinander passen, die aber bisher oft noch nichts voneinander mitbekommen haben – die Welt der Apple-Macintosh-Programmierer, die bisher wenig mit Java zu tun hatten, und die Welt der Java-Programmierer, die bisher meistens unter Windows oder Linux entwickelt und Apple-Rechner und deren Betriebssystem MacOS X als »schönes Spielzeug« abgetan haben. Egal, welcher der beiden Welten Sie angehören – in diesem Grundlagen-Kapitel bekommen Sie eine schnelle und einfache Einführung sowohl in Mac OS X als auch in Java.

Als Umsteiger werden Sie sich sicherlich speziell für die Mac-Besonderheiten interessieren. Und da Sie für dieses für Sie neue System Software entwickeln wollen, müssen Sie diese Besonderheiten verstehen, damit Sie wie ein langjähriger Mac-Anwender denken lernen. Dazu gehört unter anderem die grundlegende Philosophie des Systems (Maus, Menü, Fenster), ebenso wie allgemein übliche Tastenkürzel und Bedienweisen. `java` und `javac` sind für Sie nichts Neues, aber wir fangen wie üblich trotzdem damit an, damit Sie einen schnellen Überblick über das Java-System von Mac OS X bekommen – danach werden Sie dann vermutlich sowieso ein bisschen selbst herumprobieren. Die bekannten sowie Mac-spezifische Entwicklungsumgebungen behandelt das nächste Kapitel.

Als Mac-Anwender kennen Sie die Bedienung des Systems bereits – aber wenn auch Sie ein »kleiner« Umsteiger von Mac OS 8 bzw. 9 oder Mac OS X 10.1 sind, werden Sie hier sicherlich noch einige nützliche Tipps entdecken. Im Wesentlichen möchten Sie aber Java kennen lernen. Deshalb eine kleine Warnung vorweg: Dies ist kein reines Java-Einsteigerbuch! Im Anhang gibt es zwar eine Kurzeinführung in die Sprache Java. Wenn Sie aber noch keine Programmiersprache beherrschen oder anhand vieler Beispiele lernen möchten, wie man in Java `for`-Schleifen programmiert, sollten Sie zusätzlich ein weiteres Buch zu Rate ziehen. Wenn Sie dagegen schon programmieren können, Java vielleicht vor einigen Jahren in der Schule oder an der Uni gesehen haben oder eine neue Sprache am besten mit kleinen Beispielprogrammen kennen lernen, werden Sie sich hier prima zurecht finden. Dementsprechend geht es in diesem Kapitel mit dem bekannten »Hallo Welt« los, mit dem Sie alle wichtigen Schritte bis zum fertigen Java-Programm einmal sehen werden.

Doch zunächst einmal starten wir mit einem kurzen Abriss der Entwicklungsgeschichte von Java im Allgemeinen und von Java auf dem Macintosh im Speziellen.

1.1 Historisches

Als **Sun** die Java-Technologie im Jahre 1995 ankündigte, war der Internet-Boom gerade dabei, so richtig loszulegen. Es gab mittlerweile grafische Browser für die wichtigsten Betriebssysteme, um Webseiten plattformübergreifend darzustellen, aber die Software musste immer noch für jedes System einzeln entwickelt werden. Man suchte also nach einer Lösung, um Programme für alle Systeme zusammen nur noch ein einziges Mal schreiben und übersetzen zu müssen, und Java kam dafür wie gerufen.

Anfang 1996 veröffentlichte Sun **Java 1.0**, und schon damals war alles Nötige für den »Write once, run anywhere«-Ansatz dabei: die Programmiersprache Java mit einer ziemlich exakten Spezifikation; ein passender Compiler, um aus Java-Quelltexten ausführbaren Java-Bytecode zu generieren; die Java Virtual Machine (auch JVM oder Java VM genannt) als »Interpreter« oder besser Laufzeitumgebung zur Ausführung des Java-Bytecodes; und schließlich die Klassenbibliotheken, die es überhaupt erst ermöglichen, plattformunabhängig Benutzungsoberflächen, Netzwerkzugriffe usw. zu realisieren. Mit dieser Version konnte man schon gut erahnen, welche Möglichkeiten Java bieten würde, aber wie bei jeder Version 1.0 gab es viele kleine Macken – und eine große: die recht gemächliche Ausführungsgeschwindigkeit.

Anfang 1997 war dann **Java 1.1** verfügbar. Es gab zur Geschwindigkeitsverbesserung jetzt einen Just-in-time-Compiler (JIT), der Java-Programme während der Ausführung nebenbei in Maschinencode übersetzen konnte. Außerdem wurde die Programmiersprache selbst ein bisschen aufgeräumt und erweitert.

Seit Ende 1998 gibt es **Java 1.2**, und da diese Version eine enorme Vergrößerung und Verbesserung der Klassenbibliotheken mitbrachte (z.B. Swing als umfangreiche Oberflächen-Bibliothek oder die Collections API für dynamische Datenstrukturen), spricht Sun seitdem von der »Java 2 Plattform«. Im Jahre 2000 kam dann **Java 1.3** heraus und brachte die »HotSpot« Virtual Machine mit, die deutlich besser just in time übersetzen konnte. Seitdem braucht sich Java geschwindigkeitsmäßig nicht mehr hinter C++ verstecken. **Java 1.4** wurde Ende 2002 fertig gestellt. Neben einer neuen, schnellen I/O-Bibliothek wurden zahlreiche bis dahin externe Pakete in die Standardbibliotheken integriert, darunter solche zur XML-Verarbeitung sowie reguläre Ausdrücke zur schnellen Textsuche.

Java bzw. das JDK (»Java Development Kit«) wurde von Sun zunächst nur für Solaris und Windows, seit 2001 dann auch für Linux zur Verfügung gestellt (ebenso das JRE, das »Java Runtime Environment«, das bei Endanwendern installiert werden kann und entsprechend die Entwicklungswerkzeuge nicht enthält). Als Java das Licht der Welt erblickte, verkaufte **Apple** gerade die letzten 68k-Computer (mit 68030- bzw. 68040-Prozessoren) und hatte den Umstieg auf die PowerPC-Architektur (PPC, die ersten Generationen der heutigen G4- und G5-Prozessoren) fast abgeschlossen. Aktuell war zu der Zeit Mac OS 7. Von Anfang an hat Apple die Entwicklung des Java-Systems für das Mac OS selbst in die Hand genommen (bzw. nehmen müssen) – trotz guter Kontakte zu Sun war der Apple-Marktanteil damals einfach zu klein, als dass Apple von Sun direkt unterstützt wurde.

Mit etwas über einjähriger Verzögerung erschien Anfang 1997 **MRJ 1.0** (»Macintosh Runtime for Java«), was Suns Java 1.0.2 entsprach. Es war Bestandteil von Mac OS 8 und war für 68k- und PPC-Rechner erhältlich. Bereits ein halbes Jahr später kam MRJ 1.5 auf den Markt, das einen JIT-Compiler und die Programmierschnittstelle MRJToolkit mitbrachte, aber Java-seitig blieb es noch beim alten JDK 1.0.2. Das änderte sich Ende 1997 (am 31.12., Apple hatte es noch für 1997 versprochen) mit **MRJ 2.0**, nun war endlich Java 1.1 (genauer das JDK 1.1.3) auf dem Mac verfügbar. Und dabei blieb es für Mac OS 8 und 9 (die so genannten »Classic«-Systeme) im Wesentlichen bis heute. Seit MRJ 2.2 ist das JDK auf dem Stand 1.1.8 (der letzten Sun-Version von Java 1.1), aber Java 1.2 oder neuer gibt es für das alte Mac OS nicht.

Dann kam Mac OS X (das X wird »Ten« gesprochen), Apples nächste Betriebssystemgeneration. Auf UNIX basierend, mit einer neuen, grafischen Benutzungsoberfläche, die trotzdem wieder die Schlichtheit und einfache Benutzbarkeit bieten sollte – Apple hatte sich viel vorgenommen. Im Herbst 2000 wurde eine »Public Beta« verteilt, im Frühjahr 2001 war dann **Mac OS X 10.0** (Code-name »Cheetah«) erhältlich – und Java 1.3 (das JDK bzw. JRE 1.3.0) wurde fertig installiert mitgeliefert! Intern wurde das komplett neu entwickelte Java-System MRJ 3.0 genannt, aber Apple verabschiedet sich nach und nach von dem Namen MRJ, um sich dem Sun-Versionsschema anzupassen. Leider merkte man diesem System seinen 1.0-Status an, entsprechend wird **Mac OS X 10.1** (»Puma« – Apple mag Raubkatzen), das im Herbst 2001 erschien, von vielen als erste benutzbare Mac OS X-Version angesehen. Enthalten war darin auch das JDK 1.3.1, das es zuvor schon als separates Update gegeben hatte.

Im Jahre 2002 wurde dann das nächste größere System-Update, **Mac OS X 10.2** (»Jaguar«) veröffentlicht. Ab Werk war immer noch das JDK 1.3.1 dabei, Java 1.4.1 konnte dann 2003 als Update installiert werden. Bei **Mac OS X 10.3**

(»**Panther**«), das seit Herbst 2003 verkauft wird, sind beide JDKs (1.3.1 und 1.4.1) bereits vorinstalliert.

Von Anfang an hat Apple auch immer eigene Java-Entwicklertools und -Dokumentationen kostenlos zur Verfügung gestellt. Früher war dies das MRJ SDK mit einer Sammlung kleiner Tools, bei Mac OS X ist dies eine komplette Entwicklungsumgebung.

MRJ 2.2.x kann heute immer noch auch unter Mac OS X verwendet werden, und zwar in der »Classic«-Umgebung, wo dann ein komplettes Mac OS 9 als separater Mac OS X-Prozess läuft. Da aber mittlerweile kaum noch jemand für Java 1.1 oder das alte MacOS entwickelt, wird dieses Thema nur kurz im Anhang behandelt.

1.2 Aktuelles

Für die Betriebssysteme Solaris, Linux und Windows steht auf <http://java.sun.com/> mittlerweile **Java 1.5** bzw. die **J2SE 5.0** zur Verfügung (»Java 2 Platform Standard Edition« Version 5.0 – Sun lässt bei den Versionsnummern nun die »1« weg). Die neue Version integriert unter anderem die »Generics« (grob vergleichbar mit C++-Templates) in die Sprache Java – im Anhang finden Sie einen kurzen Überblick über die neuen Möglichkeiten.

Mac OS X 10.2 hat Updates bis auf Version 10.2.8 erfahren, bringt Java 1.3.1 mit und kann mit dem Java 1.4.1-Update erweitert werden (vermutlich wird dies auch die letzte Java-Version für diese Systemversion sein). Als Entwicklungsumgebung stellt Apple den »Project Builder« zur Verfügung.

Mac OS X 10.3 bringt von Anfang an Java 1.3.1 und 1.4.1 mit. Apples neue Entwicklungsumgebung hierfür sind die »Xcode Tools«, eine Weiterentwicklung des Project Builders. Mit einigen Monaten Verspätung steht für dieses System nun auch das JDK 1.4.2 als Update zur Verfügung, ebenso wie die Erweiterungen »Java 3D« und »Java Advanced Imaging« (JAI).

Im ersten Halbjahr 2005 soll dann **Mac OS X 10.4 (»Tiger«)** erscheinen. Welche Java-Versionen mitgeliefert werden, ist derzeit noch nicht abzusehen. Es ist aber sehr wahrscheinlich, dass Apple das aktuelle Java 1.5 fest ins Betriebssystem integriert.

Apple liefert nicht nur das »normale« Mac OS X für Client-Systeme aus, sondern auch die Server-Variante **Mac OS X Server**. Da sich beide Varianten aus Java-Sicht nicht wesentlich unterscheiden, wird in diesem Buch nur bei Bedarf auf Mac OS X Server hingewiesen. Die wichtigsten Unterschiede betreffen die Einstellungen der Java Virtual Machine, die im Anhang-Kapitel über die VM-

Op-tionen besprochen werden. Weitere Informationen zu Mac OS X Server finden Sie auf der Seite <http://www.apple.com/server/macosx/>.

Was Sie am historischen Überblick und an den aktuellen Daten sehen können: Da das Java-System von Mac OS X eine Eigenentwicklung von Apple ist (auch wenn diese zunehmend mit den Sun-Sourcen synchronisiert ist), gibt es immer gewisse Verzögerungen, bis eine neue Java-Version auch für den Mac angeboten wird. Der Zeitabstand zwischen dem öffentlichen Sun-Release und Apples Mac OS X-Anpassung ist meistens gar nicht so groß. Die Lücke wirkt allerdings deutlich länger, da Sun oft Monate vor dem öffentlichen Release Entwickler-Vorabversionen freigibt, und diese sind in der Regel nicht für den Mac erhältlich.

Wenn Sie damit leben können, nicht immer die allerneuesten Java-Vorabversionen zur Verfügung zu haben, besitzen Sie mit Mac OS X nicht nur ein hervorragendes System für den Einsatz Ihrer Java-Applikationen, sondern Sie können damit genauso gut Java-Software entwickeln, die dann natürlich auch wieder auf anderen Plattformen eingesetzt werden kann! Und bevor wir nun gleich mit der eigentlichen Java-Programmierung beginnen, lernen Sie erst noch den Mac und seine Oberfläche besser kennen.

1.3 Den Mac richtig bedienen

Wenn die grafische Benutzeroberfläche (GUI, »Graphic User Interface«) von Mac OS X neu für Sie ist, nehmen Sie sich bitte die Zeit, sich in die Mac-Philosophie ein bisschen einzudenken. Denn auch wenn Desktop-Rechner heutzutage meistens mit einer grafischen Oberfläche bedient werden, gibt es doch zum Teil erhebliche Unterschiede – nicht nur im Aussehen, sondern auch in der Funktionsweise. Apple hat eine über 20-jährige Erfahrung mit grafischen Benutzerschnittstellen (1983 kam der Lisa-Computer auf den Markt, 1984 der erste Macintosh), und in dieser Zeit wurden die Konzepte oft getestet und in vielen nützlichen Kleinigkeiten verbessert. Das Problem dabei ist: Wenn Sie bisher Oberflächen wie Windows oder Linux-KDE gewohnt sind, haben Sie sich wahrscheinlich schon an deren »Denkweise« gewöhnt und kommen manchmal nicht mehr darauf, wie man den Schreibtisch (Desktop) intuitiv bedienen könnte (das ist nicht negativ gemeint, das entspricht einfach nur regelmäßigen Erfahrungen von Umsteigern).

Da Sie Java-Anwendungen entwickeln wollen, die auch von langjährigen Mac-Benutzern als »normale« Mac-Applikation erkannt und bedient werden sollen, finden Sie in diesem Abschnitt die wichtigsten Ideen und Begriffe rund um die aktuelle Mac-Oberfläche, damit Sie diese auch in Java berücksichtigen können.

Vor Mac OS X sah die Mac-Oberfläche deutlich anders aus, daher ist es auch für Umsteiger von solchen Systemen ratsam, das Folgende wenigstens zu überfliegen. Und falls Sie sich bisher generell geweigert haben, die »Designer-Oberfläche« vom Mac anzufassen, hier noch eine kleine Beruhigung: Gerade in Mac OS X 10.3 hat Apple einige Funktionen zusätzlich eingebaut, die Ihnen als Anwender von z.B. Windows eine etwas gewohntere Umgebung präsentieren.

Zunächst einmal gibt es seit der Einführung von Mac OS X eine vernünftige Benutzerverwaltung. Wenn Sie Mac OS X selbst installiert haben, haben Sie sich ein Benutzerkonto mit Namen und Passwort eingerichtet, mit dem Sie sich nach dem Rechnerstart anmelden können. Es ist auch möglich, einen bestimmten Benutzer automatisch anmelden zu lassen, aber aus Sicherheitsgründen sollten Sie dies in den Systemeinstellungen deaktivieren. Die Systemeinstellungen finden Sie normalerweise unten am Bildschirmrand im so genannten »Dock«, in das Sie übrigens beliebige Anwendungen und (rechts vom Trennstrich) Dokumente ziehen können, die dort dann als Verknüpfung abgelegt werden. In den Systemeinstellungen können Sie dann im Bereich »Benutzer« unter »Anmelde-Optionen« festlegen, dass kein Benutzer automatisch angemeldet wird. In Zukunft werden Sie dies teilweise abgekürzt als **Systemeinstellungen · Benutzer · Anmelde-Optionen** lesen. Man kann das Dock unter **Systemeinstellungen · Dock** übrigens auch so konfigurieren, dass es beispielsweise rechts am Bildschirmrand erscheint.



Abbildung 11 Das Dock und die Systemeinstellungen

Nach der Anmeldung (dem Login) sehen Sie den »Finder« samt Desktop. Mit dem Finder verwalten Sie das Dateisystem und navigieren durch die Verzeichnishierarchie. Er ist grob mit dem Datei-Explorer anderer Systeme vergleichbar, wir sehen ihn uns gleich noch genauer an.

Oben am Bildschirmrand befindet sich die **Menüleiste**. Es gibt immer nur eine einzige sichtbare Menüleiste, und zwar die der gerade aktiven (obersten) Applikation. Wenn eine andere Anwendung aktiv wird, schaltet Mac OS X die Menüleiste entsprechend um. Innerhalb von Fenstern haben Menüleisten bei Mac-Applikationen nichts zu suchen, dort tauchen höchstens Toolbars auf.

Ganz wichtig, Sie sehen es bereits: Es kann eine Menüleiste geben, ohne dass die zugehörige Applikation ein Fenster geöffnet hat! Wenn Sie ein Fenster schließen, schließen Sie im Normalfall wirklich nur das Fenster und beenden nicht gleichzeitig die Anwendung. Dies ist erfahrungsgemäß fast die größte Hürde für Windows-Anwender. Denken Sie daran, Ihre Applikationen zu beenden! Apple geht allerdings nach und nach dazu über, Programme, die nur aus einem einzigen Dialogfenster bestehen (»grafische Tools«), beim Schließen dieses Dialogs automatisch zu beenden, beispielsweise die Systemeinstellungen.

Ganz links finden Sie in jeder Menüleiste das **Apfel-Menü**. Dies gehört nicht der Applikation, sondern dem System, dementsprechend müssen Sie sich darum bei der Programmierung auch nicht kümmern. Im Apfel-Menü haben Sie Zugang zu den Systemeinstellungen, können sich als Benutzer abmelden, den Rechner ausschalten usw.

Direkt daneben liegt immer das **Programm-Menü**, das den Namen der aktiven Applikation trägt (daran erkennen Sie am einfachsten, welche Anwendung gerade aktiv ist!). Alle Tastatureingaben – sofern sie nicht vom System abgefangen werden, z.B. spezielle Tastenkürzel – landen bei der aktiven Applikation. Im Programm-Menü können Sie das aktive Programm ausblenden (dann sind weder Fenster noch die Menüleiste sichtbar), beenden und den Einstellungen-Dialog aufrufen. Dieser Menüpunkt befindet sich immer hier und nicht in irgendeinem Extras-Menü! Gleiches gilt für die Programminformationen »Über *Programmname*«, die auch hier (und nicht im Hilfe-Menü) abzurufen sind.

Beim Finder sehen Sie im Programm-Menü, dass er eine besondere Stellung im System hat – er wird automatisch gestartet und kann nicht beendet werden. Dies macht Sinn, denn er kümmert sich ja um den Desktop, und dieser ist immer im Hintergrund zu sehen. Mit dem Desktop können Sie auch jederzeit zum Finder umschalten, indem Sie einfach auf den Desktop-Hintergrund klicken. Wenn die gerade aktive Applikation offene Fenster hat, werden diese inaktiviert, anschließend wird der Finder zur obersten Applikation.

Generell können Sie jede Applikation aktivieren, indem Sie auf deren Icon (Programmsymbol) im Dock klicken. Im Dock sieht man an einem kleinen Dreieck unter dem Programmsymbol, dass die Anwendung derzeit läuft und man durch Anklicken des Symbols sofort zum Programm umschaltet (nebenbei können Sie damit auch ausgeblendete Programme wieder einblenden). Fehlt das Dreieck, wird die Anwendung beim Anklicken erst noch gestartet. Wenn Sie lieber mit der Tastatur als mit der Maus arbeiten, können Sie auch mit $\text{⌘} + \text{⌘}$ zwischen den laufenden Applikationen wechseln. Bei Mac OS X 10.3 wird der Wechsel groß auf dem Bildschirm angezeigt, bei 10.2 nur klein im Dock.

Zur Verbesserung des $\text{⌘} + \text{⌘}$ -Programmwechsels wird – gerade unter Mac OS X 10.2 – gerne die Software »LiteSwitch« verwendet: <http://www.proteron.com/liteswitchx/>.

Sie haben gerade ein Tastaturkürzel (Shortcut) gesehen – auch ein Mac kann mit der Tastatur bedient werden, selbst wenn die Oberfläche zunächst einmal sehr Maus-lastig ist. An folgende Namen von Umschalttasten sollten Sie sich gewöhnen: ⌘ (auch Propeller- oder Befehlstaste genannt), ⌥ (Option- oder Wahl-taste), Ctrl (Steuerung) und ⇧ (Umschalttaste). Mobile Rechner haben zusätzlich noch eine Fn -Taste, die aber für Programm-Shortcuts nicht relevant ist. Die wichtigsten Kürzel sind wohl $\text{⌘} + \text{W}$ zum Schließen des obersten (aktiven) Fensters und $\text{⌘} + \text{Q}$ zum Beenden der aktiven Applikation.

Vielleicht sind Sie es von Ihrem System gewohnt, dass Sie sich mit ⌘ durch alle Elemente eines Dialogs bewegen können. Beim Mac OS bewegt man sich damit eigentlich nur durch die Texteingabefelder. Sie können aber die Option »Tastatursteuerung einschalten« in **Systemeinstellungen · Tastatur&Maus · Tastatur-Kurzbefehle** aktivieren, dann sind alle Elemente eines Dialog mit ⌘ bzw. den Pfeiltasten erreichbar. Ausgeführt wird das markierte Element mit der Leertaste.

Zurück zur Menüleiste und den typischen Menüs. Neben dem Programm-Menü sollte sich immer das **Ablage-Menü** mit den Dateioperationen befinden, daneben das **Bearbeiten-Menü**. Hier tauchen die Menüeinträge »Ausschneiden«, »Kopieren« und »Einsetzen« mit den Kürzeln $\text{⌘} + \text{X}/\text{C}/\text{V}$ auf. Beachten Sie, dass Windows mangels ⌘ -Taste stattdessen einfach Strg (Control) verwendet. Bei Java-Programmen können Sie glücklicherweise mit nur geringem Aufwand die Entscheidung »Apfel- oder Strg-Taste?« dem System überlassen. Andere Standardeinträge in diesem Menü sind »Widerrufen« (Undo) und »Alles auswählen«.

Am rechten Ende der Menüleiste gibt es üblicherweise ein **Fenster-Menü** mit allen offenen Fenstern der aktiven Applikation sowie das **Hilfe-Menü**.

Sehen wir uns den **Desktop** noch etwas genauer an. Am rechten oberen Rand liegen normalerweise alle angemeldeten Laufwerke (das können auch die einzelnen Partitionen – beim Mac »Volumes« genannt – einer Festplatte sein). Sollten diese dort nicht auftauchen, können Sie die Anzeige über den Menüpunkt **Finder • Einstellungen** einschalten (siehe Abbildung 1.2).

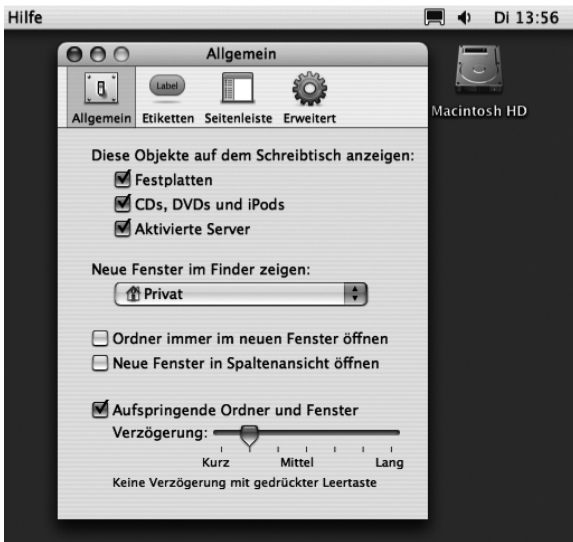


Abbildung 1.2 Finder-Einstellungen

Auf dem Desktop können Sie alles Mögliche ablegen (so wie Sie es von Ihrem »echten« Schreibtisch gewohnt sind ;-). Per Drag & Drop mit der Maus können Sie hierhin Dateien aus beliebigen Verzeichnissen ziehen, Text in einem Fenster selektieren und als Text-Clip auf den Desktop ziehen, URLs aus der Browser-Adressleiste als URL-Clip ablegen ... Und natürlich können Sie solche Clips dann auch wieder zurück in die Anwendung ziehen (oder einfach doppelt anklicken).

Um im Dateisystem zu arbeiten, öffnen Sie ein Finder-Fenster durch Doppelklick auf ein Laufwerkssymbol. Wenn der Finder die aktive Applikation ist, können Sie auch **⌘ + N** drücken, um ein neues Fenster zu öffnen. Es können beliebig viele Fenster gleichzeitig offen sein, zwischen denen Sie dann hin- und herkopieren können. Wichtig: Der Mac kennt keine MDI-(Multiple Document Interface)-Dokumente! Es gibt also keine Fenster, die die eigentlichen Dokumentfenster enthalten. Alle Dokumente liegen in eigenständigen, »normalen« Fenstern.

Wenn Sie bei so vielen offenen Fenstern den Überblick verlieren, hilft zum einen das Fenster-Menü in jeder Menüleiste. Zum anderen können Sie ab Mac OS X 10.3 einfach **F9** drücken, womit Sie die Exposé-Funktion aktivieren – alle Fenster werden verkleinert nebeneinander auf dem Bildschirm angezeigt, und Sie können das gewünschte Fenster aus dieser Übersicht auswählen.



Abbildung 1.3 Finder-Fenster

Einen Ordner öffnen Sie durch einen Doppelklick im selben Fenster. Wenn Sie zusätzlich zum Klick die **⌘**-Taste gedrückt halten, geht der Ordner in einem neuen Fenster auf. Ein Doppelklick auf ein Programm startet dieses. Alternativ können Sie das Programmsymbol auch nur selektieren (z.B. mit einem Einfachklick oder den Pfeiltasten) und dann mit **⌘**+**↵** starten.

Für UNIX-Anwender: In Abbildung 1.3 sehen Sie das System-Wurzelverzeichnis aus Anwendersicht. Wenn Sie in der Shell `ls /` eingeben, tauchen auch die Ihnen geläufigen Verzeichnisse `/bin`, `/etc`, `/usr` usw. auf.

Apple liefert übrigens alle Mäuse immer noch mit nur einer Maustaste aus. Während dies Einsteigern entgegenkommt, wünschen Sie sich als Programmierer eventuell mehr Tasten und ein Scrollrad. Kein Problem: Schließen Sie einfach Ihre Lieblings-USB-Maus an, Mac OS X unterstützt die zusätzlichen Tasten, ohne dass Sie irgendwelche Treiber installieren müssen. Und bei einer Eintasten-Maus können Sie den Rechtsklick durch **Ctrl**+Klick emulieren.

Jetzt ist es leider an der Zeit, mit einem Mythos aufzuräumen ... Auch Mac-Programme können abstürzen! Und falls eine Applikation einmal »hängt«, können

Sie sie jederzeit im »Programme sofort beenden«-Dialog hart terminieren (»abschießen« oder »killen«), den Sie mit $\boxed{\text{⌘}} + \boxed{\text{⌘}} + \boxed{\text{ESC}}$ erreichen. Seit Mac OS X 10.3 können Sie die oberste Applikation direkt ohne Nachfrage mit $\boxed{\text{⌘}} + \boxed{\text{⌘}} + \boxed{\text{⌘}} + \boxed{\text{ESC}}$ aus dem Speicher entfernen. Aber Achtung: Bei beiden Methoden gehen geänderte Daten in den Anwendungen verloren!

Im Finder-Fenster »Macintosh HD« sehen Sie die Standardverzeichnisse des Systems, darunter `Programme` (hier befinden sich Applikationen für alle Anwender des Rechners) und `Benutzer` (hier hat jeder Anwender des Rechners sein eigenes, geschütztes Home-Verzeichnis). Falls Sie stattdessen die englischen Bezeichnungen `Applications` und `Users` sehen, sollten Sie in **Finder · Einstellungen · Erweitert** die Option »Alle Suffixe zeigen« ausschalten. Die Beispiele in diesem Buch verwenden die deutschen Namen, für das Funktionieren ist dies aber egal.

In den Verzeichnissen (und damit auch auf dem Desktop) können Sie auch Verknüpfungen (Aliase, Links) erzeugen. Ziehen Sie eine beliebige Datei einfach mit gedrückter $\boxed{\text{⌘}}$ - und $\boxed{\text{⌘}}$ -Taste in das Zielverzeichnis. Dort wird dann ein Symbol mit einem kleinen Pfeil an der unteren linken Ecke angezeigt. Achtung: Dies ist leider kein UNIX-Softlink! Softlinks erzeugen Sie nur mit `ln -s` in der Shell, die der Finder dann aber trotzdem als Alias verwenden kann.

Wenn Sie mehr über die Konzepte der Mac OS X-Benutzungsoberfläche (auch »Aqua« genannt) wissen möchten, sehen Sie sich bitte die ausführlichen »Human Interface Guidelines« von Apple an, die Sie im Web auf <http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/> finden.

Ein paar Anmerkungen noch zu Festplatten am Mac. Eine Festplatte kann in mehrere Volumes (Partitionen) aufgeteilt sein. Auf jedem Volume können Sie bei Bedarf ein komplettes Betriebssystem installieren und so unterschiedliche Mac OS X-Versionen zum Testen auf einem Rechner verfügbar haben. Das Volume des aktiven Betriebssystems wird **Startvolume** genannt und kann unter **Systemsteuerung · Startvolume** für die künftigen Rechnerstarts festgelegt werden. Wenn Sie dagegen nur einmalig von einem anderen Volume booten wollen, drücken Sie beim Rechnerstart einfach die Taste $\boxed{\text{C}}$, um von einer eingelegten CD-ROM zu starten. Wenn Sie stattdessen $\boxed{\text{⌘}}$ gedrückt halten, landen Sie im Boot-Manager, der Ihnen alle verfügbaren Systeme zur Auswahl anbietet.

Nach der Standardinstallation besitzt Ihr Rechner genau ein Volume mit dem Namen »Macintosh HD«. Obwohl Sie diesen Namen jederzeit ändern können, belassen wir es in diesem Buch dabei – mit »Macintosh HD« ist hier immer das Startvolume gemeint.

Wenn Sie Ihre Festplatte in mehr als ein Volume aufteilen wollen, geht das nur, wenn von dieser Platte gerade kein System gestartet wurde. Entweder partitionieren Sie also eine externe FireWire- oder USB-Festplatte, oder Sie teilen die Platte gleich bei der Installation von Mac OS X passend auf. Das System bringt dafür das Festplatten-Dienstprogramm mit, das Sie am Anfang der Mac OS X-Installation über das Installer-Menü starten können. Später können Sie das Festplatten-Dienstprogramm jederzeit im Dienstprogramme-Ordner innerhalb des Programme-Ordners aufrufen.

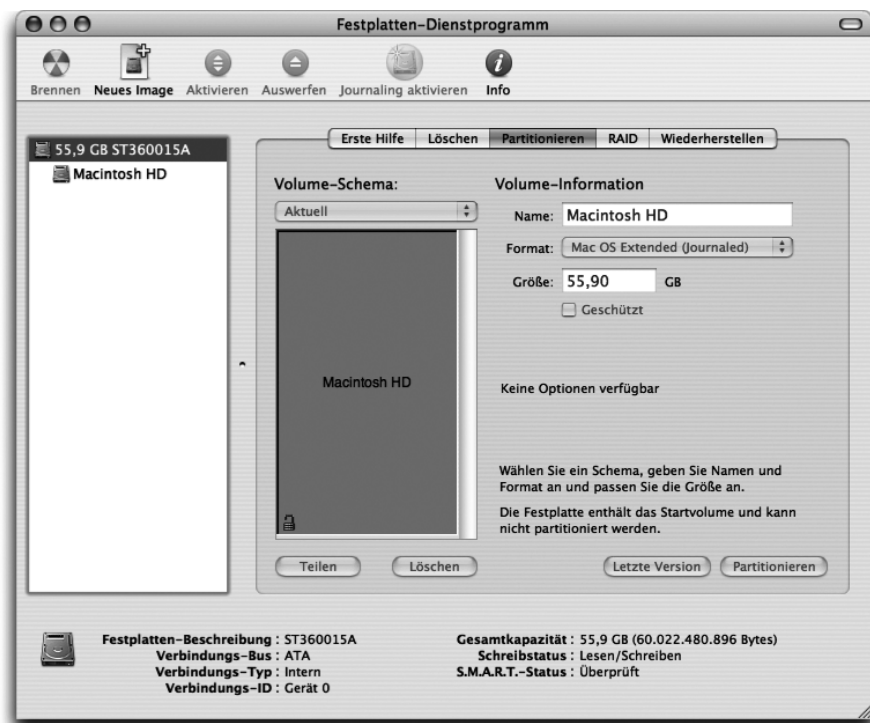


Abbildung 1.4 Festplatten-Dienstprogramm

Es ist übrigens eine gute Idee, vor jedem Wechsel zu einer neuen Mac OS X-Version die Datei-Zugriffsrechte des Volumes reparieren zu lassen. Einige Programme bringen diese leider ab und zu durcheinander, und dann geht ein Update manchmal schief ... Im Festplatten-Dienstprogramm wählen Sie dazu das Startvolume aus und klicken dann auf das Tab »Erste Hilfe«. Dort können Sie nun »Volume-Zugriffsrechte reparieren« starten.

Wenn Sie sich dafür interessieren, was »unter der Haube« passiert, bringt Mac OS X 10.3 auch dazu ein passendes Werkzeug mit. Mit dem Programm Aktivitäts-Anzeige aus dem Verzeichnis /Programme/Dienstprogramme

können Sie sich alle laufenden Prozesse übersichtlich anzeigen lassen, ebenso die Speicherauslastung, Festplattenaktivität usw. Alternativ können Sie sich diese Informationen im Terminal mit `top` und `ps -x` besorgen.

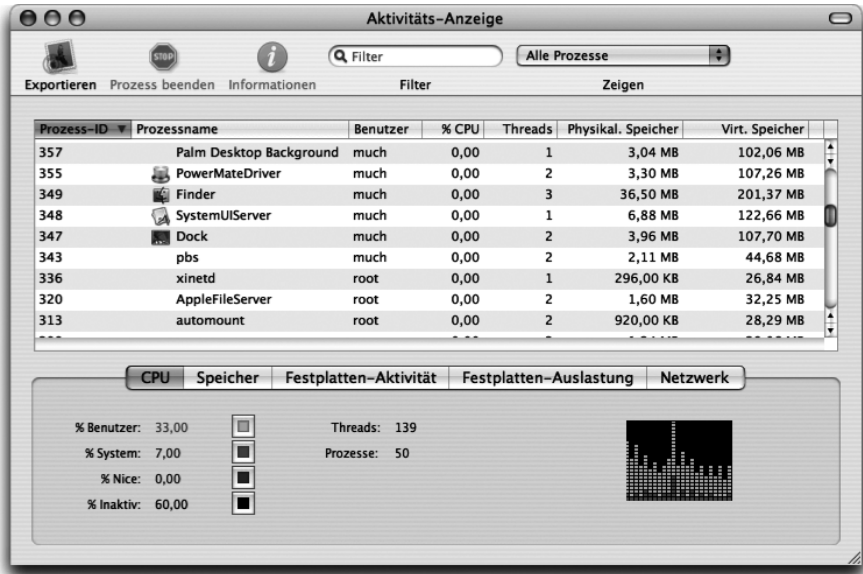


Abbildung 1.5 Aktivitäts-Anzeige

Hilfe finden

Wenn Sie weitere Hilfe zu Mac OS X benötigen, können Sie über das Finder-Hilfe-Menü die »Mac Hilfe« aufrufen (alternativ drücken Sie einfach `⌘+?`). Während diese – gerade ab Mac OS X 10.3 – für die Anwendung des Systems oft ausreichende Informationen liefert, enthält sie kaum Informationen zur Software-Entwicklung. Wo Sie diese erhalten, erfahren Sie später in diesem Kapitel (Abschnitte 1.5 und 1.6).

Tastenkürzel	Auswirkung
<code>⌘+A</code>	Alles auswählen
<code>⌘+C</code>	Markiertes Element in die Zwischenablage kopieren
<code>⌘+F</code> / <code>⌘+G</code>	Suchen/Weitersuchen

Tabelle 1.1 Die wichtigsten Mac OS X-Tastenkürzel

1 Eine Übersicht über viele weitere Tastenkürzel finden Sie auf der Seite <http://www.xshorts.de/>.

Tastenkürzel	Auswirkung
⌘ + H	Anwendung ausblenden (»Hide«)
⌘ + I	Informationen zum selektierten Element anzeigen
⌘ + M	Fenster ins Dock minimieren
⌘ + N	Neues Dokument öffnen
⌘ + O	Vorhandenes Dokument öffnen (laden)
⌘ + P	Drucken (»Print«)
⌘ + Q	Programm beenden
⌘ + S	Dokument speichern
⇧ + ⌘ + S	Sichern unter einem neuen Namen
⌘ + V	Inhalt der Zwischenablage an der Cursorposition einfügen
⌘ + W	Fenster schließen
⌘ + X	Markiertes Element in die Zwischenablage kopieren und danach das markierte Element löschen
⌘ + Z	Letzte Aktion widerrufen (»Undo«)
⌘ + ⌘	Wechsel zwischen laufenden Programmen
⌘ + ?	Hilfe
⌘ + ,	Einstellungen
⌘ + . bzw. Esc	Laufende Aktion oder angezeigten Dialog abbrechen
⌘ + ⌘ + Esc	Dialog »Programme sofort beenden« öffnen (vergleichbar mit dem Task Manager unter Windows)
⇧ + ⌘ + Esc	Aktives Programm ohne Nachfrage beenden. Daten werden nicht gesichert! Erst ab Mac OS X 10.3
⌘ + ←	Zum Zeilenanfang/-ende springen
⌘ + →	
⌘ + ←	Zum vorhergehenden/nächsten Wort springen
⌘ + →	
F9	Alle offenen Fenster im Überblick darstellen. Erst ab Mac OS X 10.3
F10	Alle offenen Fenster der aktiven Applikation im Überblick darstellen. Erst ab Mac OS X 10.3
F11	Alle Fenster ausblenden, Finder-Desktop anzeigen. Erst ab Mac OS X 10.3

Tabelle 1.1 Die wichtigsten Mac OS X-Tastenkürzel (Forts.)

Tastenkürzel	Auswirkung
<code>Ctrl</code> + <code>⏻</code>	Dialog »Computer jetzt ausschalten?« öffnen. Hier können Sie mit <code>⏻</code> den Rechner ausschalten, mit <code>R</code> (»Reboot«) neu starten und mit <code>S</code> (»Sleep«) in den Ruhezustand versetzen. <code>Esc</code> bricht den Dialog ab.
<code>Fn</code> + <code>⏻</code>	Entfernen (auf mobilen Rechnern)

Tabelle 1.1 Die wichtigsten Mac OS X-Tastenkürzel (Forts.)

1.4 Eingeben, übersetzen, ausführen

Nun ist es an der Zeit, Ihr erstes Java-Programm auf dem Mac zu starten! Dazu nehmen wir das allseits beliebte »Hallo Welt«-Programm, das eben diesen Text ausgibt. Während das Programm technisch nicht sonderlich interessant ist, können Sie mit den Schritten bis zur Ausführung das Java-System von Mac OS X schnell und einfach kennen lernen – nicht nur als Einsteiger, sondern auch als Java-Profi. Sollten Sie mit dem Quelltext selbst noch Probleme haben, lesen Sie sich bitte im Anhang die Kurzeinführung in die Sprache Java durch. Es handelt sich dort aber wirklich nur um eine extrem knappe Einführung, die Ihnen hilft, wenn Sie eine andere (vorzugsweise objektorientierte) Sprache beherrschen. Zum gründlichen Erlernen von Java sollten Sie zusätzlich ein anderes Buch lesen, beispielsweise den »Einstieg in Java« von B. Steppan.

Eine kleine Bemerkung noch vorneweg: Für alle Beispiele und Beschreibungen ist mindestens Mac OS X 10.2 Voraussetzung. Screenshots sind in der Regel mit Mac OS X 10.3 erstellt. Obwohl die meisten Programme – sofern sie mit Java 1.3 auskommen – auch unter Mac OS X 10.1 laufen dürften, ist dieses System zu alt und aus heutiger Sicht zu unausgereift, um hier speziell berücksichtigt zu werden.

Starten Sie den Standard-Texteditor von Mac OS X, TextEdit, der normalerweise das RTF-Format verwendet, aber ebenso Word-Dokumente laden und speichern und ganz normale Texte verarbeiten kann. Sie finden den Editor im Programme-Verzeichnis:

```
Macintosh HD
  Programme
    TextEdit
```

Wenn es der Übersichtlichkeit dient, werden Sie im Folgenden Pfadangaben auch in dieser Form finden, ansonsten wird die Kurzform `Macintosh HD/Programme/TextEdit` verwendet (oder nur `/Programme/TextEdit` – wenn ein UNIX-Pfad mit dem Schrägstrich beginnt, ist damit der Pfad ausgehend vom Startvolumen gemeint).

Geben Sie den Programmtext aus Abbildung 1.6 in das leere TextEdit-Fenster ein oder laden Sie den Quelltext `HalloWelt.java` mit dem Menüpunkt **Ablage · Öffnen...** (bzw. `⌘+O`) aus dem Verzeichnis `/examples/ch01/hallo/` von der Buch-CD. Wenn Sie die Datei laden, erkennt TextEdit automatisch, dass es sich um unformatierten Text handelt, ansonsten müssen Sie den Modus mit dem Menüpunkt **Format · In reinen Text umwandeln** umschalten.



Abbildung 1.6 »Hallo Welt« in TextEdit

Wenn Sie die roten Unterstreichungen stören, die in einem Programmquelltext nichts zu suchen haben, deaktivieren Sie die Rechtschreibprüfung unter **Bearbeiten · Rechtschreibung · Während der Texteingabe prüfen**. Die Rechtschreibkontrolle ist ein Systemdienst, den alle Anwendungen – auch Java-Programme! – nutzen können.

TextEdit ist ein einfacher, aber brauchbarer Editor mit grafischer Benutzungsoberfläche. Kleinere Programme kann man sicherlich damit tippen, aber ein ausgewachsener Quelltext-Editor ist es definitiv nicht. Als universeller Editor wird auf dem Mac gerne BBEdit oder der preiswertere TextWrangler von Bare Bones Software eingesetzt. Kostenlos ist BBEdit Lite erhältlich, und obwohl diese Version seit längerem nicht mehr weiterentwickelt wird, lässt sie sich nach wie vor gut und problemlos verwenden. Sie können BBEdit Lite von

<http://www.barebones.com/products/bblite/index.shtml>

oder direkt von

ftp://ftp.barebones.com/pub/freeware/BBEdit_Lite_612.smi.hqx

herunterladen.² Im nächsten Kapitel sehen Sie dann weitere und vor allem spezialisiertere Editoren und Entwicklungsumgebungen.

2 smi- und hqx-Dateien sind ältere Mac OS-Formate. Klicken Sie nach dem Herunterladen einfach doppelt auf das Archiv, dann wird es vom Stuffit Expander, der Mac OS X beiliegt, ausgepackt.

Wenn Sie aus der UNIX-Welt kommen und Ihnen grafische Oberflächen nur bedingt zusagen, können Sie das Terminal – das Sie in Kürze sehen werden – öffnen und in der Shell `vi`, `emacs` oder `pico` aufrufen.

Während Sie tippen, ist Ihnen vielleicht aufgefallen, dass im roten Fenster-Schließ-Knopf oben links ein Punkt angezeigt wird. Dies signalisiert, dass das Dokument verändert wurde und dass beim Schließen des Fensters nun nachgefragt wird, ob man den Inhalt speichern oder verwerfen möchte.

Speichern Sie den Quelltext nun als `HalloWelt.java` in Ihrem Benutzer-Verzeichnis (Home) ab, am einfachsten mit **Ablage · Sichern** (oder **Ablage · Sichern unter...**, was bei neuen Dokumenten aber auf dasselbe hinaus läuft). Im erscheinenden Sichern-Dialog können Sie mehr Informationen einblenden, indem Sie auf den Knopf mit dem Dreieck (oben rechts) klicken. Das Home-Verzeichnis wird mit Ihrem Kurznamen bezeichnet, hier im Beispiel also »much«, das Sie oben im Popup oder links in der Favoritenleiste auswählen. In Abbildung 1.7 werden die Dateien und Ordner in der Listendarstellung angezeigt, die Sie vielleicht von anderen Betriebssystemen in ähnlicher Form kennen. Sie können aber mit den beiden Buttons oben links auch zur Spalten-(Browser-)Darstellung umschalten, die von erfahrenen Mac-Anwendern bevorzugt wird, da sie eine erheblich schnellere Navigation im Dateisystem ermöglicht. Unter Mac OS X 10.2 sieht der Sichern-Dialog ähnlich aus, bietet aber weniger Komfort.



Abbildung 1.7 »Sichern unter...«-Dialog

Beachten Sie, dass der Sichern-Dialog am jeweiligen Textfenster »klebt«. MacOS X nennt solche Fenster-modalen Dialoge »Sheets«, und diese können auch aus Java heraus angesprochen werden.

Das Übersetzen und Ausführen des Programms erledigen Sie nun in der Shell innerhalb der **Terminal**-Applikation (auch Kommandozeile oder Eingabeaufforderung genannt, je nachdem, mit welchem System Sie bisher gearbeitet haben). Starten Sie dazu die Anwendung `/Programme/Dienstprogramme/Terminal`. Es öffnet sich ein Terminal-Fenster mit einer Zeile




```
[straylight:~] much%
```

oder

```
Straylight:~ much$
```

je nachdem, welche Shell bei Ihnen als Standardshell konfiguriert ist (für das folgende Beispiel ist dies egal). Sie sehen den Rechnernamen³, die Tilde `~` sowie den Kurznamen des aktuell angemeldeten Benutzers. Die Tilde bezeichnet unter UNIX das Home-Verzeichnis und kann auch in Pfadangaben als Abkürzung verwendet werden.

Wenn Sie einmal Sonderzeichen eingeben möchten, können Sie sich in MacOS X 10.3 eine Tastaturübersicht einblenden lassen. Dazu müssen Sie in den Systemeinstellungen unter **Landeseinstellungen • Tastaturmenü** das Tastaturmenü in der Menüleiste aktivieren.

Dort können Sie dann auch Tastenkombinationen für bestimmte Sonderzeichen herausfinden. Die Tilde beispielsweise erhalten Sie, wenn Sie + und danach  drücken.

Geben Sie nun den Befehl `java -version` ein um zu testen, ob die JVM gestartet werden kann (und welche Version aktiv ist). Sie sollten dann ungefähr folgenden Text sehen (die Zahlen können variieren):

```
java version "1.4.2_05"  
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_  
05-141)  
Java HotSpot(TM) Client VM (build 1.4.2-38, mixed mode)
```

Mit `javac HalloWelt.java` starten Sie den Java-Compiler und übersetzen damit den Quelltext in so genannten Java Bytecode, der dann von der Virtual

³ Der Rechnername lautet in diesem Fall »Straylight«. Sie können den Namen Ihres Rechners in den Systemeinstellungen unter **Sharing • Geräte** konfigurieren.

Machine ausgeführt werden kann. Das Ergebnis sehen Sie mit dem Kommando `ls -l` (etwa »list long«, zeige den Verzeichnisinhalt in ausführlicher Form), dort taucht die Klassendatei `HalloWelt.class` auf. Wenn Sie stattdessen von `javac` Fehlermeldungen bekommen, haben Sie vermutlich das Programm falsch abgetippt oder unter einem falschen Namen gespeichert (beachten Sie die Groß-/Kleinschreibung!).

Wenn das Übersetzen geklappt hat, können Sie die class-Datei durch Aufruf der JVM mit dem Befehl `java HalloWelt` ausführen lassen. Im Terminal sollte dann die Zeichenkette »Hallo Welt!« ausgegeben werden. Falls das nicht klappt, haben Sie vermutlich entweder die `main`-Methode falsch abgetippt oder beim Aufruf von `java` einen falschen Klassennamen angegeben.

```

Terminal -- tcsh (tty1)
[much@straylight:~] much% java -version
java version "1.4.1_01"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_01-99)
Java HotSpot(TM) Client VM (build 1.4.1_01-27, mixed mode)
[much@straylight:~] much% javac HalloWelt.java
[much@straylight:~] much%
[much@straylight:~] much% ls -l HalloWelt*
-rw-r--r-- 1 much staff 423 18 Nov 16:56 HalloWelt.class
-rw-r--r-- 1 much staff 153 17 Nov 16:38 HalloWelt.java
[much@straylight:~] much%
[much@straylight:~] much% java HalloWelt
Hallo Welt!
[much@straylight:~] much% █

```

Abbildung 1.8 Die tcsh-Shell im Terminal

Wenn Sie die Shell beenden möchten, geben Sie einfach `exit` ein. Alternativ – und das ist meistens bequemer – schließen Sie einfach das Fenster oder beenden Sie die Terminal-Applikation. Sofern in der Shell nicht noch ein Programm läuft oder mit einer Fehlermeldung wartet, wird die Shell sofort korrekt beendet. Ansonsten werden Sie erst noch gefragt, ob Sie das laufende Programm wirklich terminieren wollen.

bash oder tcsh?

In der UNIX-Welt gibt es einen Glaubenskrieg, wie man ihn sonst nur zwischen Windows- und Macintosh-Anwendern bezüglich des »besseren« Systems gewohnt ist. Bei UNIX geht es aber darum, welches die beste Shell ist ... Und auch MacOS X bringt einige davon mit, wie Sie mit `cat /etc/shells` überprüfen können.

Die bekanntesten sind sicherlich `bash` und `tcsh`, die Sie in einer beliebigen Shell jederzeit mit `/bin/bash` bzw. `/bin/tcsh` aufrufen (und mit `exit` wieder verlassen können).

Bis Mac OS X 10.2 war `tcsh` die Standardshell, und wenn Sie eine Upgrade-Installation auf 10.3 durchgeführt haben, ist das auch nach wie vor so konfiguriert. Wenn Sie aber Mac OS X 10.3 komplett neu installiert haben, ist nun `bash` die Standardshell. Da die Beispiele in diesem Buch mit der `tcsh` gezeigt werden (sofern wir später überhaupt noch das Terminal verwenden), möchten Sie vielleicht auch unter 10.3 `tcsh` als Standardshell einstellen. Dazu haben Sie zwei Möglichkeiten. Entweder rufen Sie die Terminal-Einstellungen auf und geben bei »Befehl ausführen« das Kommando `/bin/tcsh` ein (siehe Abbildung 1.9). Oder Sie geben in der `bash` folgende zwei Befehle ein:

```
export EDITOR=pico
chsh
```

Damit können Sie die Standardshell in einer Text-Konfigurationsdatei ändern, was dann für Ihr gesamtes Benutzerkonto (und nicht nur für die Terminal-Applikation) gilt. Profis, die als Editor lieber den `vi` verwenden, können den ersten Befehl natürlich weglassen (und wer aus Versehen im `vi`-Editor landet, kommt mit `:q` wieder raus).

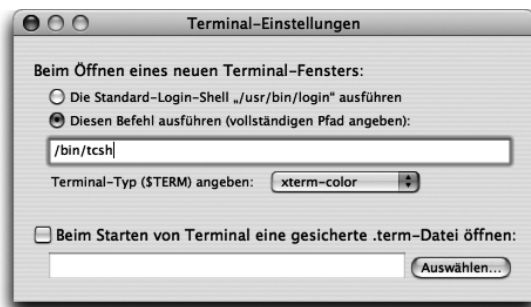


Abbildung 1.9 Terminal-Einstellungen für die `tcsh`-Shell

Als UNIX-Anwender werden Sie Ihre Shell schon in- und auswendig beherrschen, aber falls Sie bisher nur mit grafischen Oberflächen zu tun hatten, kommt Ihnen das alles vielleicht etwas kompliziert vor (seien Sie beruhigt, UNIX-Shell-Benutzern geht es mit grafischen Oberflächen meistens genauso ...). Die folgenden Kapitel betrachten vor allem grafische Entwicklungstools, aber es kann nicht schaden, sich etwas genauer mit den diversen Shell-Befehlen (`cd`, `ll` usw.) zu beschäftigen. Falls Ihnen die kurzen Beispiele in diesem Kapitel nicht

ausreichen, lesen Sie bei Bedarf bitte ein spezialisiertes Buch, z.B. »UNIX für Mac OS X-Anwender« von K. Surendorf.

Wenn der Befehl `ll` nicht gefunden wird, können Sie ihn in der `tcsh` aktivieren, indem Sie die folgenden drei Befehle ausführen:

```
echo "source /usr/share/tcsh/examples/rc"      >> ~/.tcshrc
echo "source /usr/share/tcsh/examples/login"  >> ~/.login
echo "source /usr/share/tcsh/examples/logout" >> ~/.logout
Eine ausführlichere Anleitung dazu erhalten Sie mit cat
/usr/share/tcsh/examples/README | more (weiterblättern mit )
```

Die Ausgabe des `HalloWelt`-Programms wurde direkt im Terminal angezeigt, denn dort hatten Sie das Programm ja auch gestartet. Java-Applikationen, die außerhalb einer Shell z.B. mit einem Doppelklick gestartet werden, schicken ihre Ausgaben dagegen an die **Konsole**, die Sie unter `/Programme/Dienstprogramme/Konsole` finden. Wenn Sie Mac OS X 10.3 verwenden, öffnen Sie doch einfach mal ein Finder-Fenster für ihr Home-Verzeichnis. Wenn Sie dort doppelt auf `HalloWelt.class` klicken, wird das Java-Programm gestartet und die Zeichenkette »HalloWelt!« landet in der Konsole. Einzelne Klassen auf diese Art zu starten, funktioniert aber nur, wenn keine benutzerdefinierten Bibliotheken außerhalb des Klassenpfads verwendet werden.

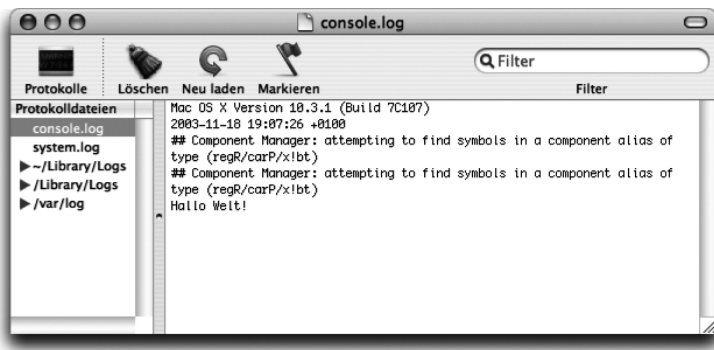


Abbildung 1.10 Java-Textausgabe in der Konsole

Die Konsole zeigt auch die so genannten Crash-Logs an, die auf der Festplatte abgelegt werden, falls eine Anwendung abstürzt. Sie finden die Crash-Logs in `~/Library/Logs/` im Verzeichnis `CrashReporter`. Dies betrifft aber eigentlich nur native Mac-Programme. Die Stack-Traces abstürzender Java-Applikationen werden im Terminal oder in der Konsole in `console.log` ausgegeben.

Wenn Sie ein bestimmtes Log-File immer im Blick haben wollen, ohne das Konsole-Fenster öffnen zu müssen, laden Sie sich doch das `GeekTool` von <http://projects.tynsoe.org/en/geektool/> herunter. Damit können Sie beliebige Textdateien im Desktop-Hintergrund anzeigen lassen.

Hilfe in der Shell

Sie haben schon kurz das Mac-Hilfesystem kennen gelernt, aber auch in der Shell können Sie ein Hilfesystem nutzen, und zwar die UNIX-»man pages« (manual pages, Handbuchseiten). Diese werden mit dem Befehl `man` aufgerufen, beispielsweise `man javac`. Mit den Pfeiltasten bewegen Sie sich zeilenweise im Dokument, mit `→` geht es seitenweise vorwärts und mit `⌘` können Sie die Hilfe jederzeit verlassen.

1.5 Entwicklerwerkzeuge und -dokumentation

Nachdem Sie gerade gesehen haben, dass Java fester Bestandteil der Mac OS X-Standardinstallation ist, sollten Sie nun noch die Apple-Entwicklertools installieren, um Apples Entwicklungsumgebung (Project Builder bis Mac OS X 10.2, Xcode ab 10.3) und alle wichtigen Java-Dokumentationen nutzen zu können. Apples »Developer Tools« tragen seit Mac OS X 10.3 den offiziellen Namen »Xcode Tools« – einen Überblick darüber erhalten Sie auf der Seite <http://developer.apple.com/tools/macosxtools.html>.

Falls Sie bei der Systeminstallation also noch nicht die Software von der Entwickler-CD installiert haben, holen Sie dies nun bitte nach (siehe Abbildung 1.11). Wenn Sie gerade einen neuen Rechner mit vorinstalliertem Mac OS X gekauft haben, bekommen Sie die Entwickler-Tools nicht auf CD mitgeliefert, sondern finden sie auf Ihrer Festplatte im Verzeichnis `/Programme/Installer/Developer Tools/`. Sie können die Tools auch von Apples ADC-Webseite, die im folgenden Abschnitt vorgestellt wird, herunterladen.

Die Entwicklerwerkzeuge werden auf der ADC-Seite in zwei Varianten angeboten: einmal als eine einzige Datei, die dann mehrere hundert MByte groß ist, und einmal in Form von kleineren Segmenten. Wenn Sie letztere Variante wählen, laden Sie alle Segmente herunter und packen Sie sie im selben Verzeichnis aus – dort sollten sich dann eine `dmg`- und viele `dmgp`-Dateien befinden. Zum Aktivieren (»mounten«) des kompletten Volumens führen Sie dann einfach einen Doppelklick nur auf die `dmg`-Datei aus.



Abbildung 1.11 Xcode-Tools installieren

Nach jeder Installation einer neuen Xcode-Version müssen Sie zusätzlich die jeweils aktuellen »Java Developer Tools« installieren, die Sie von der ADC-Webseite herunterladen können. Apple stellt für jede Java-Version ein entsprechendes Entwicklerpaket bereit – beispielsweise gehört zum »Java 1.4.2 Update 1« das Entwicklerpaket »Java 1.4.2 Update 1 Developer Tools.«

Anschließend können Sie die bekannte JDK-API-Dokumentation von Sun installieren. Apple liefert sie als Archiv mit der Entwicklersoftware mit, Sie müssen zum Auspacken nur noch ein Shell-Skript ausführen. Geben Sie dazu bis Xcode 1.2 im Terminal folgenden Befehl ein:

```
sudo /Developer/Documentation/Java/scripts/
  unjarJavaDocumentation.sh
```

Die Entwickler-Software finden Sie nun im Verzeichnis `/Developer/`, die Java-Dokumentation im Verzeichnis `/Developer/Documentation/Java/` (die API-Dokumentation von Sun ist dort im Ordner `Reference` abgelegt). Ab Xcode 1.5 führen Sie stattdessen den Befehl

```
sudo /Developer/ADC\ Reference\ Library/documentation/Java/
  scripts/unjarJavaDocumentation.sh
```

aus, wodurch die Dokumentation im mittlerweile von Apple favorisierten Verzeichnis `/Developer/ADC Reference Library/documentation/Java` angelegt wird. Wenn Sie die »Java 1.4.2 Update 1 Developer Tools« installiert haben, ist keiner der beiden obigen Befehle nötig, da die Dokumentation dann

bereits komplett ausgepackt wurde – dies ist hoffentlich auch bei den künftigen Java-Entwicklerwerkzeugen der Fall.

Obwohl Sie den Großteil von Apples Entwicklerdokumentation nun bei sich auf der Festplatte im Verzeichnis `/Developer/Documentation` bzw. `/Developer/ADC Reference Library` finden, wird hier im Buch meistens mit Webadressen auf Apples Online-Dokumentation verwiesen. Zum einen haben damit auch Entwickler auf anderen Plattformen Zugang zur Apple-Dokumentation, zum anderen greifen Sie damit immer auf die aktuellsten Informationen zu.

Um die API-Dokumentation bequem zu durchstöbern, eignet sich der `JavaBrowser` im Verzeichnis `/Developer/Applications/Java Tools/`, der in Kapitel 8 genauer vorgestellt wird.

1.6 Up to date bleiben

Vielleicht ist MacOS X 10.3 »Panther« Ihr erster Kontakt mit einem Apple-Rechner und dem aktuellen Apple-Betriebssystem, vielleicht verwenden Sie Mac OS X aber auch schon seit Version 10.0. So oder so, Sie kennen sicherlich die Tatsache, dass Software (und damit auch ein Betriebssystem) ab einer gewissen Komplexität nie ganz fehlerfrei ist. Außerdem ist Software nie ganz »fertig«, die Entwickler erweitern sie ständig – nach den Anforderungen des Marktes, nach den Wünschen der Anwender, um kompatibel mit einem neuen Standard zu sein.

Dementsprechend stellt auch Apple regelmäßig Aktualisierungen (Updates) für das Betriebssystem und die zugehörigen Programme zur Verfügung. Dies können echte Neuerungen sein, beispielsweise eine neue Java-Version, oder aber Fehlerbereinigungen, die aufgedeckte Sicherheitslücken im System schließen. Am einfachsten können Sie solche Updates von der eingebauten Software-Aktualisierung installieren lassen. Öffnen Sie dazu im Apfel-Menü oder unten am Bildschirmrand im Dock die »Systemeinstellungen« und wählen Sie dort »Software-Aktualisierung« aus. Sie können nun konfigurieren, ob das System regelmäßig automatisch nach Updates suchen soll, und Sie können die Suche durch Anklicken von »Jetzt suchen« manuell starten. Wenn die **Software-Aktualisierung** Updates findet, bekommen Sie diese in einem Dialog angezeigt, wo Sie auswählen können, ob und welche Updates installiert werden sollen.

Ab Mac OS X 10.3 können Sie im Apfel-Menü direkt den Menüpunkt »Software aktualisieren...« auswählen, dann sucht das System sofort nach verfügbaren Updates – Sie sparen sich dadurch ein paar Klicks.



Abbildung 1.12 Mac OS X Software-Aktualisierung

Wenn Sie dagegen lieber die volle Kontrolle über die Downloads der System-Aktualisierungen haben und die Archive dafür manuell herunterladen möchten oder weitere Informationen dazu benötigen, können Sie auf <http://www.info.apple.com/> in Apples Support-Datenbank nachsehen. Dies ist auch der beste Einstiegspunkt, wenn Sie nicht genau wissen, wo Sie direkt Informationen zu bestimmten Apple-Produkten finden können.

Die Aktualisierungen werden also weitestgehend über eine Online-Verbindung ins Internet abgewickelt, ebenso stehen dort aktuelle Dokumentationen zur Verfügung (sofern sie sich nicht bereits auf der Festplatte im /Developer-Verzeichnis befinden). CD-ROMs mit Software-Updates veröffentlicht Apple nur noch sehr selten, und bei gedruckten Entwicklerdokumentationen verlässt sich Apple mittlerweile komplett auf Fachbuch-Verlage⁴.

Wenn Sie ein Java-Produkt entwickeln und verkaufen (oder kostenfrei an andere Anwender abgeben), finden Ihre Kunden unter <http://www.apple.com/java/> allgemeine Hinweise zu Apples Java-Implementierung. Idealerweise bekommen die Anwender Ihrer Software aber gar nichts davon mit, dass sie ein Java-Programm starten. In Kapitel 4, *Ausführbare Programme*, werden Sie

⁴ Was einer der Gründe dafür ist, warum Sie dieses Buch aus dem Galileo-Verlag lesen ;-))

sehen, wie Sie eine Java-Anwendung so »verpacken« können, dass sie wie eine normale Mac-Applikation aussieht. Das Einzige, was reine Anwender tun müssen, ist, ab und zu die Software-Aktualisierung zu starten und vorhandene Updates installieren zu lassen.

Für Sie als Programmierer dürften Apples Entwicklerseiten deutlich interessanter sein. Die Einstiegsseite <http://developer.apple.com/> fasst die wichtigsten Technologien zusammen und gibt einen schnellen Überblick über die letzten Änderungen und Neuerungen. Für die Java-Entwicklung sollten Sie sich die Unterseite <http://developer.apple.com/java/> als Lesezeichen in Ihrem Web-Browser definieren. Hier finden Sie nicht nur die aktuelle Java-Dokumentation, sondern auch Beispielcode und FAQs (Frequently Asked Questions: häufige Fragen und nützliche Antworten).

Damit Entwickler unter sich diskutieren können, hat Apple für Java-Programmierer die **Mailingliste java-dev** eingerichtet, zu der Sie sich unter <http://lists.apple.com/mailman/listinfo/java-dev> anmelden können. Obwohl Sie hier keinen offiziellen Apple-Support bekommen, lesen und schreiben in der Liste auch Apple-Mitarbeiter. Bei dringenden Problemen kann man hier oftmals schnell Hilfe finden. Aber Achtung, das Niveau der Liste ist recht hoch – Sie sollten sich also mit den grundlegenden Java-Problemen bereits selbst beschäftigt haben. Listensprache ist Englisch. Wenn Sie ein deutschsprachiges Forum bevorzugen und Sie sich mit dem Usenet auskennen, können Sie mit einem Newsreader (z.B. Mozilla⁵ oder MacSOUP⁶) in den Gruppen *news:de.comp.lang.java* und *news:de.comp.sys.mac.misc* mitdiskutieren, je nachdem, ob Ihre Frage vor allem Java oder vor allem den Macintosh betrifft.

Die wichtigste Quelle für zukünftige, noch nicht öffentlich verfügbare Systemsoftware ist die **Apple Developer Connection (ADC)**. In einem geschützten Bereich können Sie Vorabversionen z.B. von Java-Erweiterungen herunterladen, um Ihre Software damit zu testen und rechtzeitig daran anzupassen. Um in den geschützten Bereich zu gelangen, benötigen Sie eine ADC-Mitgliedschaft. Drei Stufen werden angeboten: Online, Select und Premier. Während die Select- und die Premier-Mitgliedschaft 500 bzw. 3500 US-\$ jährlich kosten (und Ihnen dafür auch entsprechende Leistungen und Vergünstigungen bieten), ist die *Online-Mitgliedschaft kostenlos* und reicht zum Herunterladen der wichtigsten Vorabversionen vollkommen aus. Die ADC-Anmeldung und das Login in den geschützten Bereich geschieht unter <https://connect.apple.com/>.

5 Download von <http://www.mozilla.org/>

6 Download von <http://home.snafu.de/stk/macsoup/>

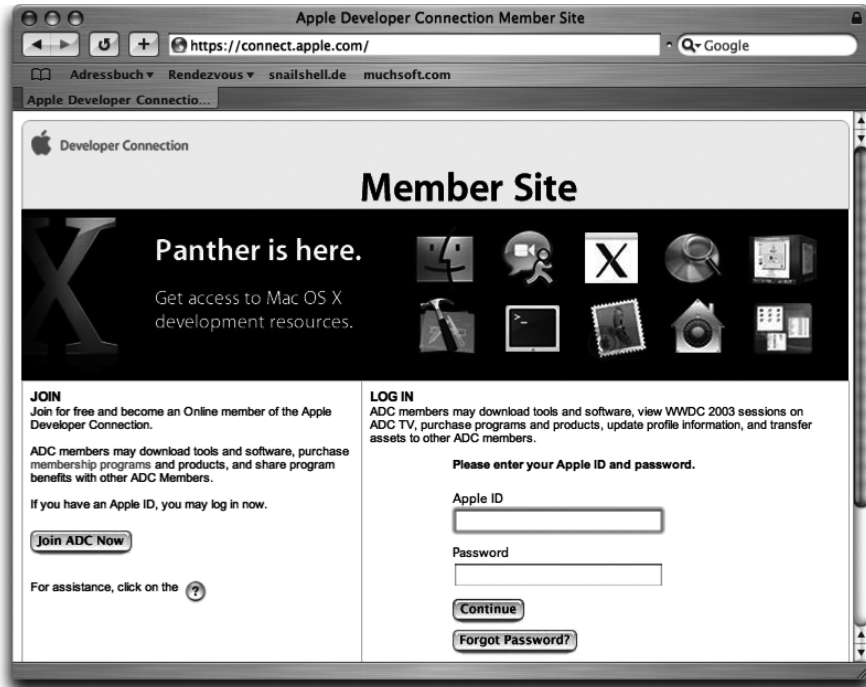


Abbildung 1.13 Apple Developer Connection (ADC): Anmeldung und Login

Sie kommen nun also problemlos an aktuelle Apple-Informationen heran, es fehlen nur noch die neuesten Informationen zu diesem Buch – denn auch das hier Geschriebene unterliegt mit der Software, die es beschreibt, ständigen Änderungen. Sollte es also Ergänzungen oder Korrekturen zu diesem Text geben, finden Sie diese auf <http://www.galileocomputing.de/> bei der Beschreibung dieses Buches als »BuchUpdate«. Wenn Sie sich bei MyGalileo anmelden, können Sie dieses Buch dort registrieren und haben dann alle Zusatzinformationen im einfachen Überblick.

Die Quelltexte der in diesem Buch vorgestellten Beispiele finden Sie auch unter <http://www.muchsoft.com/java/> in der jeweils aktuellen (und gegebenenfalls fehlerkorrigierten) Version.

1.7 Mac OS X erkennen

Wenn Sie Ihre Java-Applikation für eine bestimmte Plattform optimieren möchten, müssen Sie entweder für jede Plattform einen speziell angepassten Quelltext pflegen oder Sie pflegen nur einen einzigen Quelltext, in dem Sie zur Laufzeit feststellen, auf welcher Plattform die Software gerade läuft. Natürlich stellt sich die Frage, warum Sie eine Java-Applikation überhaupt an ein

Betriebssystem anpassen sollten – ist Java nicht gerade eine Programmiersprache, die speziell für plattformübergreifende Programmierung entwickelt wurde?

Wie Sie in den nachfolgenden Kapiteln noch sehen werden, laufen Java-Programme zwar generell ohne Änderungen auch auf MacOS X. Das von Sun propagierte »Write once, run anywhere« (einmal schreiben, überall laufen lassen) ist also, wenn man von echten Bugs der diversen Java-Implementierungen einmal absieht, weitestgehend Wirklichkeit. Der Java-Standard ist auf MacOS X verfügbar und funktioniert. Es gibt aber noch einen weiteren Standard, die »Apple Human Interface Guidelines«, also die Definition, wie ein Mac-Programm auszusehen und wie es sich zu verhalten hat. Ein Anwender, der eine gewisse Zeit mit einem Mac gearbeitet hat, erwartet einfach, dass sich eine Applikation gemäß dieses Apple-Standards verhält. Dieses Verhalten betrifft vor allem die Benutzungsoberfläche (GUI), und hier unterscheiden sich »normale« Java-Programme und Mac-Applikationen am meisten. Man sieht Ihrem Programm einfach an, dass es nicht auf einem Macintosh entwickelt oder zumindest nicht an dieses System angepasst wurde.

Zum Glück sind die Mac-Anpassungen nicht sehr aufwändig, und sie sind so kompatibel mit dem Java-Standard realisiert, dass Ihre Applikationen danach trotzdem noch auf anderen Systemen laufen! Und wenn Sie Ihren Code an MacOS X anpassen, werden Sie dabei eventuell sogar Stellen entdecken, die nicht nur auf einem Mac, sondern vielleicht auf allen Nicht-Windows-Systemen zu Problemen führen oder zumindest »unschönes« Verhalten zur Folge haben können.

Wie die speziellen Systemanpassungen konkret aussehen, werden Sie vor allem in den Kapiteln 3, *Benutzungsoberflächen*, und 5, *Portable Programmierung*, erfahren. Hier geht es zunächst darum, überhaupt zu erkennen, auf welchem Betriebssystem unser Java-Programm läuft, um dann alternativen oder optionalen Code auszuführen:

```
String osname = System.getProperty("os.name");
if (osname.toLowerCase().startsWith("mac os x")) {
    // Code speziell für Mac OS X
} else {
    // entsprechender Code für andere Betriebssysteme
}
```

Listing 11 Richtiges Erkennen von MacOS X

Es wird also zunächst der Name des Betriebssystems ermittelt und in Kleinbuchstaben umgewandelt. Wenn die Zeichenkette mit »mac os x« beginnt, kann spezieller Mac-Code ausgeführt werden.⁷

Achten Sie unbedingt darauf, nicht nur auf »Mac OS«, sondern immer auf »Mac OS X« (bzw. auf die kleingeschriebenen Zeichenketten) zu testen! Ansonsten würden Sie auch Apples alte Java-Umgebung MRJ auf Mac OS 8 und 9 erkennen, und diese Java-Implementierung – die bei Java 1.1.8 stehen geblieben ist – verhält sich teilweise deutlich anders als die Mac OS X-Java-Implementierung. Im Anhang finden Sie eine kurze MRJ-Einführung.

In alten Dokumentationen (und dementsprechend auch in alten Java-Programmen) findet man häufig noch folgendes Vorgehen, um das Mac OS zu erkennen:

```
if (System.getProperty("mrj.version") == null) {
    // Windows, Linux oder ein anderes System...
} else {
    // Mac OS-spezifischer Code
}
```

Listing 1.2 Falsches Erkennen von Mac OS X

Dieses Vorgehen funktioniert zwar derzeit noch, hat aber zwei Nachteile. Zum einen muss man die Zeichenkette in `mrj.version` noch in eine Zahl umwandeln (beispielsweise mit `Double.parseDouble()` – aber Achtung: `parseDouble()` kann z.B. »2.2.5« nicht direkt umwandeln, Sie müssen die Zeichenkette ab dem zweiten Punkt kürzen!). Dann müssen Sie aber anhand der Nummer raten, welches Mac-Betriebssystem Sie gefunden haben. 2.x wird von MRJ auf Mac OS 8 und 9 zurückgegeben, 3.x oder 69.1 oder 99 von Mac OS X ... Das ist unschön und fehleranfällig.

Zum anderen – und das ist der schwerwiegendere Grund – garantiert Apple nicht mehr, dass es `mrj.version` bei zukünftigen Java-Versionen überhaupt noch geben wird! Dann würde also auch unter Mac OS X die Abfrage in Listing »Falsches Erkennen von Mac OS X« `null` liefern und damit den Mac nicht als Mac erkennen. Also: Testen Sie die System-Property `os.name`, damit sind Sie nicht nur bei Mac OS X, sondern auch bei anderen Systemen auf der sicheren Seite.

⁷ Die Umwandlung in Kleinbuchstaben wird von Apple im Dokument <http://developer.apple.com/technotes/tn2002/tn2110.html> empfohlen – siehe auch Kapitel 19.

Es gibt nun mehrere Möglichkeiten, den Java-Quelltext für systemspezifische Anpassungen zu strukturieren. Sie können beispielsweise alle systemabhängigen Entscheidungen innerhalb einer bestimmten Klasse fällen, die Sie dann je nach Zielsystem komplett austauschen können. Oder Sie verteilen die Systemanpassungen über den gesamten Quelltext, und zwar immer an die Stellen, wo Sie systemabhängigen Code schreiben. Für beide Vorgehensweisen gibt es gute Gründe, daher soll hier keine davon favorisiert werden. Im zweiten Fall ist die Systemabfrage aber an vielen verschiedenen Stellen nötig, und das kann nicht nur lästig werden, sondern ist auch recht fehleranfällig, wenn Sie den Abfragecode an die jeweiligen Stellen kopieren.

Aus diesem Grund finden Sie auf der Buch-CD im Verzeichnis `utility/sys/` ein jar-Archiv mit einer einzigen Klasse, die die Systemabfrage für Sie schnell und übersichtlich erledigt. Diese Klasse `Sys` wird wie folgt verwendet:

```
import com.muchsoft.util.Sys;
//...
if (Sys.isMacOSX()) {
    // Mac OS X erkannt
} else if (Sys.isMacOS()) {
    // Mac OS 8 oder 9
} else if (Sys.isLinux()) {
    // auch Linux kennt Java
} else {
    // anderes System, z.B. Windows
}
```

Listing 13 Einfaches und korrektes Erkennen des Betriebssystems

In dieser Klasse gibt es neben den oben erwähnten Methoden noch `isWindows()`, `isOS2()` und `isAMac()`. Eine ausführliche JavaDoc-Dokumentation und den Quelltext finden Sie auf der Buch-CD. Zum Ausprobieren können Sie auf der beiliegenden CD das Programm `SysTest` im Verzeichnis `examples/ch01/systest/` starten.

```
//CD/examples/ch01/systest/SysTest.java
import com.muchsoft.util.Sys;
public class SysTest {
public static void main(String[] args) {
    System.out.println( "java.version: " +
                        System.getProperty("java.version") );
    System.out.println( "mrj.version: " +
                        System.getProperty("mrj.version") );
}
```

```


if ( Sys.isMacOSX() ) {
    System.out.println("Hurra, Mac OS X!");
}
else {
    System.out.println("Irgend ein anderes System...");
}
}
}

```

Listing 1.4 Einfacher Test der Klasse `com.muchsoft.util.Sys`

Entweder kopieren Sie die dortigen Dateien in Ihr Home-Verzeichnis, oder Sie wechseln mit dem Shell-Befehl »cd« im Terminal in das passende Verzeichnis auf der CD.

Pfadangaben in der Shell

Das Eingeben langer Pfadangaben kann in der Shell recht mühselig sein. Deswegen bietet das Mac OS X-Terminal eine einfache Möglichkeit, mit Pfaden (und ebenso Dateien) umzugehen: Drag & Drop. Öffnen Sie das Terminal, geben Sie »cd« (ohne Anführungszeichen) ein und ziehen Sie das Icon des Ordners »system« aus dem Finder-Fenster, das die CD-Verzeichnisse anzeigt, in das Terminal-Fenster. Der Pfad wird nun nach der letzten Texteingabe im Terminal-Fenster eingefügt! Nach dem Drücken der -Taste befinden Sie sich im system-Verzeichnis auf der CD.

Die Klasse `SysTest` verwendet das Archiv `Sys.jar`, daher müssen Sie dieses beim Kompilieren und Ausführen dem Java-System bekannt geben. Dies geschieht über den Klassenpfad (Classpath), der angibt, wo Klassen und Bibliotheken gefunden werden. Normalerweise wird nur nach den Systembibliotheken gesucht, daher geben wir beim Aufruf von `javac` und `java` über die `-classpath`-Option das Archiv mit an – nun kann Java die Methoden der Klasse `Sys` finden. Weil dieses Vorgehen auf Dauer etwas umständlich ist, werden in Kapitel 4, *Ausführbare Programme*, ein paar Möglichkeiten vorgestellt, um jar-Archive komfortabel in eigene Applikationen einzubinden.

Der Aufruf des Java-Compilers in der ersten Zeile des folgenden Beispiels ist eigentlich unnötig, da auf der Buch-CD alle Klassen bereits fertig übersetzt vorliegen. Und falls Sie das Beispiel direkt auf der CD ausführen, ist das Kompilieren dort auch gar nicht möglich, da der Compiler die resultierende class-Datei natürlich nicht auf der CD speichern kann.

```
[straylight:~] much% javac -classpath Sys.jar SysTest.java
[straylight:~] much% java -classpath .:Sys.jar SysTest
java.version: 1.4.2_05
mrj.version: 141
Hurra, Mac OS X!
```

Wenn Sie obige Ausgabe sehen, funktioniert Ihr Java-System, und ganz nebenbei wird Ihnen noch einmal versichert, dass Sie wirklich mit einem Macintosh arbeiten!

1.8 Konfigurationsdateien speichern

Sobald Sie etwas größere Java-Applikationen schreiben, werden Sie vermutlich auch bestimmte Optionen bzw. Einstellungen vorsehen, mit denen das Programm konfiguriert werden kann. Auf dem Mac werden solche Konfigurationsdaten »Einstellungen«, »Voreinstellungen« oder »Preferences« genannt. Wie und wo diese Einstellungen grafisch in der Benutzungsoberfläche repräsentiert werden, wird in Kapitel 3, *Grafische Benutzungsoberflächen*, erklärt. Hier soll es darum gehen, wo die Konfigurationsdateien im Dateisystem gespeichert werden. Der Aufbau der Dateien bleibt Ihnen überlassen, aber Sie sollten bei MacOS X bestimmte Verzeichnisse verwenden, in denen Sie die Dateien ablegen.

MacOS X bietet wie die meisten Betriebssysteme die Möglichkeit, mehrere Benutzer auf einem Rechner zu verwalten. Sie müssen daher herausfinden, welcher Benutzer Ihr Java-Programm gestartet hat, um die Konfigurationsdateien speziell für diesen Benutzer zu speichern – andere Benutzer auf demselben Rechner verwenden ja unter Umständen andere Einstellungen. Zum Glück bietet Java standardmäßig bereits eine einfache Methode, um das Home-Verzeichnis des aktuellen Benutzers herauszufinden. Dazu ruft man – wie schon bei der Erkennung des Betriebssystems – `System.getProperty()` mit einem passenden Property-Namen auf:

```
String homeDir = System.getProperty("user.home");
```

In diesem Verzeichnis können Sie die Konfigurationsdateien speichern, um den Rest (die spätere Zuordnung zum Benutzer) kümmern sich Java und das Betriebssystem.

Viele Applikationen speichern also genau hier die Einstellungen, und wenn Sie sich beispielsweise unter Linux ein Home-Verzeichnis ansehen, werden Sie feststellen, dass dort oft viel zu viele Konfigurationsdateien herumfliegen, mit denen ein »normaler« Anwender nichts anfangen kann. MacOS X versteckt die Einstellungen daher ein bisschen besser vor den Anwendern, um mehr Über-

sichtlichkeit im Home-Verzeichnis zu bekommen, und verwendet als Verzeichnis `~/Library/Preferences`. Unter Mac OS X ermitteln Sie das korrekte Verzeichnis also mit

```
String prefsDir =
    System.getProperty("user.home") + "/Library/Preferences";
```

Damit Sie nun nicht jedes Mal noch `Sys.isMacOSX()` aufrufen müssen, gibt es in der Klasse `Sys`, die Sie schon aus dem vorangegangenen Abschnitt kennen, ein paar nützliche Methoden, die Ihnen jeweils einen passenden String zurückgeben:

- ▶ `Sys.getWorkingDirectory()`
Liefert das aktuelle Arbeitsverzeichnis, d.h. das Verzeichnis, in dem Ihr Programm ausgeführt wird. Wenn Sie im Terminal arbeiten, erhalten Sie das Arbeitsverzeichnis dort mit dem Kommando »`pwd`«. In Java wird dies mit `System.getProperty("user.dir")` ermittelt.
- ▶ `Sys.getHomeDirectory()`
Liefert das Home-Verzeichnis des aktuellen Benutzers, ermittelt mit `System.getProperty("user.home")`.
- ▶ `Sys.getPrefsDirectory()`
Gibt das Verzeichnis zurück, in dem Sie die Konfigurationsdateien Ihres Programms speichern sollten. Unter Mac OS X entspricht dieses Verzeichnis `~/Library/Preferences`, alle anderen Systeme verwenden direkt die System-Property `user.home`.
- ▶ `Sys.getLocalPrefsDirectory()`
Falls Ihr Programm Einstellungen speichern muss, die alle Benutzer des lokalen Rechners gemeinsam betreffen, ermitteln Sie hiermit ein geeignetes Verzeichnis. Unter Mac OS X ist dies `/Library/Preferences/` (beachten Sie, dass die Tilde vor dem Pfad fehlt!), unter Linux `/etc`, andere Systeme verwenden das Arbeitsverzeichnis der Applikation. Aber Achtung: Es kann sein, dass Sie für das globale Konfigurationsverzeichnis keine passenden Zugriffsrechte besitzen, das Schreiben kann also fehlschlagen (auch wenn es bei Mac OS X problemlos funktioniert).

Auch für diese Methoden steht auf der CD im Verzeichnis `examples/ch01/pathstest/` ein kleines Testprogramm zur Verfügung, das Ihnen ungefähr folgende Ausgabe liefern sollte:

```
[straylight:~] much% java -classpath .:Sys.jar PathTest
Work:          /Users/much/Desktop/Galileo/CD/examples/ch01/pathstest
Home:          /Users/much
```

```
Prefs:      /Users/much/Library/Preferences
LocalPrefs: /Library/Preferences
```

Java 1.4 Preferences API

Wenn Ihre Java-Applikation Java 1.4 oder neuer voraussetzen kann, können Sie zur Verwaltung der Voreinstellungen auch die Preferences API verwenden, die Konfigurationsdaten plattformunabhängig an den richtigen Orten verwaltet. Dabei wird zwar ein systemabhängiges Format verwendet, die Daten können aber jederzeit mit `exportNode()` und `exportSubtree()` als XML-Daten exportiert (und mit `importPreferences()` wieder importiert) werden.

```
//CD/examples/ch01/prefapi/PrefTest.java
import java.util.prefs.Preferences;
public class PrefTest {
public static void main(String[] args) {
    Preferences benutzer =
        Preferences.userNodeForPackage(PrefTest.class);
    Preferences global =
        Preferences.systemNodeForPackage(PrefTest.class);
    benutzer.putInt("benutzerwert",42);
    global.putBoolean("systemweit",true);
}
}
```

Wenn Sie dieses Programm ausführen lassen, befinden sich danach zwei neue Dateien auf Ihrer Festplatte: `/Library/Preferences/com.apple.java.util.prefs.plist` für die globalen, systemweiten Einstellungen und `/Benutzer/benutzername/Library/Preferences/com.apple.java.util.prefs.plist` für die Benutzerdaten. Eine `plist`-Datei (»property list«) besitzt das typische Format für Konfigurationsdateien unter Mac OS X, dementsprechend sind die Daten darin bereits im XML-Format gespeichert.

Weitere Informationen zur Preferences API finden Sie in der Sun-Java-Dokumentation auf Ihrer Festplatte:

```
/System/Library/Frameworks/JavaVM.framework/Versions/1.4.2/
Resources/Documentation/Reference/doc/api/java/util/prefs/
package-summary.html.
```

Der Rest dieses Grundlagen-Kapitels ist für Java-Einsteiger nun nicht mehr so relevant – wenn Sie sich dazu zählen, können Sie sich gleich im nächsten Kapitel über eine geeignete Entwicklungsumgebung informieren. Java-Profis und diejenigen, die den internen Aufbau von Apples Java-System kennen lernen möchten, sollten aber hier weiterlesen.

1.9 Aufbau des Apple-Java-Systems

Während Sun für Solaris, Linux, Windows und viele andere Systeme das JRE und JDK entwickelt und zum Download anbietet, ist Apple für die Java-Implementierung in Mac OS X verantwortlich. Das ist nichts Ungewöhnliches, denn Sun sieht solche Lizenzierungen der Java-Technologie an andere Hersteller ausdrücklich vor, und Firmen wie Microsoft und IBM haben dies auch schon genutzt. Es bietet sogar enorme Vorteile, wenn der Betriebssystemhersteller Java möglichst gut in das System integriert, wie Apple dies getan hat – Programmierer können sich auf das Vorhandensein von Java verlassen, und Endanwender erleben Java als normalen Systembestandteil (und nicht als eine externe Software, die lange heruntergeladen und separat installiert werden muss).

Diese starke Integration von Java ins das Betriebssystem führt aber andererseits dazu, dass der interne Aufbau des Java-Systems nicht exakt dem Sun-Java-System entspricht. Für Endanwender ist dies nicht von Bedeutung, schließlich soll Java einfach nur funktionieren – wofür Tests und Zertifizierungen von Sun sorgen. Programmierer möchten aber ab und an wissen, wo genau die Tools installiert sind, wo die Bibliotheken zu finden sind, wo eigene Java-Erweiterungen installiert werden sollen usw. Daher werden in diesem Abschnitt die relevanten Verzeichnisse des Apple-Java-Systems vorgestellt. Doch zunächst erhalten Sie einen schnellen Überblick über die Systemarchitektur von Mac OS X.

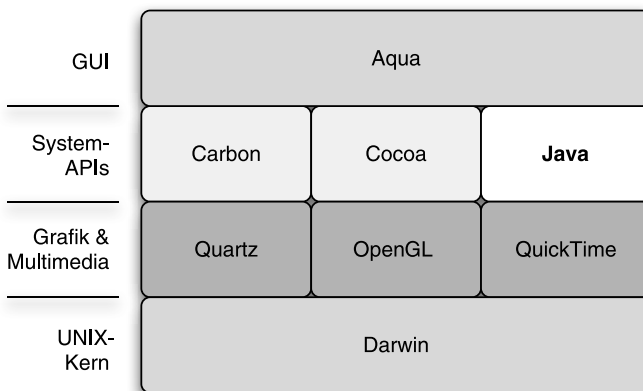


Abbildung 1.14 Mac OS X-Systemarchitektur

Als Erstes haben Sie sicherlich Aqua kennen gelernt, die Benutzungsoberfläche von Mac OS X. Auch Java-Programme verwenden normalerweise das Aqua-Aussehen (falls bei Swing-Applikationen kein anderes Look & Feel eingestellt ist). Das Verhalten von Java-Anwendungen weicht dagegen oft mehr oder weniger vom Aqua-Standard ab, es sei denn, Sie haben Ihre Anwendung mit den Informationen aus Kapitel 3, *Grafische Benutzungsoberflächen*, an den Mac angepasst. Apple beschreibt die neuesten Aqua-Eigenschaften auf <http://www.apple.com/macosx/features/aqua/>, aber das Wichtigste zur Aqua-Bedienung haben Sie schon am Anfang dieses Kapitels kennen gelernt.

Als Basis für Mac OS X dient Darwin, Apples BSD-UNIX-Implementierung. Da Apple das freie FreeBSD verwendet und für Mac OS X angepasst hat, ist Darwin selbst wieder »frei«, Apple hat also die Quelltexte des UNIX-Kerns offen gelegt⁸. Auf <http://developer.apple.com/darwin/> finden Sie die Entwicklerdokumentationen zu Darwin, und auf <http://www.opensource.apple.com/darwin-source/> können Sie schließlich den Darwin-Quelltext herunterladen – nebenbei finden Sie auf dieser Seite eine Übersicht über die verschiedenen Mac OS X-Versionen und die zugehörige Darwin-Version. Mit Mac OS X 10.3 wird Darwin 7.0 ausgeliefert.

Wenn Sie gerne etwas mehr vom UNIX-Unterbau des Systems sehen möchten, drücken Sie doch einmal + sofort nach dem Einschalten des Rechners. Sie starten dann im »Verbose«-Modus und sehen bis zum Anmelde-dialog die von Linux bekannten Text-Startmeldungen.

Dazwischen liegt eine Schicht mit Grafik- und Multimediatechnologien. Für 2D-Grafik hat Apple Quartz entwickelt, eine Art »Display PDF«⁹ (PDF für den Bildschirm), die auf der PDF-Spezifikation 1.4 aufbaut. So ganz nebenbei kann dadurch jede Mac-Applikation PDF-Dateien erzeugen, indem bei der Druckausgabe einfach eine PDF-Datei als Ziel angegeben wird! Für 3D-Grafik kommt Open GL zum Einsatz, und als Multimediatechnologie hat sich seit über zehn Jahren QuickTime als Standard etabliert (nicht nur auf der Mac-Plattform, sondern auch unter Windows und in Zukunft auch bei mobilen Endgeräten wie z.B. UMTS-Handys).

8 Dies gilt wirklich nur für den UNIX-Kern Darwin und nicht etwa für die Aqua-Oberfläche oder andere Systemkomponenten.

9 Der technische Vorgänger von Mac OS X ist nicht so sehr Mac OS 9, sondern vielmehr NeXTStep, das Betriebssystem der NeXT-Rechner. Dort wurde als 2D-Grafiktechnologie »Display Postscript« verwendet. NeXT wurde 1996 von Apple aufgekauft.

Für Sie als Software-Entwickler ist die Schicht der Programmierumgebungen wohl am interessantesten. *Programmierumgebungen* deshalb, weil sich jede Umgebung aus Systembibliotheken und einer oder mehrerer bevorzugter Programmiersprachen zusammensetzt. **Carbon** ist eine Bibliothek, die noch auf Mac OS 8/9 entwickelt wurde mit dem Ziel, den Übergang zu Mac OS X zu vereinfachen – idealerweise können Carbon-Programme sowohl auf Mac OS 8/9 als auch auf Mac OS X ohne Änderung ausgeführt werden, es werden intern einfach die richtigen Systembibliotheken verwendet. Carbon wird meistens in C, C++ oder Pascal programmiert. **Cocoa** (gesprochen »kokoh«) ist ein objekt-orientiertes Framework, das ursprünglich vom NeXT kommt und für Mac OS X weiterentwickelt wurde. Als Programmiersprache kommt Objective-C zum Einsatz, eine Smalltalk-ähnliche Weiterentwicklung der Sprache C. Neue Projekte, die als reine Mac-Anwendung entwickelt werden, verwenden meist Cocoa. Zu guter Letzt findet sich als Standard-Programmierungsumgebung auch **Java**. Neben den Java-Standardbibliotheken haben Sie daraus auch Zugriff auf Mac-spezifische Systemkomponenten, so können Sie beispielsweise auf Cocoa-Klassen zugreifen. Doch davon handeln spätere Kapitel in diesem Buch.

Die Systemkomponenten und -bibliotheken sind bei Mac OS X als so genannte »Frameworks« realisiert, was sich in den Verzeichnisnamen widerspiegelt. Doch bevor wir nun die einzelnen Mac-Verzeichnisse genauer untersuchen, finden Windows- und Linux-Programmierer zum besseren Vergleich noch einen kleinen Überblick über die Java-Struktur auf ihrem bisherigen System. Und selbst wenn Sie Java nur auf dem Mac programmieren, kann es nicht schaden, ein bisschen über den Tellerrand zu blicken – Ihre Java-Programme sollen ja auch auf anderen Systemen laufen!

1.9.1 Java-Struktur anderer Systeme

Java-Installation unter Windows

Unter Windows müssen Sie das JDK und die Dokumentation einzeln installieren, Sie können sich aber den Ort dafür beliebig aussuchen. Normalerweise wird das JDK direkt auf Laufwerk C: installiert und die Dokumentation innerhalb des JDK-Ordners, so dass Sie ungefähr folgende Verzeichnisstruktur bekommen:

```
C:\jdk1.4.2_05
  bin
  lib
  include
  docs
  demo
```

Im `bin`-Ordner liegen alle Tools (`javac`, `javadoc` usw.), im `lib`-Ordner die Systembibliotheken, die die Java-Umgebung zur Ausführung benötigt. Das `include`-Verzeichnis enthält C-Headerdateien, die Sie benötigen, wenn Sie Java über das Java Native Interface (JNI) aus C ansprechen wollen. `docs` enthält die Java-API-Dokumentation im HTML-Format, und in `demo` finden Sie schließlich einige Beispielprogramme.

Das JRE wird dagegen immer an einer festen Stelle installiert. Wenn Sie das JDK installieren, wird das JRE automatisch mit auf die Festplatte gepackt, aber natürlich können Sie das JRE auch separat installieren.

```
C:\Programme
  Java
    j2re1.4.2_05
      bin
      lib
      ext
```

In `bin` finden Sie alle nötigen Tools zur Java-Programmausführung (z.B. `java`), in `lib` wie gehabt die passenden Systembibliotheken. Interessant ist der Ordner `ext` – hier können Sie eigene Java-Bibliotheken (`jar`-Archive) speichern, die dann systemweit allen Java-Programmen zur Verfügung stehen.

Java-Installation unter Linux

Auch unter Linux kann man JDK und JRE separat installieren, wir sehen uns hier der Einfachheit halber nur das JDK an, und zwar eine Standardinstallation von Sun (es gibt noch weitere Java-Implementierungen für Linux, die dann teilweise eine andere Struktur besitzen). Die API-Dokumentation muss zusätzlich heruntergeladen und installiert werden.

```
/usr/java/j2sdk1.4.2_05
  lib
  bin
  jre
    bin
    lib
    ext
  plugin
  demo
  man
```

Wie bei Windows finden sich in `lib` und `bin` die Tools und Bibliotheken zur Programmentwicklung bzw. -ausführung. `demo` und `man` enthalten Beispielpro-

gramme und die »man pages« (knappe Dokumentationen der Tools), die Sie auch auf Mac OS X bereits kurz kennen gelernt haben. Das JRE wird auch hier vom JDK mitinstalliert und befindet sich direkt im JDK-Verzeichnis (alternativ oder zusätzlich wird für das JRE das Verzeichnis `/usr/java/j2re1.4.2_05` angelegt) – im `ext`-Verzeichnis können wieder eigene, systemweite Java-Bibliotheken abgelegt werden.

Das `plugin`-Verzeichnis enthält Plugins (kleine Software-Komponenten), mit denen Web-Browser wie Opera oder Netscape Java-Programme als Applets innerhalb eines HTML-Dokuments darstellen können.

Java-Installation unter Mac OS (Classic)

Wenn Sie bisher noch mit Java auf Mac OS 8 oder 9 gearbeitet haben, waren drei Verzeichnisse relevant (auch wenn Sie eines oder zwei davon nie direkt ansprechen mussten). Im Folgenden ist der Aufbau von Mac OS 9.2.2 mit installiertem MRJ 2.2.6 beschrieben – ältere Systeme haben teilweise andere Verzeichnisse verwendet.

```
Macintosh HD
  Systemordner
    Systemerweiterungen
      MRJ Libraries
        lib
        MRJClasses
  Applications (Mac OS 9)
    Apple Extras
      Mac OS Runtime For Java
        Apple Applet Runner
```

Als Gelegenheits-Java-Programmierer kennen Sie wahrscheinlich nur das Verzeichnis `Apple Applet Runner`. Darin befindet sich das gleichnamige Tool, um Applets außerhalb eines Web-Browsers starten zu können, sowie einige Beispiel-Applets. Innerhalb eines HTML-Dokuments im Web-Browser werden Applets mit Hilfe spezieller Systemfunktionen angezeigt, ein Java-Plugin gibt es nicht.

Der Ordner `lib` enthält nur ein paar Text-Konfigurationsdateien (u. a. mit den Sicherheitseinstellungen), irgendwelche Systembibliotheken selbst sind hier nicht gespeichert. Interessanter ist das Verzeichnis `MRJClasses`. Hierin befinden sich nämlich die vom System vorinstallierten Java-Klassenbibliotheken – und hier legen Sie auch weitere Java-Bibliotheken ab, z. B. für Swing.

1.9.2 Java-Struktur von Mac OS X

Nach diesem Überblick über »andere« Java-Systeme geht es nun aber zum eigentlichen Thema – wie sieht die Struktur der Java-Implementierung von Mac OS X aus? Apple hat das Java-System auf viele Verzeichnisse verteilt, um eine bessere Trennung zwischen Anwender-, Entwickler- und Systemkomponenten zu erreichen. In dieser Reihenfolge sehen wir uns nun die Java-Komponenten an, beginnen wir also mit dem Anwender.

Anwender

Ein normaler Anwender hat zwei bis drei Berührungspunkte mit Java: auf der Festplatte installierte Java-Applikationen, Applets im Web-Browser und seit einiger Zeit auch Java Web Start (JWS)-Anwendungen. Wie Sie als Programmierer diese verschiedenen Auslieferungsformen realisieren können, wird in Kapitel 4, *Ausführbare Programme*, beschrieben. Der Anwender möchte Applikationen aber einfach nur doppelt anklicken, dann muss das Programm gestartet werden. Es ist ihm dabei ziemlich egal, ob das Programm mit Java oder C geschrieben wurde, Hauptsache ist, dass es funktioniert.

```
Macintosh HD
  Programme
    Dienstprogramme
      Java
  Library
    Internet Plug-Ins
  Benutzer
    benutzername
      Library
        Internet Plug-Ins
```

Listing 1.5 Mac OS X Java-Struktur aus Anwendersicht

Ebenso erwartet der Anwender, dass Applets in einem Web-Browser angezeigt werden, ohne dass dafür noch irgendetwas konfiguriert oder installiert werden muss. Um dies zu gewährleisten, sind im Ordner `/Library/Internet Plug-Ins` passende Java-Plugins vorinstalliert, auf die die diversen Browser zur Applet-Darstellung zugreifen können. Neben anderen Plugins (QuickTime, Flash, Windows Media – je nachdem, welche Software auf dem Rechner installiert ist) finden Sie dort `Java Applet.plugin` für den Zugriff auf Java 1.3.1 und `JavaPluginCocoa.bundle` für den Zugriff auf Java 1.4.x.

Am Rande bemerkt: Es wäre auch möglich, bestimmte Plugins nur für einzelne Anwender zu installieren (im `Internet Plug-Ins`-Verzeichnis im jeweiligen

Benutzerordner), aber bei den Browser-Plugins ist es natürlich sinnvoll, sie allen Anwendern eines Rechners zur Verfügung zu stellen.

Konfiguriert werden die Plugins durch die (Java)-Programme `Java 1.3.1 Plugin Einstellungen` und `Java 1.4.2 Plugin Einstellungen` im Order `/Programme/Dienstprogramme/Java/`. Dort befinden sich auch der `Applet Launcher` (Apples grafische Version des `appletviewers`, der aber trotzdem noch mitgeliefert wird) und `Java Web Start`. All diese Programme werden in Kapitel 4, *Ausführbare Programme*, genauer untersucht.

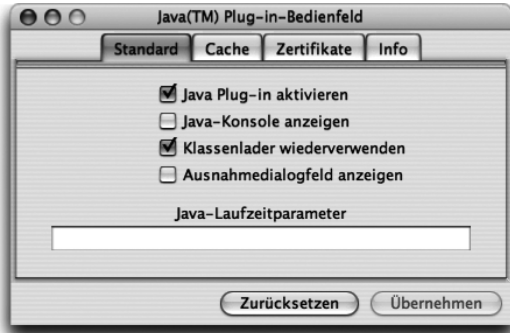


Abbildung 1.15 Java 1.3.1-Plugin-Einstellungen

Entwickler

Zur Software-Entwicklung benötigen Sie weitere Tools und geeignete Dokumentationen. Diese finden Sie, wenn Sie Apples Entwicklerwerkzeuge installiert haben, in folgender Verzeichnisstruktur:

```
Macintosh HD
  Developer
    Java
      J2EE
      Documentation
        Java
          Reference
        Examples
          Java
          J2EE
        Applications
          Java Tools
```

Listing 1.6 Mac OS X-Java-Struktur aus Entwicklersicht

In der `Documentation/Java/` sind alle wesentlichen Java-Komponenten beschrieben, und im dortigen Verzeichnis `Reference` finden Sie auch die API-Dokumentation von Sun inklusive der Apple-Erweiterungen. Passend dazu liegen in `Examples/Java/` die üblichen Beispielprogramme von Sun sowie spezielle Beispiele von Apple. Ab MacOS X 10.3 wird auch der xPetstore als Xcode-Projekt in `Examples/J2EE/` installiert.

In `/Developer/Java/J2EE` werden von Apples aktuellen Entwicklerwerkzeugen die OpenSource-Tools Ant, XDoclet und log4j mit passenden Standardlinks installiert. So erreichen Sie beispielsweise die aktuelle Ant-Installation immer mit dem Pfad `/Developer/Java/J2EE/Ant`.

Als Mac-spezifisches Tool finden Sie im Verzeichnis `Java Tools` den `Jar Bundler`, mit dem Sie bequem aus Java-Programmen ganz normal ausführbare Mac-Applikationen machen. Mit dem `JavaBrowser` können Sie einfach durch ganze Klassenhierarchien navigieren und einzelne Klassen untersuchen.

Nicht Java-spezifisch sind die Entwicklungsumgebungen `Project Builder` bzw. (ab MacOS X 10.3) `Xcode`, die sich auch gut zur Java-Entwicklung eignen. Sie finden sie in `/Developer/Applications`, und im nächsten Kapitel werden wir uns mit diesen und weiteren Entwicklungsumgebungen (IDEs) näher beschäftigen.

System

Nun fehlt nur noch die Systemintegration von Java. Wo befindet sich der Compiler, wo die JVM, wo die Klassenbibliotheken? Diese sind zusammen als eine Systemkomponente, als »Framework« realisiert. Dementsprechend sind sie wie alle System-Frameworks in `/System/Library/Frameworks` zu finden:

```
Macintosh HD
  Library
    Java
      Home
  System
    Library
      Frameworks
        JavaEmbedding.framework
        JavaVM.framework
          Classes
          Commands
          Home
          Libraries
```

```

Resources
Versions
  1.3.1
  1.4.2
    Classes
    Commands
    Home
      bin
      lib
A
  Current
  CurrentJDK

```

Listing 1.7 Systeminterne Mac OS X Java-Struktur

Im JavaVM-Framework sind fast alle Verzeichnisse nur Links auf andere Verzeichnisse, und zwar auf die aktuellste installierte Java-Version, die sich im Ordner `Versions` befindet (genauer gesagt auf die Java-Version, auf die `CurrentJDK` zeigt, aber dies ist derzeit immer die aktuellste). Dadurch brauchen externe Tools nicht die genaue Java-Versionsnummer kennen, sondern können allgemein gültige Pfade verwenden.

Wichtig: Sie selbst müssen diese Pfade normalerweise nie direkt verwenden! Falls doch (aus welchem Grund auch immer), verändern Sie die Verzeichnisstruktur niemals und betrachten Sie die Verzeichnisse als *read only*. Apples Empfehlung ist in diesem Fall, von den vorhandenen Verzeichnissen nur `Classes`, `Commands` und `Home` zu verwenden.

Als konkretes Beispiel sehen wir uns das Verzeichnis `1.4.2` an, das ältere `1.3.1` ist aber genauso aufgebaut (und falls in der Zwischenzeit das JDK 1.5.0 zur Verfügung steht, gilt dies dafür höchstwahrscheinlich ebenso). Hier sind nun keine Links mehr vorhanden, sondern »echte« Verzeichnisse. `Classes` enthält die diversen Java-Systembibliotheken, und in `Commands` finden Sie die Java-Kommandozeilen-Tools (`java`, `javac`, `javadoc`, `rmic` usw.).

Wenn Sie Java-Befehle in der Shell verwenden, werden diese normalerweise in `/usr/bin` gesucht – und dort befinden sich ganz einfach nur symbolische Links auf die Tools im aktuellen `Commands`-Verzeichnis:

```

[straylight:~] much% ll /usr/bin/javac
lrwxr-xr-x  1 root  wheel  58 26 Oct 10:57 /usr/bin/javac ->
/System/Library/Frameworks/JavaVM.framework/Commands/javac

```

Mit diesen Umwegen über Aliase möchte Apple eine größere Flexibilität erreichen, damit bei der Installation einer neuen Java-Version nicht größere Teile des Systems (oder der Anwendungssoftware) angepasst werden müssen. Nicht gelöst wird damit eine generelle Umschaltung zwischen den verschiedenen installierten Versionen – dies ist derzeit nicht von Apple vorgesehen, die Kommandozeilen-Tools verwenden also immer die neueste Java-Version. Falls Sie nun auf die Idee kommen, die ganzen symbolischen Links abzuändern: Tun Sie's nicht. Spätestens beim nächsten System-Update wird sich dies sonst mit einer defekten Java-Installation rächen ... Für speziell Mac-»verpackte« Java-Applikationen stellt sich dieses Problem aber eh nicht, denn hier kann man bequem die benötigte Java-Version angeben, und das System kümmert sich um den Rest.

Wenn Sie unbedingt ein Kommandozeilen-Tool einer bestimmten Java-Version aufrufen müssen, empfiehlt Apple, den absoluten Pfad zu verwenden. Zum Aufruf der 1.3.1-JVM können Sie im Terminal also Folgendes eingeben:

```
/System/Library/Frameworks/JavaVM.framework/Versions/1.3.1/  
Commands/java
```

Falls Sie die alte Java-Version häufiger verwenden, können Sie sich dafür einen Alias erzeugen:

```
sudo ln -s /System/Library/Frameworks/JavaVM.framework/  
Versions/1.3.1/Commands/java /usr/local/bin/java131
```

Nun können Sie die alte JVM in der Shell mit dem Kommando `java131` aufrufen (evtl. müssen Sie das Terminal-Fenster einmal schließen und neu öffnen).

Im JavaVM-Framework finden Sie auch die eigentliche Java Virtual Machine – als Link direkt im Framework sowie im Verzeichnis `A` (auf das auch `Current` zeigt). Auch hier gilt: Verwenden Sie dieses Programm und die Verzeichnisse nie direkt, das erledigen die Standard-Java-Tools für Sie.

Und vielleicht haben Sie bei der vorangegangenen Verzeichnisstruktur bemerkt, dass es auch ein `JavaEmbedding-Framework` gibt. Dies ist eigentlich nur für die Hersteller von Browsern von Interesse – es sei denn, Sie wollen in eine native Mac-Applikation direkte Aufrufe der JVM einbinden, so dass diese dann grafische Ausgaben innerhalb der Mac-Applikation vornehmen kann. Genau das machen ja Applets im Browser, und dafür hatte Apple das Embed-

ding-Framework entwickelt. Mittlerweile favorisiert Apple aber den bereits vorgestellten Java-Plugin-Mechanismus.

JAVA_HOME

In der systeminternen Verzeichnisstruktur taucht ein Home-Verzeichnis auf (entweder als Link direkt in `JavaVM.framework` oder als echtes Verzeichnis innerhalb einer bestimmten Java-Version). Nützlich hierin ist vor allem das `bin`-Verzeichnis, in dem die Java-Tools liegen. Diese Tools kennen Sie bereits aus dem `Commands`-Ordner, der aber Apple-spezifisch ist. `bin` ist dagegen systemübergreifend gültig, und Apple hat ganz einfach Links auf die Tools im `Commands`-Verzeichnis gesetzt. Einige Anwendungen benötigen das Java-Home-Verzeichnis, um zur Laufzeit die Tools aufrufen zu können, z.B. den Java-Compiler.

Dazu werten Installationsprogramme oder Shell-Skripte die Umgebungsvariable `$JAVA_HOME` aus. Wenn Sie diese Variable setzen müssen, verwenden Sie als Wert immer `/Library/Java/Home` und niemals einen spezielleren Pfad! Dieses Home-Verzeichnis ist wieder nur ein Link auf die aktuelle Java-Version, hat aber den Vorteil, dass es nach dem nächsten Java-Update wieder korrekt gesetzt ist. Im Terminal können Sie die Umgebungsvariable wie folgt setzen:

```
setenv JAVA_HOME /Library/Java/Home
```

Wenn Sie `tcsh` als Shell verwenden, können Sie die Umgebungsvariable dauerhaft setzen, indem Sie obige Zeile in die Datei `~/login` eintragen (wenn die Datei schon existiert, ergänzen Sie die Zeile irgendwo am Ende der Datei).

Wenn Sie unabhängig von einer Shell für alle Benutzer-Prozesse bestimmte Variablen setzen möchten, tragen Sie diese in die Datei `~/MacOSX/environment.plist` ein (das Verzeichnis und die Datei müssen Sie dafür normalerweise erst noch erzeugen):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>JAVA_HOME</key>
    <string>/Library/Java/Home</string>
</dict>
</plist>
```

Listing 1.8 Die Datei `environment.plist` zum Setzen von Umgebungsvariablen

Solche XML-Konfigurationsdateien können Sie am einfachsten mit dem Property List Editor aus dem Verzeichnis `/Developer/Applications/Utilities/` bearbeiten, der in Kapitel 4, *Ausführbare Programme*, noch ausführlicher vorgestellt wird.

Innerhalb eines Java-Programms kann das Java-Home-Verzeichnis wie üblich mit `System.getProperty("java.home")` abgefragt werden und sollte auch passend gesetzt sein.

In der bereits vorgestellten Klasse `com.muchsoft.util.Sys` gibt es die Klassenmethode `getJavaHome()`, die normalerweise die System-Property `java.home` ausliest und zurückgibt. Falls aber `java.home` nicht gesetzt ist, wird unter Mac OS X fest `/Library/Java/Home` geliefert.

Java-Erweiterungen

Java-Programme setzen sich neben der Standard-Klassenbibliothek oft aus vielen weiteren Bibliotheken zusammen – selbst geschriebenen und fremden, die man irgendwo gekauft oder kostenlos heruntergeladen hat. Damit diese dem Java-System bekannt sind (und damit genutzt werden können), müssen Sie sie in Verzeichnisse kopieren, die zum Klassenpfad (Classpath) gehören. Apples Java-Implementierung bietet dafür diverse `Extensions`-Ordner an, die verschiedene Zwecke erfüllen.

```
Macintosh HD
  Benutzer
    benutzername
      Library
        Java
          Extensions
Library
  Java
    Extensions
System
  Library
    Java
      Extensions
    com/apple/...
```

Listing 1.9 Verzeichnisstruktur für Java-Erweiterungsbibliotheken

In `/Library/Java/Extensions` installierte Bibliotheken gelten systemweit, können also von allen Benutzern gemeinsam genutzt werden. Im Gegensatz dazu gelten Bibliotheken in `~/Library/Java/Extensions` (beachten Sie die Tilde, es ist das Verzeichnis im entsprechenden Benutzer-Ordner gemeint) nur für den jeweiligen Benutzer. Achtung: Sie müssen `Java/Extensions` in `~/Library` normalerweise erst noch erzeugen, wenn Sie dieses Verzeichnis nutzen wollen! Der Vorteil des letztgenannten Verzeichnisses ist, dass Sie zum Schreiben keine Administrator-Rechte benötigen. Ein Installationsprogramm, das dort JAR-Archive ablegen möchte, kann also ganz normal mit Benutzer-Rechten laufen. Für das systemweite Verzeichnis müssen Sie als Administrator angemeldet sein, um Schreibzugriff zu haben, aber immerhin sind dafür keine Superuser-(Root-)Rechte nötig.

Der `Extensions`-Ordner in `/System/Library/Java` ist für Apples Java-Erweiterungen reserviert, z.B. QuickTime. Hier sollten Sie niemals irgendwelche Bibliotheken selbst installieren. Dass Sie hier keine »normalen« Java-Erweiterungen vor sich haben, erkennen Sie auch daran, dass sich im Verzeichnis `com/apple/` viele Apple-spezifische Java-Klassen befinden, die oft nur systemintern genutzt werden.

Gesucht wird nach Erweiterungsbibliotheken in der in Listing »Verzeichnisstruktur...« angegebenen Reihenfolge der Verzeichnisse (von oben nach unten). Benutzerspezifische Erweiterungen werden also zuerst gefunden, systemspezifische zum Schluss.¹⁰

Wichtig: *Die Extensions-Ordner sind nur für JAR-Archive und JNI-Bibliotheken gedacht!* Einzelne Klassen, die Sie dort unterbringen, werden nicht gefunden, selbst wenn sie gemäß ihres Pakets (Package) in der korrekten Verzeichnishierarchie abgebildet sind.

Es gibt noch ein weiteres Verzeichnis für Erweiterungen¹¹, und zwar `$JAVA_HOME/lib/ext`, also z.B. `/System/Library/Frameworks/JavaVM.framework/Versions/1.4.2/Home/lib/ext`. Dieses Verzeichnis wird am Ende nach allen anderen `Extensions`-Ordnern durchsucht. Apple empfiehlt aber ausdrücklich, dieses Verzeichnis *nicht* zu verwenden. Es ist z.B. nicht garantiert, dass hier installierte Bibliotheken nach einem System-Update noch vorhanden sind. Die besseren Alternativen haben Sie ja gerade kennen gelernt.

¹⁰ Sun beschreibt den genauen Mechanismus zum Finden von Klassen auf <http://java.sun.com/j2se/1.4.2/docs/tooldocs/findingclasses.html>. Lassen Sie sich dazu ruhig einmal die System-Properties `java.ext.dirs` und `sun.boot.class.path` ausgeben.

¹¹ Genau genommen gibt es noch ein fünftes Erweiterungsverzeichnis, `/Network/Library/Java/Extensions/`, siehe <http://developer.apple.com/documentation/ReleaseNotes/Java/Java131MOSX10.1RN/#JavaExtensions>.

Wenn Ihre Java-Bibliothek nur von einem einzigen Programm genutzt wird, ist es ratsam, diese Bibliothek in keinem der oben genannten Verzeichnisse zu installieren, sondern nur für den jeweiligen Programmaufruf einzubinden. Die dazu geeignete `classpath`-Option beim Aufruf der JVM haben Sie schon in diesem Kapitel gesehen. In Kapitel 4, *Ausführbare Programme*, sehen Sie dann, wie Sie ein Mac OS X-Programmpaket erstellen, in dem Sie solche Bibliotheken »versteckt« unterbringen können.

Ein Problem kann auftreten, wenn Ihre Applikation mit der `classpath`-Option eine Bibliothek einbindet, die sich bereits in einer anderen, eventuell unpassenden Version in einem der Erweiterungsverzeichnisse befindet (beispielsweise weil eine andere Anwendung sie dort installiert hat) – die Bibliotheken aus den Erweiterungsverzeichnissen haben normalerweise nämlich Priorität vor den `classpath`-Bibliotheken. Falls so etwas vorkommt, können Sie beim Aufruf von `java` mit der `-D`-Option die System-Eigenschaft `java.ext.dirs` setzen und damit die Suchreihenfolge für die Erweiterungsverzeichnisse verändern. Beachten Sie beim folgenden Terminal-Befehl den Punkt direkt hinter dem Gleichheitszeichen. Damit werden dann Bibliotheken aus dem aktuellen Arbeitsverzeichnis zu allererst gefunden:

```
java -Djava.ext.dirs=.:$HOME/Library/Java/Extensions:/Library/Java/Extensions:/System/Library/Java/Extensions:/Library/Java/Home/lib/ext MeineApplikation
```

Natürlich können Sie anstelle des Punktes auch irgendein anderes Verzeichnis angeben. Bitte setzen Sie die Neudefinition von `java.ext.dirs` aber nicht generell ein, sondern wirklich nur als Notlösung, wenn es mit dem Laden von Erweiterungsbibliotheken Probleme gibt.

Falls Ihr Programm weitere Ressourcen (Bilder, Texte usw.) verwendet, können Sie in `~/Library/Application Support/` (für benutzerabhängige Dateien) bzw. in `/Library/Application Support/` (für Dateien, die sich alle Benutzer teilen) ein Unterverzeichnis mit dem Namen des Programms anlegen und die Ressourcen dort installieren. Sie müssen dann aber alle Daten explizit aus diesen Verzeichnissen laden – insbesondere werden Java-Bibliotheken dort nicht gefunden, weil die Verzeichnisse nicht zum Standard-Klassenpfad gehören.

Der Standard-Klassenpfad ist derzeit in der Datei `/System/Library/Java/JavaConfig.plist` festgelegt. Sie können den Wert im Terminal mit

```
javaconfig DefaultClasspath
```


abfragen. Beides – das Programm `javaconfig` und die Datei `JavaConfig.plist` – wird von Apple aber als internes Implementationsdetail betrachtet und kann sich jederzeit ändern.

1.10 Kleine Fallen

Leider (als Programmierer würde man wohl sagen: natürlich) hat jedes Betriebssystem auch ein paar Besonderheiten, bei denen man nicht sofort weiß, ob es sich um Fehler handelt, die »historisch bedingt« immer noch vorhanden sind (und zum Standard erklärt wurden), oder ob das Verhalten absichtlich so eingebaut wurde. Wie dem auch sei, über solche Kleinigkeiten kann man leicht stolpern, wenn man neu auf einem System Software entwickelt. Die wichtigsten Mac OS X-Fallen sind hier daher kurz aufgeführt und betreffen das Dateisystem.

Das Standard-Dateisystem von Mac OS X heißt HFS+ (oder auch »Mac OS Extended«). Es stammt noch aus Mac OS 9-Zeiten und hatte damals das HFS-Format abgelöst. Übernommen davon wurde die Eigenschaft, dass sich das System bei Dateinamen zwar die Groß-/Kleinschreibung merkt (case-preserving), dass es aber beim Öffnen von Dateien egal ist, wie der Name geschrieben wird (case-insensitive). Zwei Dateien `INDEX.JSP` und `index.jsp` können also niemals gleichzeitig im selben Verzeichnis liegen, da es sich um dieselbe Datei handeln würde! Stellen Sie daher sicher, dass sich Ihre Dateien in mehr unterscheiden als nur in der Groß-/Kleinschreibung. Wenn Sie aus der Windows-Welt kommen, kennen Sie das Problem vermutlich schon, für UNIX-Anwender ist dies eher gewöhnungsbedürftig.

Achten Sie auch darauf, in Dateinamen keine Doppelpunkte zu verwenden! Bei den meisten Betriebssystemen trennen Schrägstriche (Slashes / oder Backslashes \) Unterverzeichnisse (so auch bei Mac OS X), bis Mac OS 9 wurde dafür jedoch der Doppelpunkt benutzt. Was Sie bei Mac OS X also als `Macintosh HD/System` kennen, wird bei Mac OS Classic mit `Macintosh HD:System` bezeichnet. Da Mac OS X aus Kompatibilitätsgründen immer noch zahlreiche ältere Funktionen besitzt, bei denen Pfade ins alte Format umgewandelt werden, würde ein Doppelpunkt die Pfadangabe unbenutzbar machen. Da zudem die meisten UNIX-Systeme den Doppelpunkt als Trenner von kompletten Pfadangaben verwenden (z.B. `/usr/local/bin:/usr/bin` – geben Sie einfach mal `printenv` im Terminal ein und sehen Sie sich den Wert der Umgebungsvariablen `PATH` an), ist es generell eine gute Idee, dieses Zeichen in Dateinamen zu vermeiden.

In Kapitel 5, *Portable Programmierung*, werden Sie weitere solcher Fallstricke kennen lernen, die nicht Mac- und zum Teil nicht einmal Java-spezifisch sind, die aber die systemunabhängige Programmierung erschweren können.

1.11 Literatur & Links

Am Ende jedes Kapitels finden Sie noch einmal die wichtigsten Literaturangaben und Webadressen aufgelistet – zum schnellen Überblick, wo Sie weitergehende Informationen finden können.

- ▶ C. Ullenboom, »Java ist auch eine Insel«, 4. Auflage., Galileo Press 2005
Ein umfassendes Werk zu allen Bereichen der Java-Technologie. Nicht nur zum Lernen, sondern auch als Nachschlagewerk hervorragend geeignet!
- ▶ B. Steppan, »Einstieg in Java«, 2. Auflage, Galileo Press 2005
Mit diesem Buch können Sie Java lernen und die wichtigsten Bibliotheken im Einsatz sehen. Und nebenbei werden Ihnen noch ein paar Informatik-Grundlagen sowie die Prinzipien guter Software-Entwicklung erklärt.
- ▶ K. Surendorf, »UNIX für Mac OS X-Anwender«, Galileo Press 2003
Wenn Ihnen als Mac-Anwender die Shell (das Terminal) immer noch ein bisschen fremd ist, lernen Sie hiermit die Basis von Mac OS X, den UNIX-Kern »Darwin« und die zugehörigen Tools, ausführlich kennen.
- ▶ <http://java.sun.com/>
Die Einstiegsseite von Sun rund um die Java-Technologie. Hier finden Sie die neuesten Dokumentationen und Software-Downloads für diverse Systeme. Was Sie hier nicht bekommen: die grundlegende Java-Software (JDK oder JRE) für Mac OS X – die gibt es nur direkt bei Apple.
- ▶ <http://java.net/>
Auf dieser Seite werden täglich neue Fachartikel (einige wenige davon auch speziell für den Mac) rund um die Anwendungsentwicklung mit Java veröffentlicht.
- ▶ <http://community.java.net/mac/>
Die »Mac Java Community« listet alle wichtigen Ressourcen rund um die Java-Entwicklung für Mac OS X auf – Bücher, Online-Artikel, Software-Downloads, Problemlösungen usw.
- ▶ <http://www.java.de/>
Die Java User Group (JUG) Deutschland bietet Fachartikel (auf Deutsch) und Kontakte zwischen Java-Entwicklern.
- ▶ <http://developer.apple.com/java/>
Apples Entwicklerseite für Java-Programmierer

- ▶ <http://developer.apple.com/java/faq/>
Eine wichtige, aber gerne übersehene Seite – Apple beantwortet die häufigsten Fragen rund um die Macintosh-Java-Implementation.
- ▶ <http://developer.apple.com/referencelibrary/>
In der »Reference Library« hat Apple die Entwicklerdokumentationen und Beispielcodes zu allen Bereichen von Mac OS X zusammengefasst.
- ▶ <http://www.versiontracker.com/macosx/>
Listet täglich die aktuellen Updates von Apple und Drittherstellern auf. Hier können Sie auch prima nach bestimmten Programmen oder Kategorien suchen.
- ▶ <http://www.macosxhints.com/>
Wenn Sie irgendein Problem mit Mac OS X haben, ist die Wahrscheinlichkeit hoch, dass Sie hier einen Tipp zur Lösung bekommen. Außerdem werden regelmäßig Artikel zum besseren und einfacheren Umgang mit Mac OS X veröffentlicht.

2 Entwicklungs- umgebungen

2.1	Project Builder und Xcode	76
2.2	Eclipse	98
2.3	NetBeans und Sun Java Studio Creator	103
2.4	IntelliJ IDEA	105
2.5	OmniCore CodeGuide	106
2.6	Borland JBuilder	107
2.7	Borland Together Control Center und Together Solo	108
2.8	Oracle JDeveloper	108
2.9	Metrowerks CodeWarrior	110
2.10	jEdit	111
2.11	Jext	112
2.12	JJEdit	113
2.13	BlueJ	113
2.14	DrJava	115
2.15	Literatur & Links	116

1 Grundlagen

2 Entwicklungsumgebungen

3 Grafische Benutzungsoberflächen (GUI)

4 Ausführbare Programme

5 Portable Programmierung

6 Mac OS X-spezifische Programmierung

7 Grafik und Multimedia

8 Werkzeuge

9 Datenbanken und JDBC

10 Servlets und JavaServer Pages (JSP)

11 J2EE und Enterprise JavaBeans (EJB)

12 J2ME und MIDP

A Kurzeinführung in die Programmiersprache Java

B Java auf Mac OS 8/9/Classic

C Java 1.5 »Tiger«

D System-Properties

E VM-Optionen

F Xcode- und Project Builder-Einstellungen

G Mac OS X- und Java-Versionen

H Glossar

I Die Buch-CD

2 Entwicklungsumgebungen

*»Das Wissen macht den Schüler, die freie Entwicklung den Meister.«
(Werner Kollath)*

Wenn Ihre Java-Anwendung nur aus wenigen Klassen besteht, können Sie die Quelltexte in einem einfachen Editor schreiben und das Übersetzen und Ausführen von Hand durchführen – so, wie Sie es gerade im ersten Kapitel gesehen haben. Sobald Ihr Projekt aber eine gewisse Größe überschreitet, benötigen Sie eine vernünftige Projektverwaltung, damit Sie den Überblick über die Quelltexte, die zu erzeugenden Produkte, Dokumentationen usw. behalten. Dies wird von einer Entwicklungsumgebung (»Integrated Development Environment« oder kurz IDE) geleistet, die es mittlerweile sehr zahlreich gibt – entweder speziell für Mac OS X oder häufiger systemübergreifend.

Von Apple werden geeignete Entwicklerwerkzeuge kostenlos mitgeliefert: Früher war dies der Project Builder, heute heißt die Entwicklungsumgebung Xcode. Weil dies die Standard-IDE für Mac OS X ist, wird Xcode im folgenden Abschnitt ausführlich vorgestellt. Anschließend folgt eine kurze praktische Einführung in Eclipse, eine der wohl am häufigsten benutzten systemübergreifenden Entwicklungsumgebungen für Java.

Die anderen Umgebungen werden danach nur kurz besprochen, damit Sie einen Überblick bekommen. Es gibt nicht »die« beste Entwicklungsumgebung, entsprechend kann hier auch keine absolute Wertung vergeben werden. Wenn Sie die Eigenschaften einer IDE besonders interessieren oder Sie auch einfach nur das Aussehen anspricht, testen Sie die jeweilige Umgebung am besten mit einem kleinen Projekt. Zum Schluss werden dann noch einfachere Entwicklungsumgebungen und Programmiereditoren vorgestellt. Diese sind insbesondere für Einsteiger gut geeignet, da sie nicht mit einer schier endlosen Zahl von Konfigurationsmöglichkeiten verwirren.

Bei den Preisangaben hier und in den folgenden Kapiteln wurden die Herstellerangaben übernommen. Spezielle Rabattaktionen oder generell niedrigere Preise für Schul- und Studienversionen (»academic licence«) wurden dabei nicht berücksichtigt – sehen Sie dazu bei Bedarf auf den angegebenen Webseiten nach. Und kaufen Sie nicht sofort eine der vorgestellten Entwicklungsumgebungen – gerade wenn Sie Einsteiger sind. Zum einen gibt es immer auch kostenlose Testversionen, die Sie unbedingt vor dem Kauf mit einem praxisnahen Beispiel prüfen sollten. Zum anderen gibt es zahlreiche kostenfreie IDEs, die sowohl für den Einstieg als auch für die professionelle Anwendung geeignet sind.

2.1 Project Builder und Xcode

Ab Mac OS X 10.3 ist Xcode Apples Entwicklungsumgebung für Cocoa-, Carbon- und Java-Projekte. Xcode ist eine Weiterentwicklung des Project Builders (PB), der bis Mac OS X 10.2 zum Einsatz kam. Sie finden die zu Ihrem System passende IDE im Verzeichnis `/Developer/Applications/`. In diesem Buch wird ausschließlich Xcode verwendet, was seit 2003 die Standard-IDE ist – wenn Sie wirklich noch den Project Builder einsetzen, sollten Sie die Beschreibungen und Abbildungen aber dennoch weitestgehend nutzen können.

Xcode kann problemlos Project Builder-Projekte öffnen. Sie erhalten dann allerdings einen Warnhinweis, dass Xcode das Projekt dafür konvertieren muss. Wenn Sie aufgefordert werden, eine Zeichenkodierung der Quelltexte festzulegen, wählen Sie einfach das angebotene »Mac Roman«.

In diesem Kapitel werden alle wesentlichen Eigenschaften von Xcode angesprochen, damit Sie einen Überblick erhalten. In den weiteren Kapiteln werden dann die jeweils nötigen Einstellungen detaillierter beschrieben. Die Abbildungen wurden mit Xcode 1.2 und Xcode 1.5 erstellt – in der Bedienung der beiden Versionen gibt es aber kaum Unterschiede. Die Neuerungen in Xcode 1.5 stellt Apple auf der Seite <http://developer.apple.com/tools/xcode/reviewxcode.html> vor, allgemeine Informationen erhalten Sie unter der Adresse <http://developer.apple.com/tools/xcode/>. Xcode 2.0 soll zusammen mit Mac OS X 10.4 im ersten Halbjahr 2005 erscheinen. Eine besondere Eigenschaft wird das grafische Modellieren sein – ob dies dann aber auch für Java zur Verfügung steht, bleibt abzuwarten. Xcode gehört zu den »Xcode Tools«, die Sie von Apples Entwicklerseite <https://connect.apple.com/> kostenlos herunterladen können (siehe Kapitel 1, *Grundlagen*).

Jedesmal, wenn Sie ein Xcode-Update installiert haben, sollten Sie die »Java Developer Tools« der aktuellen Java-Version erneut installieren, damit sich wieder die korrekten Java-Projekttypen auf Ihrem System befinden – die Xcode-Installation überschreibt die aktuellen Projekttypen leider mit älteren Dateien. Die »Java Developer Tools« erhalten Sie auch auf Apples Entwicklerseite.

2.1.1 Projekttypen

Beim allerersten Start stellt Ihnen Xcode ein paar Fragen, übernehmen Sie einfach alle Voreinstellungen (Sie können die Angaben später jederzeit in den Programm- bzw. Projekteinstellungen ändern). Wenn Sie nun den Menüpunkt **File**

• **New Project...** aufrufen, zeigt Ihnen Xcode eine recht lange Liste mit möglichen Projekttypen («Templates«, Schablonen) an. Sie finden viele Typen für Carbon- und Cocoa-Projekt, für Geräte-Treiber und natürlich für Java-Anwendungen (siehe Abbildung 2.1).



Abbildung 2.1 Xcode 1.5-Projekttypen

Folgende Java-Projekttypen stehen Ihnen zur Verfügung:

► Java

- ▶ »Ant-based Application Jar«
- ▶ »Ant-based Empty Project«
- ▶ »Ant-based Java Library«

Seit Xcode 1.5 wird die Projekterzeugung mit Ant unterstützt. Kapitel 8, *Werkzeuge*, behandelt Ant und die Projekttypen ausführlich – insbesondere wird auch besprochen, wie Sie Ant von Hand (und damit auch bei älteren Versionen) in Xcode einbinden können.

- ▶ »Java AWT Applet«
Erzeugt eine Projektstruktur für ein AWT-basiertes Applet. Als Produkt kommt ein JAR-Archiv heraus, das mit dem `appletviewer` gestartet wird (siehe Kapitel 4, *Ausführbare Programme*).
- ▶ »Java AWT Application«
Erzeugt ein Projekt für eine AWT-basierte Anwendung, die in ein Mac OS X-Programmpaket (Bundle) verpackt ist (siehe Kapitel 3, *Grafische Benutzungsoberflächen*).

- ▶ »Java JNI Application«
Dieser Projekttyp enthält neben einer Java-Klasse auch eine C-Datei, die über JNI eingebunden wird (siehe Kapitel 6, *Mac OS X-spezifische Programmierung*).
 - ▶ »Java Swing Applet«
Entspricht dem AWT-Applet-Typ, außer dass das Applet hierbei Swing-basiert ist.
 - ▶ »Java Swing Application«
Entspricht dem AWT-Application-Typ, wobei dieses Programm auf Swing basiert.
 - ▶ »Java Tool«
Dies ist der einfachste Java-Projekttyp, der einfach nur ein JAR-Archiv (inklusive Manifest) erzeugt.
- ▶ J2EE
- Die J2EE-Entwicklung wird in Kapitel 11, *J2EE und Enterprise JavaBeans*, ausführlicher vorgestellt. Apple weist bei den J2EE-Projekttypen explizit darauf hin, dass das Projektlayout unter Umständen nicht aufwärtskompatibel ist – wahrscheinlich vor allem, was die verwendeten Software-Komponenten (JBoss, Ant) betrifft.
- ▶ »EJB Module«
Hiermit wird ein Projekt für ein EJB-Modul (eine einfache Session Bean) mit Ant und XDoclet angelegt, wobei auch die passenden Deskriptoren für JBoss erzeugt werden.
 - ▶ »Enterprise Application«
Dieser Projekttyp legt einige grundlegende Komponenten (EJB-Modul, Servlet) für eine Enterprise-Applikation an. Es werden wieder Ant und XDoclet und für den Einsatz JBoss verwendet.
 - ▶ »Web Module«
Dieser Typ eignet sich für Webapplikationen, die auf Servlets bzw. JSP basieren (siehe auch Kapitel 10, *Servlets und JavaServer Pages*).

Wenn Sie ein einfaches Java-Programm entwickeln wollen und nicht wissen, welcher Projekttyp optimal ist, nehmen Sie einfach den »Java Tool«-Typ – die recht einfache Projektstruktur können Sie später bei Bedarf problemlos erweitern. Klicken Sie dann auf »Next« und geben Sie den Namen sowie den Pfad des neuen Projekts an (siehe Abbildung 2.2). Das Projektverzeichnis wird dabei automatisch erzeugt.

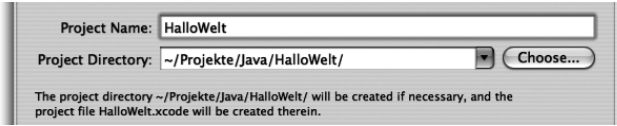


Abbildung 2.2 Projektnamen und -pfad festlegen

Im Projektverzeichnis werden – abhängig vom Projekttyp – diverse Quelltextdateien und Unterverzeichnisse angelegt. Außerdem befindet sich dort die Projekt-Konfigurationsdatei, die beim Project Builder die Dateinamenserweiterung `.pbproj`, bei Xcode entsprechend `.xcode` besitzt. Die Konfigurationsdatei ist eigentlich gar keine Datei, sondern ein Paketverzeichnis («Bundle»; siehe Kapitel 4, *Ausführbare Programme*), in dem sich die eigentlichen Konfigurationsdateien befinden. Der Finder zeigt solche speziellen Paketverzeichnisse aber als eine Datei an, damit man sie einfacher verwenden kann.

2.1.2 Projektstruktur

Nach dem Erzeugen eines Projekts öffnet sich das Projektfenster. Darin finden Sie oben die »Toolbar« (Werkzeugleiste) mit diversen Befehlen, darunter die Statuszeile für einen schnellen Überblick über die gerade laufende bzw. die zuletzt abgeschlossene Aktion (die gerade laufenden Aktionen aller geöffneten Projekte werden in **Window · Activity Viewer** angezeigt). Ganz links sehen Sie die Spalte »Groups & Files« mit der Projektstruktur, der übrige Bereich wird für die Detailansicht verwendet, d.h. für Dateilisten und Editorbereiche (siehe Abbildung 2.3).

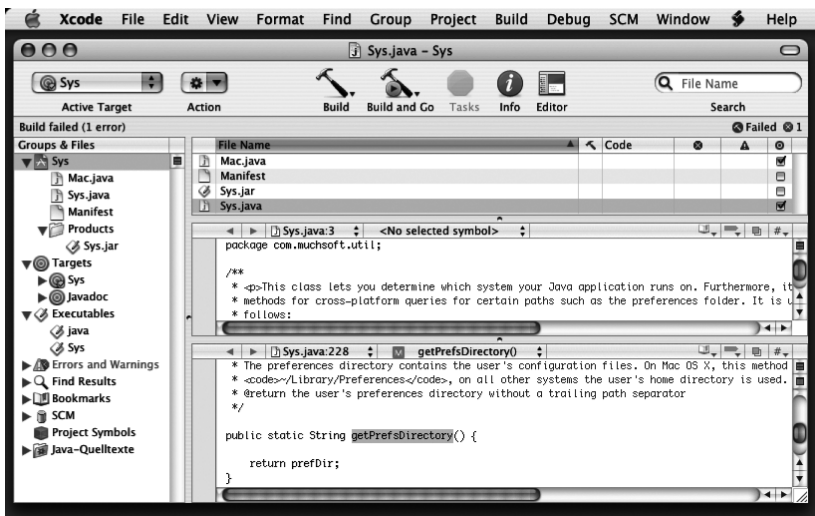


Abbildung 2.3 Das Projektfenster

In der Spalte »Groups & Files« befindet sich als Erstes die **Projekt-Gruppe**, die immer so heißt wie das Projekt. Darin werden alle Ein- und Ausgabedateien des Projekts aufgelistet. Mithilfe des Group-Menüs können Sie hier beliebige Untergruppen (beispielsweise für Java-Packages) anlegen – die Untergruppe »Products« für das oder die Produkte wird automatisch generiert. Die **Targets-Gruppe** definiert die Projekt-»Ziele« – für jedes gewünschte Produkt (JAR-Archiv, Javadoc-Dokumentation usw.) legen Sie ein separates Target an, wobei immer nur eines aktiv sein kann, auf das sich dann die Build-Aktionen beziehen. Das aktive Target wird durch ein grünes Häkchen gekennzeichnet und kann links oben im »Active Target«-Popup-Menü umgeschaltet werden. Zum Ausführen der fertigen Produkte werden **Executables** festgelegt. Es können mehrere Executables vorhanden sein, wenn das Projekt mehrere ausführbare Produkte erzeugt oder wenn ein Produkt auf diverse Arten gestartet werden kann (beispielsweise als Applet oder als Applikation). Das automatisch generierte Executable mit dem Projektnamen kann nicht direkt aus Xcode gestartet werden und ist damit eigentlich unsinnig – normalerweise werden Sie daher immer das »java«-Executable ausführen. Unter dem Executables-Eintrag werden einige »**Smart Groups**« (intelligente Gruppen) angezeigt, deren Inhalt dynamisch ermittelt wird – beispielsweise enthält die Gruppe »Errors and Warnings« die Fehler und Warnungen des letzten Compiler-Durchlaufs. Wie Sie solche Gruppen selber anlegen, sehen Sie am Ende dieses Abschnitts.

Wenn Sie eine Gruppe markieren, werden in der Detailansicht in der Spalte »File Name« alle Dateien angezeigt, die dieser Gruppe zugeordnet sind. Bei der Projekt-Gruppe können Sie zusätzlich ganz rechts an einem Häkchen erkennen, ob ein Quelltext vom aktiven Target verwendet wird. Ein Klick auf einen Dateinamen zeigt den enthaltenen Quelltext im Editorbereich unten in der Detailansicht an. Sollte der Editorbereich nicht sichtbar sein, klicken Sie einfach auf den Toolbar-Knopf »Editor«, alternativ ziehen Sie den Bereich mit dem »Griff«-Punkt auf der horizontalen Trennlinie nach oben auf.

Wenn Sie von einem anderen System zum Mac kommen, wird Ihnen der folgende Tipp nichts Neues sein: Arbeiten Sie viel mit dem Kontext-Popup-Menü, das in Xcode – wie in eigentlich allen IDEs – massiv unterstützt wird (siehe Abbildung 2.4). Bei Mac OS X wird das Kontextmenü mit Ctrl+Klick aufgerufen. Sollten Sie eine beliebige USB-Maus mit mehr als einer Taste angeschlossen haben, können Sie – ohne irgendwelche Treiber installieren zu müssen – auch wie gewohnt die rechte Maustaste verwenden. Alle Befehle des Kontextmenüs erreichen Sie aber auch über die Menüs Group, Project und SCM sowie über das »Action«-Menü in der Toolbar. Die Toolbar selbst lässt sich übrigens über ein Kontext-Popup-Menü an die eigenen Bedürfnisse anpassen.



Abbildung 2.4 Kontext-Popup-Menüs erleichtern die Arbeit.

Sollte Ihnen das Projektfenster viel zu vollgestopft mit Funktionen erscheinen, können Sie es auch einfach nur zur Anzeige der Projektstruktur und der Dateilisten benutzen. Die Quelltexte können Sie dann mit einem Doppelklick jeweils in einem eigenen Editorfenster öffnen lassen.

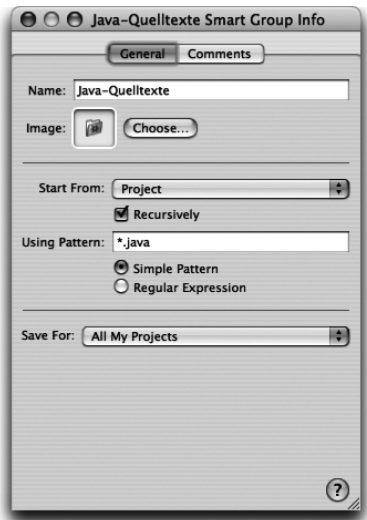
















Abbildung 2.5 »Smart Groups« festlegen und konfigurieren

Ganz unten in der Spalte »Groups & Files« taucht die benutzerdefinierte Smart Group »Java-Quelltexte« auf. Mit dem Menüpunkt **Group · New Smart Group · Simple Filter SmartGroup** können Sie selber eine solche Gruppe anlegen (siehe Abbildung 2.5). Wenn Sie als Filter-Muster »*.java« eingeben, listet die Gruppe alle Java-Quelltexte auf – gerade bei großen Projekten kann dies mit passenden Filter-Einträgen die Übersichtlichkeit deutlich erhöhen.

2.1.3 Eingeben und übersetzen

Eine neue Klasse legen Sie mit **File · New File...** an, indem Sie im erscheinenden Assistenten **Pure Java · Java class** auswählen. Vergeben Sie einen Namen und wählen Sie aus, zu welchen Targets die Datei hinzugefügt werden soll (typischerweise nur zu denen, die aus den Java-Quelltexten beispielsweise JAR-Archive erzeugen). Die neue Datei wird dann automatisch zur Spalte »Groups & Files« hinzugefügt.

Xcodes Quelltexteditor besitzt alle üblichen Fähigkeiten: Es gibt farbliche Syntaxhervorhebung, mit +Pfeiltasten können Sie Text markieren, mit +/ wortweise springen, mit **Ctrl**+/ auch innerhalb eines Wortes zu den Großbuchstaben. +/ springt zum Zeilenanfang bzw. -ende, +/ zum Dokumentanfang bzw. -ende (hierfür stehen aber auch spezielle Tasten über dem Cursorblock zur Verfügung) und **Ctrl**+/ bewegt die Eingabemarke seitenweise durch den Quelltext (auch hierfür gibt es spezielle Tasten über dem Cursorblock). Wenn Sie die Eingabemarke über eine schließende Klammer bewegen, leuchtet die dazugehörige öffnende Klammer kurz auf. Die Eigenschaften des Editors können Sie im Programm-Menü unter **Preferences... · Text Editing** (z.B. Zeilennummernanzeige, Syntax Coloring und Indentation) konfigurieren.

Die aktuelle Zeilennummer wird auch im Editorbereich links oben neben dem Dateinamen angezeigt. Rechts daneben können Sie in einem Popup-Menü ein Symbol (Klassenname, Methodename) auswählen, zu dem der Editor dann im Quelltext springt. Noch weiter rechts werden alle vorhandenen Lesezeichen (Bookmarks) angezeigt – Sie definieren diese, indem Sie an der gewünschten Stelle im Quelltext einen **Ctrl**+Klick ausführen und im Kontext-Popup-Menü den Eintrag »Add to Bookmarks« auswählen. Die Lesezeichen werden auch in der Spalte »Groups & Files« aufgelistet. Neben den Lesezeichen können Sie auf die definierten Unterbrechungspunkte (Breakpoints) zugreifen, siehe Abschnitt 2.1.4.

Der Editor-Bereich ist mit dem Symbol ganz rechts über der Bildlaufleiste beliebig häufig teilbar (»Splitting«), damit Sie verschiedene Stellen eines Quelltextes oder mehrere Quelltexte gleichzeitig bearbeiten können.

Um schnell eine Datei in einem großen Projekt zu finden, eignet sich das »Search«-Feld in der Toolbar. Für die Textsuche in einer Datei oder projektweit verwenden Sie die entsprechenden Einträge im Find-Menü.

Achten Sie darauf, dass die Quelltextdateien und der Java-Compiler dieselbe Zeichenkodierung verwenden, sonst werden Umlaute und andere Sonderzeichen falsch in den Klassendateien gespeichert. Die Quelltextkodierung stellen Sie im Info-Dialog jedes Quelltextes unter »File Encoding« ein, für den Compiler legen Sie dies in den Target-Einstellungen unter **Java Compiler Settings · Source file encoding** fest. Am besten wählen Sie in beiden Fällen »Unicode (UTF-8)«.

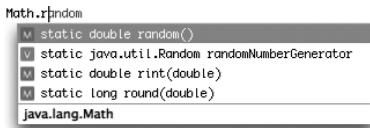


Abbildung 2.6 Eingabevervollständigung

Ab Version 1.5 beherrscht Xcode endlich auch die **Eingabevervollständigung** (»Code Completion«) für Java, die Apple auch »Code Sense« nennt (siehe Abbildung 2.6). Derzeit funktioniert dies aber nur innerhalb von Methoden so vernünftig, dass nur sinnvolle Erweiterungen angeboten werden – anderswo (d.h. auch bei Objekt- und Klassenvariablen) kommt leider eine viel rudimentärere Textvervollständigung zum Einsatz, die einfach nur bisher schon einmal getippte Textbruchstücke anbietet.

Die Eingabevervollständigung kann so eingestellt werden, dass beim Tippen automatisch ein Menü mit den passenden Vorschlägen erscheint. Alternativ können Sie dieses Menü jederzeit aufrufen, indem Sie **[F5]**, **[Ctrl]+[.]** oder **[⌘]+[ESC]** drücken (diese Tastaturkürzel sind unter **Preferences · Key Bindings · Text Key Bindings · Text Editing** konfigurierbar). Am besten schalten Sie unter **Preferences · Navigation** in der Spalte »Code sense« einfach alle Kontrollkästchen ein (siehe Abbildung 2.7).

Bei Projekten, die mit älteren Xcode-Versionen angelegt wurden, muss die Eingabevervollständigung separat aktiviert werden. Dazu markieren Sie die Projekt-Gruppe, rufen den Menüpunkt **File · Get Info** auf und schalten dort im Bereich »Code sense« alle Kontrollkästchen ein (siehe Abbildung 2.8). Wichtig ist hier vor allem auch der Eintrag »Enable Indexing«, denn ohne Index kann Xcode die möglichen Vervollständigungen nicht ermitteln. Sicherheitshalber sollten Sie dann noch »Rebuild Index« anklicken.

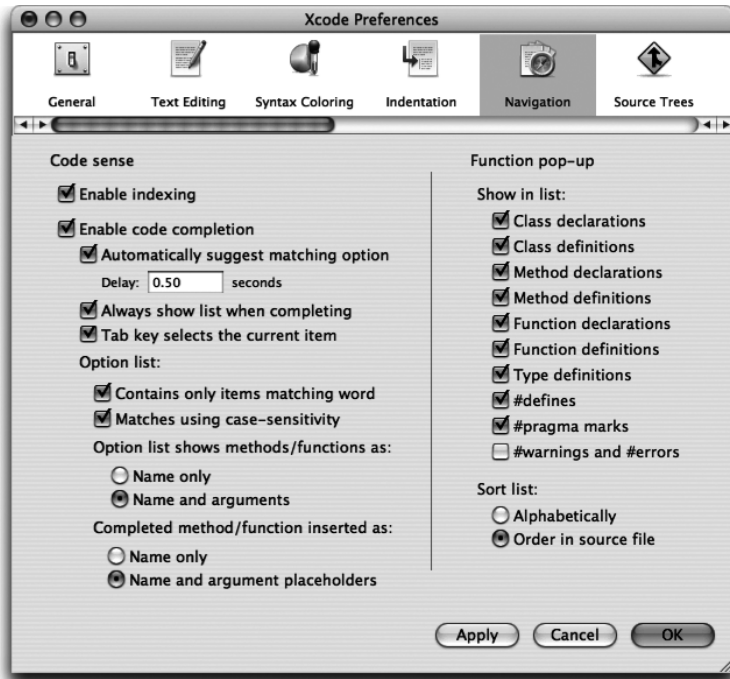


Abbildung 2.7 Einstellungen für die Eingabevervollständigung

Im Terminal können Sie das Reindizieren eines Projekts mit dem Befehl `xcodeindex` veranlassen, siehe `man xcodeindex`. Dieser Befehl wird auch für die »Xcode Index Templates« eingesetzt, mit denen Sie das zeitraubende Index-Erstellen für die Standardklassen zu Beginn jedes Projekts verhindern können. Die Index Templates legen Sie mit den folgenden Befehlen an:

```
cd /Developer/Extras/Xcode\ Index\ Templates/
./install_templates
```

In dem Verzeichnis finden Sie auch eine Anleitung, wie Sie die Index Templates manuell installieren können. Die fertigen Indizes liegen anschließend im Verzeichnis:

```
~/Library/Application Support/Apple/Developer Tools/
  Index Templates/
```

Um Hilfe zu Xcode und Java zu bekommen, verwenden Sie am einfachsten den Menüpunkt **Help • Xcode Help** oder **Help • Documentation**. Aktuelle Xcode-Änderungen finden Sie unter **Help • Show Release Notes**.

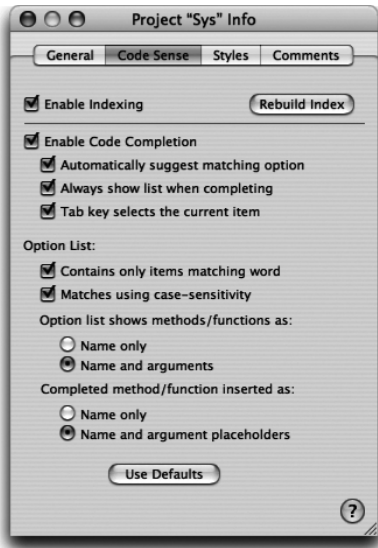




Abbildung 2.8 Index alter Projekte neu erstellen

Im Dokumentationsfenster ist über ein Suchfeld die Textsuche innerhalb der kompletten Dokumentation möglich. Wenn Sie einfach nur die Beschreibung einer Java-Klasse oder -Methode sehen möchten, ist es einfacher, wenn Sie +Doppelklick auf den gewünschten Bezeichner im Quelltext ausführen. Mit +Doppelklick auf einen Bezeichner wird die Definition des Bezeichners im Java-Quelltext angezeigt – dies klappt auch bei den Standardklassen und deren Methoden.

Damit Einträge in der Java-API-Doku gefunden werden, müssen Sie bis Xcode 1.2 einen Alias auf das Verzeichnis `/Developer/Documentation/Java/` im Verzeichnis `/Developer/Documentation/Help/` anlegen und anschließend das Programm `/Developer/Tools/pbhelpindexer` starten. Genauso können Sie auch eigene Javadoc-Dokumentationen im Xcode-Dokumentationsfenster verfügbar machen. Ab Xcode 1.5 führen Sie stattdessen den Befehl

```
sudo /Developer/Tools/pbhelpindexer -d -o /Developer/
ADC\Reference\ Library\documentation\Help\Developer\ Help\
Viewer/
```

aus. Im hinter der Option `-o` angegebenen Verzeichnis wird dabei – falls vorhanden – eine Datei mit der Dateinamenserweiterung `.pbHelpIndexerList` ausgewertet, die angibt, welche Verzeichnisse indiziert werden sollen.

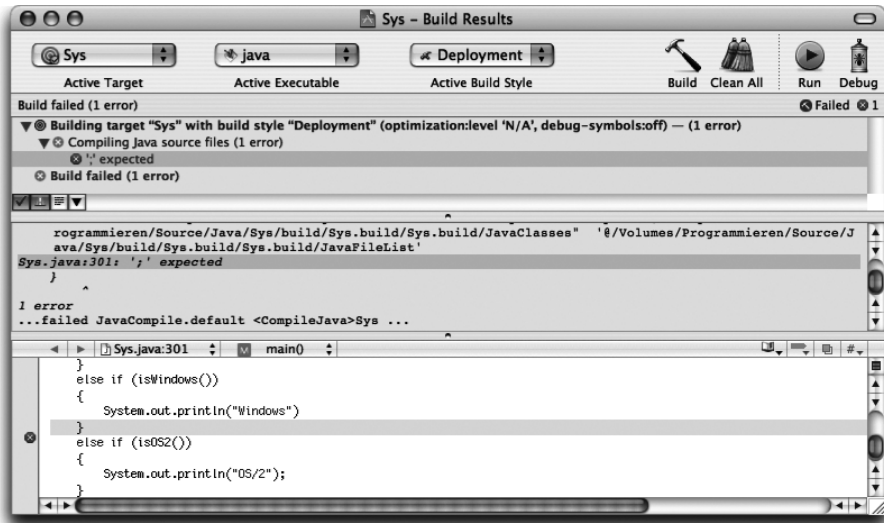


Abbildung 2.9 Ausführliche Build-Ergebnisse

Zum Übersetzen des aktuellen Targets rufen Sie »Build« oder »Build and Go« in der Toolbar auf (oder die entsprechende Einträge im Build-Menü). Ausführlichere Informationen über den Build-Prozess erhalten Sie unter **Build • Detailed Build Results** (siehe Abbildung 2.9). Wenn Sie dort im oberen Bereich einen Fehler markieren, zeigt der Editor im unteren Bereich sofort die fehlerhafte Stelle an. In der Mitte können Sie ein ausführliches Build-Protokoll (»build transcript«) ausgeben lassen – ziehen Sie dazu einfach den mittleren Bereich mit dem »Greifer« auf einer der horizontalen Trennlinien größer oder klicken Sie auf das dritte Symbol links unterhalb des Fehler-Protokolls.

Fehler und Warnungen werden auch direkt im Quelltext mit Symbolen links neben den Zeilen angezeigt (siehe Abbildung 2.10), Warnungen werden mit einem gelben Dreieck dargestellt. Wenn Sie den Mauszeiger kurze Zeit über dem Symbol stehen lassen, erscheint eine Erklärung zum Fehler.

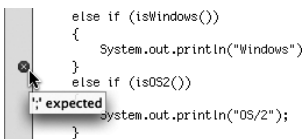


Abbildung 2.10 Syntaxfehler im Editor-Fenster

Wie fremde Klassen und Archive zum Compiler-Klassenpfad hinzugefügt werden, ist im Abschnitt 2.1.5 unter »Search Paths« beschrieben.

Den Build-Prozess können Sie auch im Terminal mit den Befehlen `pbxbuild` (beim Project Builder) bzw. `xcodebuild` starten. Eine genauere Dokumentation dazu finden Sie in den entsprechenden `man`-Seiten.

2.1.4 Ausführen und debuggen

Zum Ausführen des übersetzten Programms halten Sie in der Toolbar das Symbol »Build and Go« etwas länger angeklickt und wählen im erscheinenden Popup-Menü den Eintrag »Run Executable« aus (alternativ verwenden Sie den Menüpunkt **Debug · Run Executable** oder das Run-Symbol im »Detailed Build Results«-Fenster). Normalerweise geht dann auch ein Xcode-Konsolenfenster auf, in dem eventuelle Textmeldungen des Programms angezeigt werden – in den Executable-Einstellungen ist dies mit »Pseudo terminal« beschrieben (siehe Abbildung 2.11). Dort können Sie die Ausgabe alternativ in die System-Konsole umlenken.

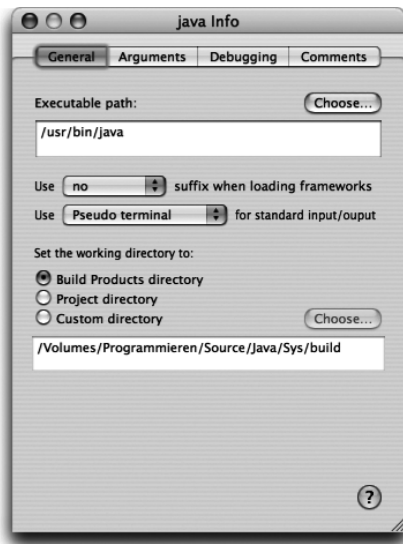


Abbildung 2.11 Executable konfigurieren

Wenn Ihre Anwendung auf fremde Klassen bzw. Bibliotheken zugreift, setzen Sie die entsprechenden Verzeichnisse und Archive mit der `-classpath`-Option auf den Klassenpfad. Die Option tragen Sie im Executable-Info-Dialog im Bereich »Arguments« ein.

Bei MacOS X-Programmpaketen (siehe Kapitel 4, *Ausführbare Programme*) erweitern Sie den Laufzeit-Klassenpfad in den Target-Einstellungen unter **Info.plist Entries · Simple View · Pure Java Specific · Classpath**.

Zur Suche nach Laufzeitfehlern (»Debugging«) starten Sie das Programm über den Menüpunkt **Build · Build and Debug** – es öffnet sich ein Debugger-Fenster, in dem das Programm ausgeführt wird. Falls ein Laufzeitfehler (eine Exception) auftritt oder das Programm einen Unterbrechungspunkt (Breakpoint) erreicht, stoppt der Debugger die Programmausführung und zeigt die aktuellen Variablenwerte an (siehe Abbildung 2.12).

Einen Unterbrechungspunkt können Sie einfach durch einen Klick links neben die Quelltextzeile setzen (und genauso wieder löschen). Die Programmausführung wird dann *vor* dieser Zeile angehalten. Den weiteren Ablauf steuern Sie mit den Aktionen in der Debugger-Toolbar: Mit »Step Over« wird die nächste Anweisung ausgeführt, wobei ein Methodenaufruf als eine Anweisung betrachtet wird, die komplett ausgeführt wird. Mit »Step Into« können Sie in Methodenaufrufe hineinspringen, mit »Step Out« entsprechend wieder heraus. »Continue« führt das Programm ganz normal weiter, »Terminate« bricht es sofort ab.

Oben links im Debugger-Fenster sehen Sie die Aufrufhierarchie (den »Stack Trace«) der Methoden, die aktuelle Methode finden Sie ganz zuoberst. Rechts daneben befindet sich eine Liste mit den aktuellen Variablen, deren Werten und (sofern sinnvoll) einer Zusammenfassung des referenzierten Objekts.

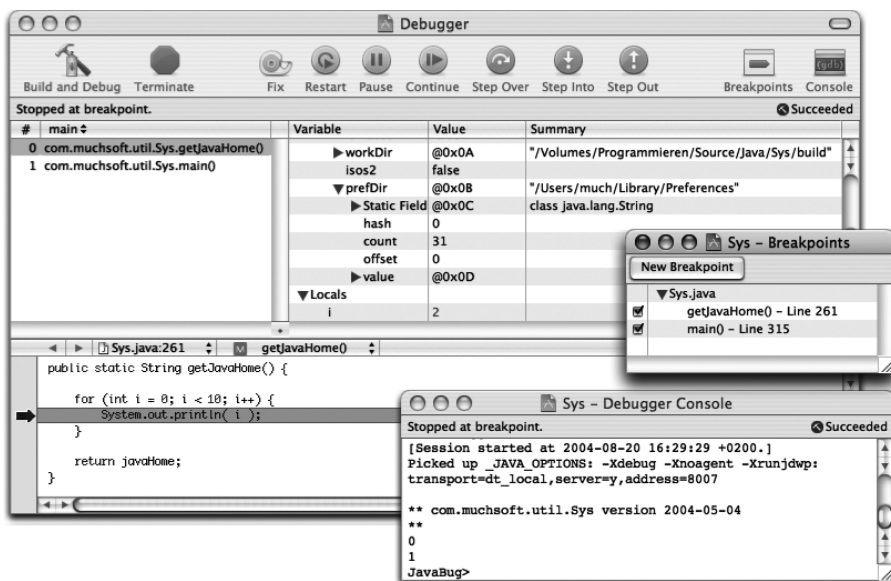


Abbildung 2.12 Programmausführung im Debugger

In der Variablenliste des Debuggers können Sie während der Programmausführung den Wert einer Variablen ändern. Rufen Sie dazu das Kontext-Popup-Menü der gewünschten Variablen auf und wählen Sie den Eintrag »Edit Value« (siehe Abbildung 2.13) – oder führen Sie einfach einen Doppelklick auf den Wert aus. Im Kontextmenü können Sie auch die Zahlendarstellung umschalten, den Datentyp einblenden oder (bei Objekten) die Zusammenfassung als Text in der Konsole ausgeben lassen.

Die Änderung und sofortige Aktivierung von Anweisungen in einer Debugging-Sitzung, die Apple »Fix and Continue« nennt, ist für Java leider noch nicht verfügbar.

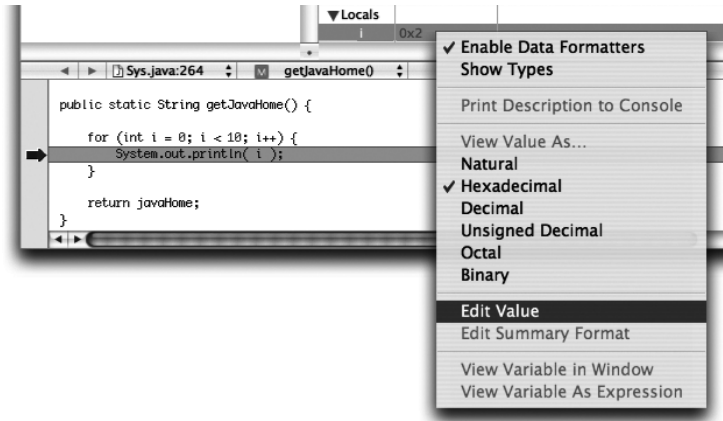


Abbildung 2.13 Variablenwerte zur Laufzeit ändern

Wenn der Debugger nicht an den definierten Unterbrechungspunkten anhält, stellen Sie im Executable-Info-Dialog im Bereich »Debugging« sicher, dass der »Java Debugger« (und nicht »GDB«) eingestellt ist.

Falls im Debugger-Fenster keine Variablen angezeigt werden, müssen Sie in den Target-Einstellungen unter **Settings • Simple View • Java Compiler Settings** das Kontrollkästchen »Generate debugging symbols« einschalten. Sicherheitshalber sollten Sie danach **Build • Clean** aufrufen und das Projekt neu übersetzen.

Weitere Möglichkeiten zur Untersuchung des fertigen Programms, z.B. Performanzmessungen mit Apples »Shark«, werden in Kapitel 8, *Werkzeuge*, vorgestellt.

2.1.5 Build-System

Normalerweise ist das **Projekt** nach dem Anlegen soweit fertig eingerichtet, dass Sie es übersetzen und ausführen können. Es enthält diverse Targets für verschiedene Produkte (JAR-Archiv, Javadoc-Dokumentation usw.). Die Targets werden über Build-Einstellungen und Build-Phasen definiert (siehe Abbildung 2.14).

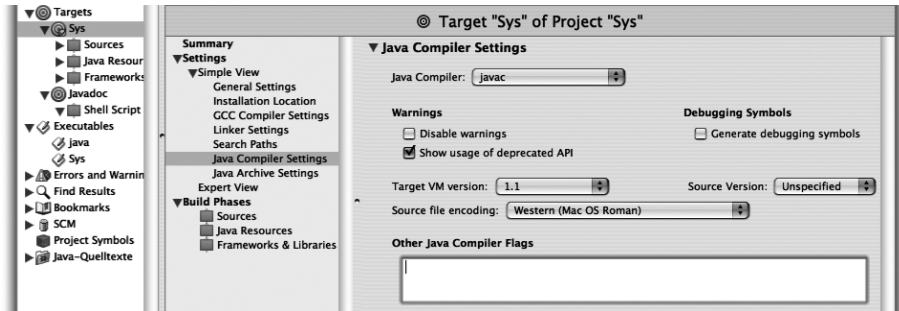


Abbildung 2.14 Target-Einstellungen

Mit dem Menüeintrag **Project · New Target...** können Sie neue **Targets** zum Projekt hinzufügen. Dabei stehen Ihnen mit Target-Typen wie »Java Tool« oder »Java Application« im Wesentlichen dieselben Möglichkeiten wie bei den Projekttypen zur Verfügung, die letztendlich nur die entsprechenden Target-Typen »verpacken.« Es gibt aber auch ein »External Target« für ein externes Build-System (das bei Ant in Kapitel 8 eingesetzt wird) oder ein »Shell Script Target« beispielsweise für Javadoc (s. u.).

Es sind auch einige »Legacy«-Target-Typen vorhanden. Damit legen Sie alte Targets an, die auch vom Project Builder verarbeitet werden können. Xcode benutzt eigentlich ein neues, Xcode-eigenes Build-System, aber für Java-Targets kann dieses leider nicht verwendet werden – hier kommt immer noch das alte Jam-basierte Build-System zum Einsatz. Insofern besteht bei Java kaum ein Unterschied zwischen den alten und neuen Target-Typen. Wenn Sie aber auf Project Builder-Kompatibilität verzichten können, nehmen Sie für neue Projekte besser die neuen Varianten.

Targets können auch von einem oder mehreren anderen Targets abhängig sein. Beispielsweise ist in Abbildung 2.15 das Javadoc-Target davon abhängig, dass vorher das eigentliche Programm erzeugt wurde. Um solche Abhängigkeiten zu definieren, fügen Sie die gewünschten Targets im Target-Info-Dialog mit dem »+«-Knopf in der richtigen Reihenfolge hinzu.



Abbildung 2.15 Abhängigkeiten von Targets

Um die **Build-Einstellungen** («Build Settings») zu verändern, markieren Sie das Target, woraufhin die Konfigurationsmöglichkeiten im Editorbereich angezeigt werden. Für Java sind folgende Einträge interessant:

► Search Paths

Klassen und JAR-Archive, die der Compiler auf dem Klassenpfad kennen muss, tragen Sie unter »Java Classes« ein – ziehen Sie die gewünschten Dateien einfach aus einem Finder-Fenster dorthin.

► Java Compiler Settings

Bestimmt, ob als Compiler `javac` oder `jikes` (siehe Kapitel 8, *Werkzeuge*) verwendet wird. Als Ziel-Laufzeitumgebung («Target VM version») sollten Sie mindestens Java 1.3 eintragen. Verwenden Sie Version 1.1 nur noch, wenn Ihr Projekt kompatibel mit ganz alten VMs sein muss, beispielsweise unter Mac OS Classic.

Falls Sie noch für Java 1.1 entwickeln müssen, stellen Sie hier als Target-VM »1.1« ein und geben Sie bei »Other Java Compiler Flags« die Option `-bootclasspath /Systemordner/Systemerweiterungen/MRJ\Libraries/MRJClasses/JDKClasses.zip` an. Dies setzt voraus, dass Mac OS Classic mit MRJ installiert ist (siehe Anhang).

► **Java Archive Settings**

Legt fest, ob das Produkt als Archiv (und mit welcher Dateinamenserweiterung) oder als einzelne Klassen in einer Verzeichnishierarchie erzeugt wird. Außerdem können Sie hier die Archiv-Kompression und die Manifest-Datei einstellen.

► **Expert View**

Zeigt alle obigen Einstellungen noch einmal als Liste mit Build-Variablen und deren Werten an. Bei Bedarf können Sie hier auch weitere Variablen hinzufügen – eine Liste der für Java wichtigen Build-Einstellungen finden Sie im Anhang.

Aufbauend auf den Build-Einstellungen definieren die **Build-Phasen** nun die eigentlichen Schritte zur Erzeugung des Produkts. Bei Java-Targets existieren normalerweise drei oder vier Phasen, deren Reihenfolge wichtig ist:

► **Sources**

Übersetzt die Quelltexte, die hier angegeben sind, in ein temporäres Verzeichnis.

► **Java Resources**

Kopiert die hier angegebenen zusätzlichen Dateien (Bilder, Texte usw.) in das temporäre Verzeichnis.

► **Bundle Resources**

Bei Mac OS X-Programmpaketen (Bundles) werden häufig systemspezifische Ressourcen (z.B. eine Icon-Datei) in das Produkt mit eingebunden. Diese Phase kopiert die angegebenen Dateien an die richtigen Stellen im Produkt. Kapitel 4, *Ausführbare Programme*, dokumentiert die Bundle-Struktur ausführlich.

► **Frameworks & Libraries**

Macht aus den Dateien im temporären Verzeichnis ein JAR-Archiv und – wenn der Target-Typ dies vorsieht – ein Mac OS X-Programmpaket. Wenn Sie fremde JAR-Archive haben, die von Ihrem Programm nicht nur über den Klassenpfad benutzt, sondern die fest in das Produkt-JAR-Archiv eingebunden werden sollen, fügen Sie die Archive mit dem Menüpunkt **Project · Add to Project...** hinzu und markieren Sie dann in dieser Build-Phase das Kontrollkästchen »Merge« (siehe Abbildung 2.16). Ein so eingebundenes Archiv taucht dann mit seiner Klassenhierarchie ausgepackt im Zielarchiv auf.



Abbildung 2.16 Fremde JAR-Archive statisch ins Produkt einbinden

Es gibt einige weitere nützliche Build-Phasen, die Sie mit dem Menüpunkt **Project • New Build Phase** hinzufügen (das Target muss dazu aktiv markiert sein) und anschließend an die passende Stelle in der Phasen-Reihenfolge ziehen können:

► Copy Files

Hiermit können Sie beliebige Dateien kopieren. Es gibt bereits einige vordefinierte Zielverzeichnisse, die bestimmte Stellen im System oder im Produkt beschreiben.

► Shell Script Files

Diese Phase lässt Sie ein in Xcode geschriebenes Shell-Skript mit einer wählbaren Shell ausführen.

Wenn Sie beispielsweise Ihrem Projekt ein »Shell Script«-Target hinzufügen (hier im Beispiel »Javadoc« genannt), besitzt dieses automatisch eine »Shell Script Files«-Build-Phase. Markieren Sie diese Phase in der Spalte »Groups & Files« und rufen Sie mit `[⌘]+[I]` den Info-Dialog auf. Als Shell ist darin `/bin/sh` voreingestellt, bei »Script« tragen Sie die gewünschten Befehle ein (siehe Abbildung 2.17) – welche Variablen Sie im Skript verwenden können, ist bei den Build-Einstellungen im Anhang dokumentiert. Bei »Input Files« und »Output Files« könnten Sie die Eingabedateien und die erzeugten Dateien des Skripts angeben. Xcode führt dann diese Build-Phase nur aus, wenn die Eingabedateien neuer als die Ausgabedateien sind.

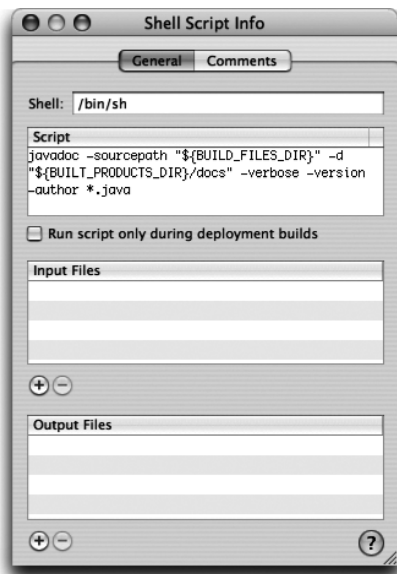


Abbildung 2.17 »Shell Script«-Build-Phase für Javadoc

Eine ausführliche Dokumentation des Xcode-Build-Systems finden Sie bei Apple auf der Seite http://developer.apple.com/documentation/DeveloperTools/Conceptual/Build_System/.

2.1.6 Versionsverwaltung

Wenn Sie mit mehreren Programmierern an einem Projekt arbeiten und damit Quelltexte von mehr als einer Person bearbeitet werden oder wenn Sie einfach nur eine Versionierung für Ihr lokales Projekt benötigen, um schnell zu früheren Versionen des Quelltexts zurückkehren zu können, sollten Sie eine Quelltext- oder Versionsverwaltung einsetzen – Apple nennt dies »Source Control Management« (SCM). Xcode unterstützt drei weit verbreitete SCM-Systeme: Concurrent Versions System (CVS, <https://www.cvshome.org/>), Subversion (<http://subversion.tigris.org/>) und Perforce (<http://www.perforce.com/>). Sie können pro Projekt entscheiden, ob und welches System Sie verwenden möchten. Voraussetzung ist nur, dass das jeweilige SCM-System auf Ihrem Rechner bereits fertig installiert ist. Die Initialisierung eines SCM-Projekts ist mit Xcode 1.5 allerdings nicht möglich. Dies müssen Sie ganz zu Anfang des Projekts mit speziellen Kommandozeilenwerkzeugen durchführen.

Im Folgenden ist die Verwendung von CVS mit Xcode beschrieben. CVS hat den Vorteil, dass es fester Bestandteil von MacOS X ist. Zum Arbeiten mit einem SCM-System müssen Sie zunächst ein Verzeichnis definieren, in dem die verwalteten Quelltexte gespeichert werden – dieses Verzeichnis wird meistens »Repository« (Aufbewahrungsort) genannt. Am einfachsten legen Sie in Ihrem Benutzerverzeichnis den Ordner `cvsroot` an. Damit CVS dieses Verzeichnis kennt, müssen Sie nun noch die Umgebungsvariable `CVSROOT` passend setzen. Da die Variable nicht nur in einer Shell, sondern auch in der Aqua-Oberfläche zur Verfügung stehen soll, müssen Sie sie in der Datei `~/macosx/environment.plist` beschreiben (siehe Kapitel 1, *Grundlagen*). Bequemer können Sie dies mit der Software »RCEnvironment« (<http://www.rubicode.com/Software/RCEnvironment/>) erledigen, die Ihnen das Setzen von Aqua-Umgebungsvariablen als Dialog in den Systemeinstellungen anbietet (siehe Abbildung 2.18).

Nun können Sie CVS im Terminal mit `cvs init` initialisieren. Danach passen Sie die CVS-Konfiguration an, um eine einfache Verwaltung von Bundles (Paketverzeichnissen) zu ermöglichen – ein Bundle wird dadurch von CVS als eine Datei und nicht als Verzeichnisstruktur angesehen. Apple liefert die dafür nötigen Definitionen in der Datei `cvswrappers` mit den Entwicklerwerkzeugen mit. Die Anpassung geschieht in einem temporären Verzeichnis, das anschließend wieder gelöscht wird:

```
mkdir cvstemp
cd cvstemp
cvs checkout CVSROOT
cp /Developer/Tools/cvswrappers ./CVSROOT
cvs commit -m "Apple cvswrappers"
cd ..
rm -R cvstemp
```

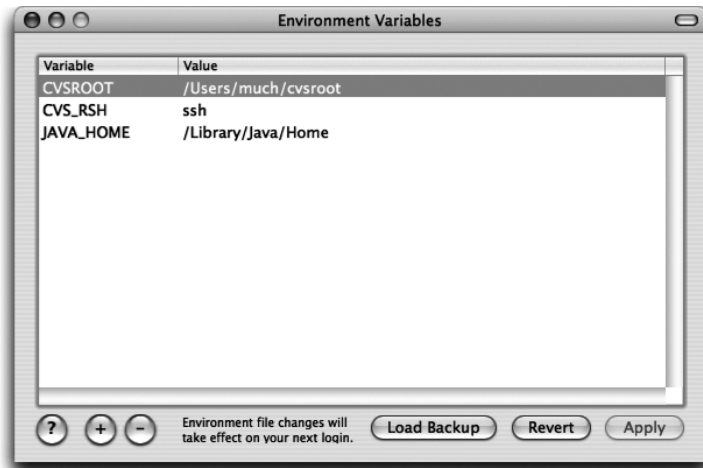


Abbildung 2.18 Umgebungsvariablen mit REnvironment setzen

Das CVS-Wurzel-Projekt, das zur Konfiguration des Systems dient, wird zur Bearbeitung aus dem Repository ausgelesen («ausgecheckt»), die gewünschten Dateien werden mit dem `cp`-Befehl aktualisiert und die Änderungen mit `cvs commit` in das Repository zurückgeschrieben. Die `-m`-Option gibt dabei eine kurze Meldung an, mit der Sie den Vorgang später leichter wiederfinden können.

Dann fügen Sie ein bestehendes Projekt zum Repository hinzu. Führen Sie dazu am besten vorher ein »Clean All« durch oder löschen Sie den Ordner `build` im Projektverzeichnis – es sollen nur die Quelltexte versioniert werden, nicht aber die fertigen Produkte (die Sie ja jederzeit durch Neukompilieren wieder erzeugen können). Im Projektverzeichnis rufen Sie nun den `cvs import`-Befehl auf (die beiden letzten Parameter, der Herstellername und der Ausgabezusatz, werden hier eigentlich nicht benötigt, müssen aber angegeben werden):

```
cd ~/Projektname
cvs import -m "Import bestehendes Projekt" Projektname tm start
cd ..
```

```
mv Projektname Projektname.backup
cvs checkout Projektname
```

Zum Schluss wird das bisherige Projektverzeichnis als Sicherungskopie gespeichert und das eigentliche Projektverzeichnis mit `cvs checkout` aus dem Repository abgefragt. Wenn Sie letzteres in Xcode laden, sehen Sie zunächst keinen Unterschied zu vorher – Sie müssen CVS erst noch aktivieren. Markieren Sie in der Spalte »Groups & Files« die Projektgruppe und rufen Sie den Info-Dialog auf (siehe Abbildung 2.19). Schalten Sie dort das Kontrollkästchen »Enable SCM« ein und wählen Sie im Popup-Menü rechts daneben »CVS« aus. Mit »Edit...« können Sie überprüfen, ob das richtige Kommando `/usr/bin/cvs` eingestellt ist.

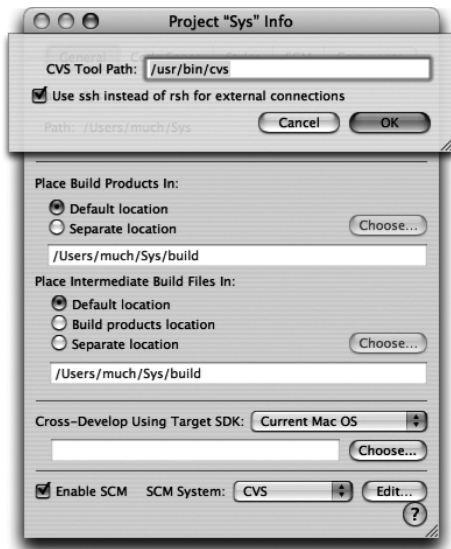


Abbildung 2.19 CVS für ein Projekt aktivieren

Damit ist CVS für Ihr Projekt aktiv! Sie können die Dateien des Projekts nun ganz normal verwenden – wobei Sie aber immer mit einer lokalen Kopie arbeiten und nicht direkt mit den Daten aus dem Repository. Sobald Sie einen Quelltext verändert haben, sehen Sie in der Spalte »Groups & Files« ein »M« (»modified«, verändert) neben dem Dateinamen, außerdem taucht diese Datei dann unter dem Eintrag »SCM« weiter unten in der Spalte auf.

Mit dem Menüpunkt **SCM • SCM** rufen Sie eine Liste aller lokal veränderten Dateien auf, außerdem können Sie sich dort das Protokoll der letzten SCM-Vorgänge ansehen (siehe Abbildung 2.20). Im Kontext-Popup-Menü jeder Datei übertragen Sie die Änderungen mit »Commit Changes...« ins Repository (man

spricht auch vom »Festschreiben«), wobei Sie noch einen kurzen Kommentar hinzufügen können. Wenn Sie dagegen merken, dass die letzten Änderungen nicht sinnvoll waren, können Sie mit »Discard Changes« zum letzten festgeschriebenen Stand zurückkehren, die Änderungen also verwerfen. Das Festschreiben und Verwerfen ist auch über entsprechende Einträge im SCM-Menü und in den Kontext-Popup-Menüs in der Spalte »Groups & Files« möglich.

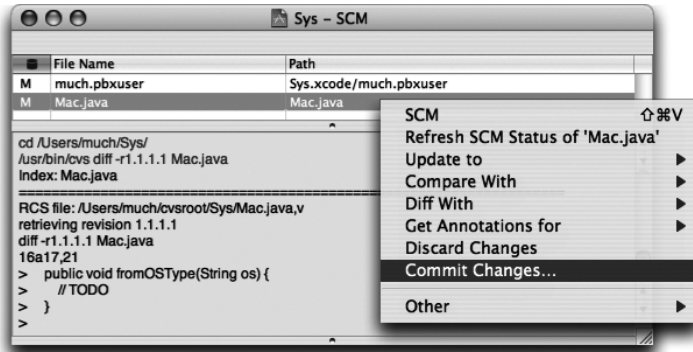


Abbildung 2.20 Dateien mit SCM verwalten

In den Menüs finden Sie auch die Einträge »Diff With« und »Compare With«. Damit können Sie den aktuellen Stand mit einer gespeicherten Version vergleichen. »Diff« gibt die gefundenen Unterschiede als Text in einem Editorfenster aus, »Compare« ruft das Programm FileMerge aus dem Verzeichnis /Developer/Applications/Utilities/ auf, das Ihnen die Unterschiede auch grafisch darstellt (siehe Abbildung 2.21).

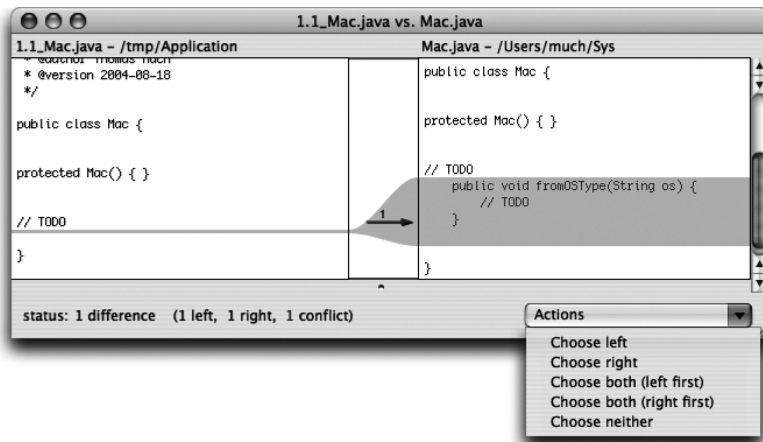


Abbildung 2.21 Quelltexte mit FileMerge vergleichen

Wenn Sie CVS nicht nur lokal auf Ihrem Rechner, sondern für Teamarbeit im Netzwerk nutzen möchten, finden Sie auf der Seite <http://maczealots.com/tutorials/xcode-cvs/> eine gelungene Anleitung. Wie Sie Xcode-Projekte für CVS, Subversion und Perforce einrichten, hat Apple im Dokument http://developer.apple.com/documentation/DeveloperTools/Conceptual/Xcode_SCM/ beschrieben. Für Subversion finden Sie auf <http://homepage.mac.com/martinott/> spezielle Mac-Installer und auf <http://www.lachoseinteractive.net/en/community/subversion/> einen grafischen Subversion-Client.

2.2 Eclipse

Das Eclipse-Projekt von IBM gehört sicherlich zu den bekanntesten Entwicklungsumgebungen, steht es doch für viele Systeme (u.a. Windows, Linux, Solaris und MacOS X) zur Verfügung. Dabei ist Eclipse selbst eigentlich nur eine kleine Rahmenanwendung und wird erst durch die zahlreichen Plugins wirklich mächtig. Vor allem ist Eclipse dadurch nicht auf die Java-Programmierung beschränkt – mit den passenden Plugins werden auch andere Sprachen unterstützt. Wenn Sie das »Eclipse SDK« herunterladen, sind aber alle grundlegenden Elemente für die Java-Entwicklung bereits enthalten. Einen knappen Überblick über Eclipse finden Sie auch bei Apple auf der Seite <http://developer.apple.com/tools/eclipse.html>.

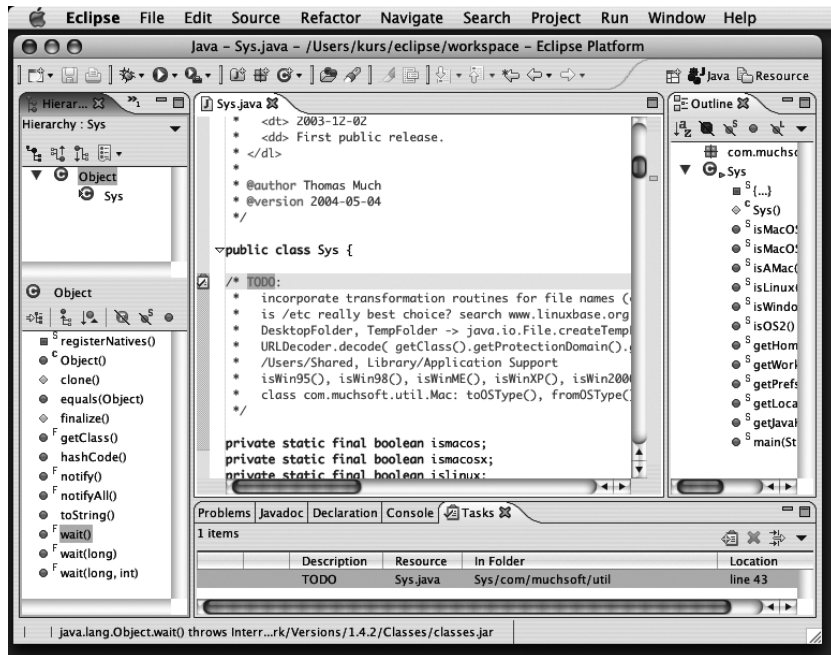


Abbildung 2.22 Eclipse 3.0

Wie die meisten in Java geschriebenen Entwicklungsumgebungen benötigt Eclipse ausreichend RAM, um annehmbar zu laufen. 256 MByte RAM sollten es mindestens sein, besser deutlich mehr. Es gibt derzeit zwei stabile Hauptversionen, die auch beide mit Mac OS X genutzt werden können: Eclipse 2.1 und Eclipse 3.0. Wenn Sie keine bestimmten Projektvorgaben haben, nehmen Sie die neuere Version – das aktuelle Eclipse ist deutlich besser ins System integriert als die älteren Versionen.

- ▶ Eclipse 3.0 – kostenlos
<http://www.eclipse.org/>

Eclipse verwenden

Eclipse stellt beim Programmieren viele Hilfen zur Verfügung, die dem Profi das Leben erleichtern, die sich aber für einen Einsteiger zunächst als Hürde erweisen können. Daher sehen Sie nun im Folgenden eine kurze Anleitung, wie Sie mit Eclipse ein Java-Projekt anlegen, Quelltexte eingeben und das Programm schließlich übersetzen und ausführen. Zuerst rufen Sie dazu mit dem Menüpunkt **File • New • Project...** den Projekt-Assistenten auf, in dem Sie ein »Java Project« anlegen (siehe Abbildung 2.23). Im »Next«-Dialog können Sie als Projektnamen beispielsweise »Test« eintragen.

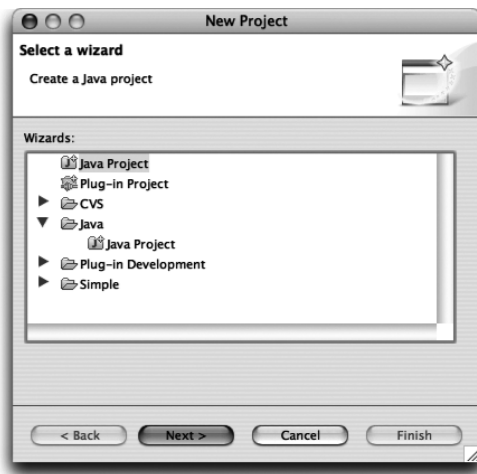


Abbildung 2.23 Neues Projekt in Eclipse anlegen

Nachdem Eclipse das Projekt erzeugt hat, rufen Sie mit **File • New • Class** den Klassen-Assistenten auf (siehe Abbildung 2.24). Tragen Sie den gewünschten Klassennamen (hier »HalloWelt«) und gegebenenfalls das Package ein. Durch die Kontrollkästchen unten im Dialog können Sie u.a. automatisch eine

main()-Methode generieren lassen. Haben Sie alle Daten eingegeben, klicken Sie auf »Finish«.

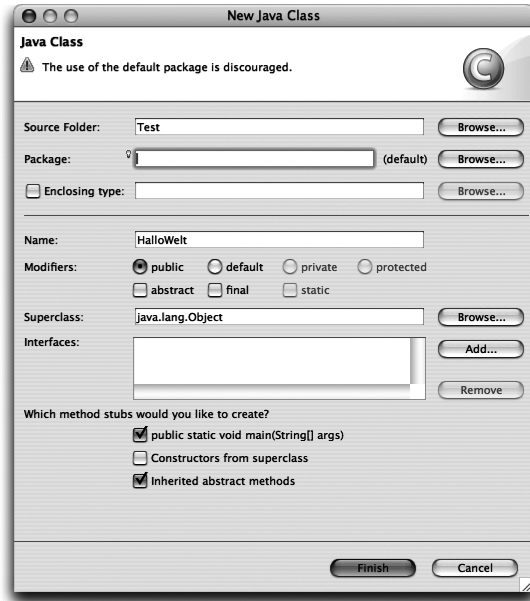


Abbildung 2.24 Java-Klasse anlegen

Während Sie den Quelltext eingeben, übersetzt Eclipse den Quelltext im Hintergrund (dies nennt sich inkrementelles Kompilieren oder »autobuild«) – normalerweise müssen Sie Ihr Projekt also gar nicht mehr explizit übersetzen lassen! Sollten Sie bei der Eingabe einen Fehler machen, unterkringelt Eclipse die fehlerhafte Stelle rot (siehe Abbildung 2.25). Wenn Sie den Mauszeiger über die entsprechende Stelle bringen (oder über das Lampensymbol links am Rand), wird Ihnen der Fehlergrund angezeigt. Teilweise bekommen Sie hier auch Vorschläge, wie Eclipse den Fehler für Sie korrigieren kann.

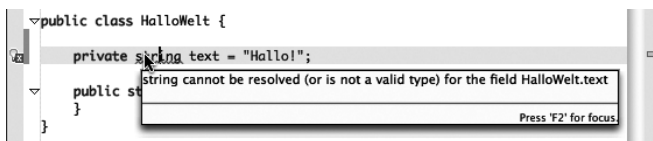


Abbildung 2.25 Fehler bei der Eingabe

Schon lange beherrscht Eclipse die Eingabevervollständigung. Zum einen öffnet sich ein Popup-Menü mit den Textvorschlägen, wenn Sie einen Ausdruck mit einem Punkt beenden und kurz warten – in Abbildung 2.26 wurde beispielsweise `System.out.` eingegeben (zum Eintrag im Popup-Menü wird hier

zusätzlich ein kurzer Hilfetext angezeigt). Zum anderen können Sie jederzeit **Ctrl** + **[]** drücken, dann macht Eclipse passende Erweiterungsvorschläge für den Text bei der Eingabemarke.

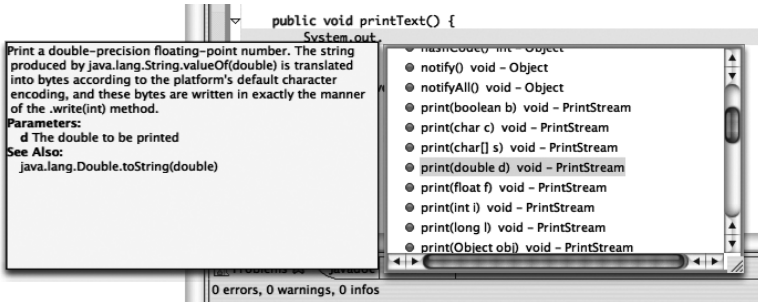


Abbildung 2.26 Eingabevollständigung

Zum Starten des Programms rufen Sie am einfachsten **Run · Run As · Java Application** auf. Eclipse sucht dann die `main()`-Methode des Projekts heraus und startet die Anwendung. Sollte der Menüpunkt nicht anwählbar sein, müssen Sie zunächst eine Konfiguration zum Ausführen anlegen, was aber zum Glück nicht weiter schwierig ist. Rufen Sie **Run · Run...** auf, wählen Sie dort links in der Liste »Java Application« aus und klicken Sie dann auf »New« – fertig (siehe Abbildung 2.27). Mit »Run« starten Sie nun Ihr Programm.

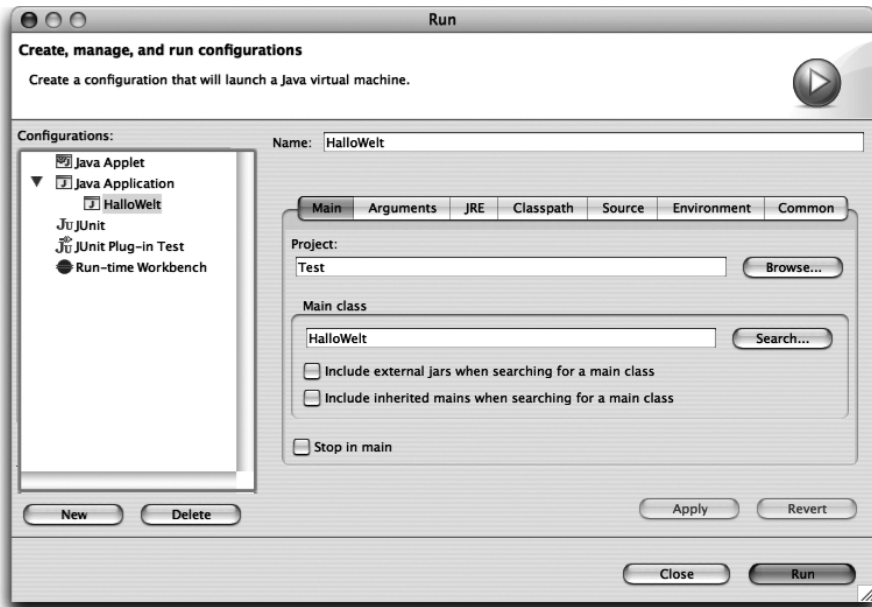


Abbildung 2.27 Ausführung vorbereiten

Damit Sie in Zukunft nicht immer wieder den Konfigurationsdialog verwenden müssen, erreichen Sie die zuletzt gestarteten Konfigurationen über **Run · Run History** oder bequemer über das grüne »Play«-Symbol in der Toolbar (siehe Abbildung 2.28).

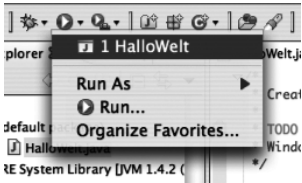


Abbildung 2.28 Java-Programm einfach ausführen

Links neben dem »Play«-Symbol finden Sie eine kleine grüne Wanze, mit der Sie Ihr Programm im Debug-Modus ausführen können. Wenn Sie in den Quelltexten Unterbrechungspunkte (Breakpoints) gesetzt haben oder Laufzeitfehler auftreten, landen Sie in der Debug-Perspektive (siehe Abbildung 2.29). Hier können Sie sich nun schrittweise durch das Programm bewegen, die Variablenwerte inspizieren oder die Ausführung abbrechen. Mit dem »Java«-Knopf oben rechts im Fenster gelangen Sie zurück zur Java-Perspektive mit der Projektstruktur und dem Quelltexteditor.

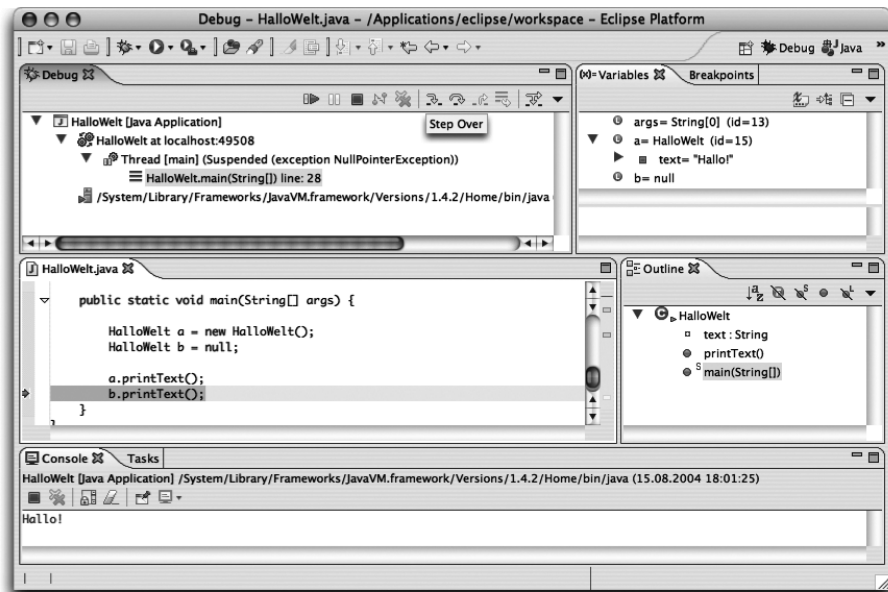


Abbildung 2.29 Fehler suchen im Debugger

2.3 NetBeans und Sun Java Studio Creator

Neben Eclipse ist Suns NetBeans eine der bekanntesten Java-Entwicklungs-umgebungen. Für eine kostenlose Umgebung ist der Funktionsumfang beträchtlich, beispielsweise ist ein Modul zur Oberflächengestaltung integriert. Und auch für Webapplikationen (Servlets, JSP) werden passende Plugins mitgeliefert. Durch die vielen Möglichkeiten gestaltet sich allerdings der Einstieg in NetBeans etwas schwieriger. Da sich die Idee der »Workspaces« bei Eclipse und der »Filesystems« bei NetBeans etwas unterscheidet, ist das Profi-Lager zwischen den Anhängern dieser beiden Umgebungen gespalten. Kaum jemand kennt die Vorzüge der jeweils anderen Umgebung – machen Sie sich daher am besten selbst ein Bild.

NetBeans 4.0 ist für den Herbst 2004 angekündigt und steht zur Zeit bereits als Beta-Version zur Verfügung. Mit dieser Version wird die Projektverwaltung komplett auf Ant umgestellt, außerdem wird – wie von vielen neuen Versionen der hier vorgestellten IDEs – Java 1.5 unterstützt. Endlich fehlen dann auch nicht mehr die wichtigsten Refactoring-Werkzeuge, die die anderen größeren Entwicklungsumgebungen schon länger mitbringen.

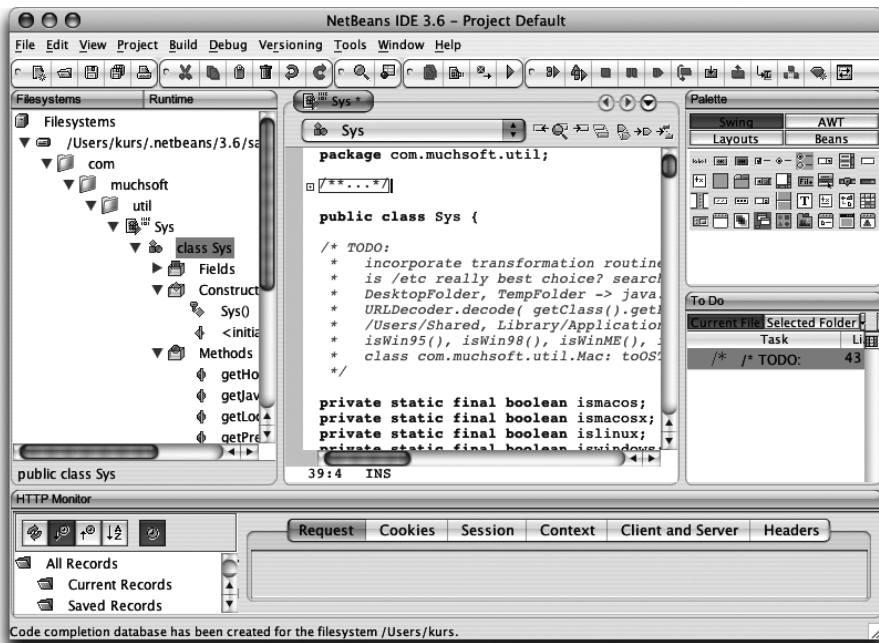


Abbildung 2.30 NetBeans 3.6

Wenn Sie NetBeans herunterladen, nehmen Sie das »Netbeans IDE«-Archiv für MacOS X (das »X« fehlt auf der Webseite, lassen Sie sich davon nicht irritieren). Im Archiv finden Sie den NetBeans Launcher, mit dem Sie NetBeans per Doppelklick starten können. Das Startprogramm ist dabei nur eine Hülle für MacOS X, die eigentliche Umgebung ist innerhalb dieser Hülle verborgen. Im Dock sehen Sie dadurch unter Umständen zweimal das NetBeans-Symbol – einmal für das Startprogramm, einmal für die Entwicklungsumgebung.

- ▶ NetBeans 3.6 – kostenlos
<http://www.netbeans.org/>

Obwohl NetBeans schon einige Funktionalität für J2EE-Entwicklungen mitbringt, hat Sun auf den NetBeans-Grundlagen aufbauend eine neue Umgebung entwickelt: Java Studio Creator (früher auch »Project Rave« genannt). Die IDE bringt erweiterte Möglichkeiten zur Entwicklung von Webapplikationen mit – unter anderem werden Java Server Faces (JSF) und aktuelle JDBC-Standards direkt unterstützt. Derzeit steht für MacOS X eine »Early Access«-Version kostenfrei zur Verfügung.

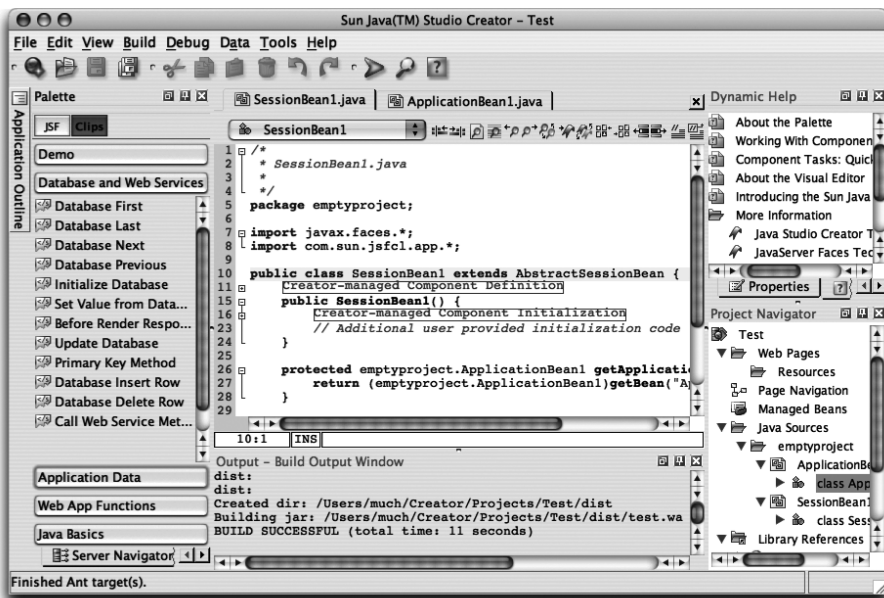


Abbildung 2.31 Sun Java Studio Creator 2004Q2

- ▶ Sun Java Studio Creator – 99 US-\$
<http://www.sun.com/software/products/jscreator/>

2.4 IntelliJ IDEA

Es folgen zwei »Geheimtipps« – weniger bekannte Umgebungen, die aber mindestens ebenso gut zur Entwicklung anspruchsvoller Java-Anwendungen geeignet sind. Leider wird ihr Einsatz von manchen Projektvorgaben verhindert, die eine der bekannteren IDEs verlangen.

Die vielleicht beste Umgebung zur schnellen und produktiven Java-Entwicklung ist IDEA. Das Programm ist mittlerweile nahezu perfekt an MacOS X angepasst und bietet viele Programmierhilfen, die die Arbeit nicht stören, sondern einem einfach so unter die Arme greifen, wie man es erwartet. Als kleine Beispiele seien die Code-Analyse oder Editierhilfen wie die hervorragende Faltung von Quelltextblöcken genannt. Abstriche muss man allerdings bei der Enterprise-Entwicklung und bei Webapplikationen machen, die nicht ganz so automatisiert unterstützt werden – immerhin können aber Tomcat und WebLogic direkt angesprochen werden.

Beim ersten Start von IDEA werden Sie aufgefordert, ein passendes JDK zu konfigurieren. Geben Sie dabei für den JDK-Pfad einfach /Library/Java/Home ein, IDEA ermittelt dann die neueste installierte Java-Version.

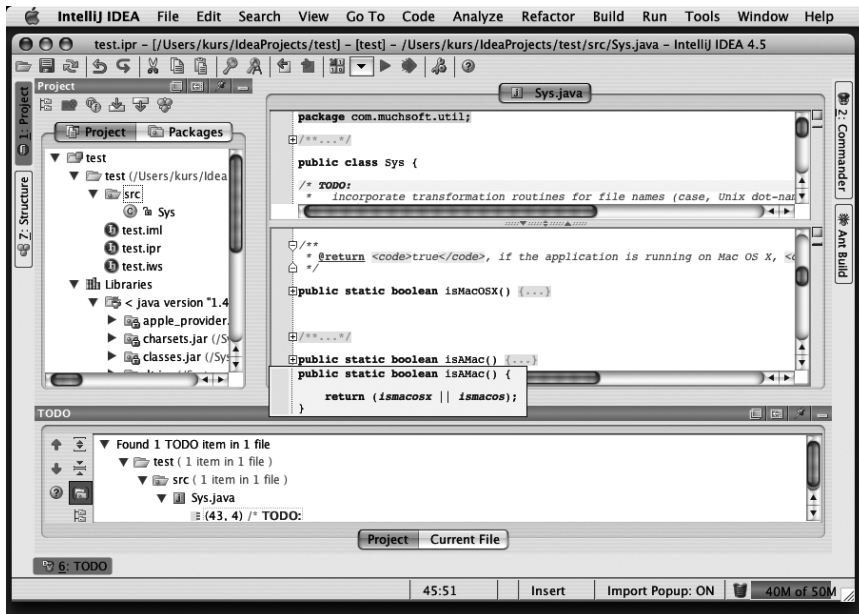


Abbildung 2.32 IntelliJ IDEA 4.5

- ▶ IntelliJ IDEA 4.5 – ca. 500 US-\$
<http://www.jetbrains.com/idea/>

2.5 OmniCore CodeGuide

Die zweite ungewöhnliche Entwicklungsumgebung ist der CodeGuide. Diese IDE eignet sich auch für Webapplikationen (Servlets, JSP) und bietet auch sonst alles, was man von einer »großen« Entwicklungsumgebung erwarten darf. Gut gelöst sind die Code-Analyse und das JSP-Debugging.

Eine einzigartige Fähigkeit ist das »Back-in-Time-Debugging«. Wenn der Debugger an einem Unterbrechungspunkt angehalten hat, können Sie einfach Schritt für Schritt durch die Anweisungen *zurück*gehen und die alten Werte inspizieren! Dadurch müssen Sie die Debugging-Session nicht jedes Mal neu starten, wenn Sie den Fehler irgendwo vor dem Unterbrechungspunkt vermuten.

Ebenso nützlich sind die Navigationshilfen im Quelltext. Wenn Sie Methoden überschreiben oder Schnittstellen implementieren, sehen Sie rechts neben den Methodenköpfen kleine Auf- und Ab-Pfeile, mit denen Sie schnell zu den entsprechenden Methoden in den Ober- und Unterklassen gelangen.

Leider ist der CodeGuide nicht ganz so gut ins System integriert, vor allem was die Tastaturkürzel betrifft. Alles in allem ist diese IDE dennoch sehr empfehlenswert.

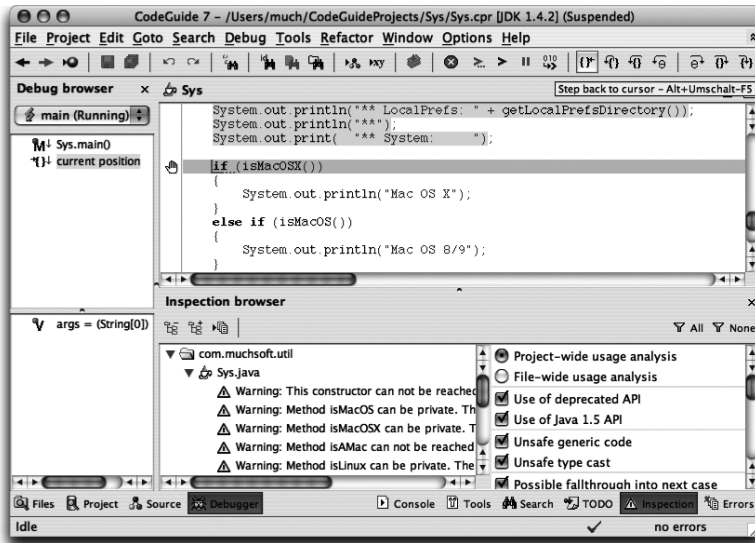


Abbildung 2.33 CodeGuide 7.0

- ▶ CodeGuide 7.0 – ca. 300 US-\$
<http://www.omnicore.com/>

2.6 Borland JBuilder

Der JBuilder ist eine der ältesten Java-Entwicklungsumgebungen – und durch die vielen Weiterentwicklungen mittlerweile auch eine der vollständigsten, gerade was den Enterprise-Bereich betrifft. Die Umgebung ist gut zu bedienen und durch die zahlreichen Assistenten auch für Einsteiger einigermaßen beherrschbar. Bereits in der kostenlosen Variante ist ein Modul zur Oberflächengestaltung enthalten. Für Bereiche wie Webapplikationen und Enterprise-Entwicklung müssen Sie eine der nicht ganz billigen Varianten erwerben – ob sich das angesichts der Konkurrenz lohnt, müssen Sie anhand Ihrer Bedürfnisse entscheiden.

Eines müssen Sie beim JBuilder unbedingt bedenken: Schon die anderen in Java geschriebenen IDEs sind nicht überragend schnell, aber verglichen dazu geht der JBuilder doch eher schwerfällig ans Werk. Damit Sie ihn sinnvoll einsetzen können, ist ein schneller, aktueller Mac mit viel RAM erforderlich.

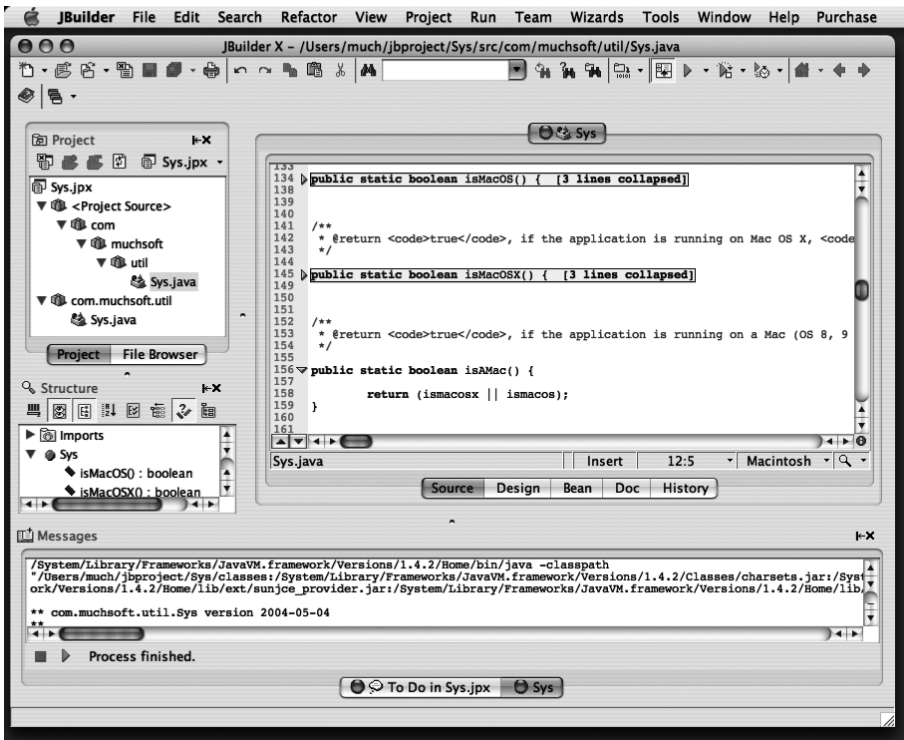


Abbildung 2.34 JBuilder X Foundation

Bereits für die Version 6 und 7 gab es spezielle Installer für Mac OS X, und auch JBuilder X und JBuilder 2005 unterstützten Mac OS X wieder explizit. Wenn Sie

die Version 8 oder 9 besitzen, finden Sie auf der Seite <http://www.visi.com/~gyles19/fom-serve/cache/437.html> eine Anleitung, wie Sie die CD-Version auf dem Mac installieren können.

- ▶ JBuilder X und JBuilder 2005 – ab ca. 500 US-\$, Foundation kostenlos
<http://www.borland.com/jbuilder/>

2.7 Borland Together Control Center und Together Solo

Together ist ein vollständiges CASE-Werkzeug, das projektbegleitend von Analyse, Design und Entwicklung bis hin zum Einsatz der Software genutzt werden kann. Als Modellierungssprache kommt dabei konsequent UML zum Einsatz – Reverse- und Round-Trip-Engineering ist bei Together so gut wie bei keiner anderen der vorgestellten Umgebungen realisiert. Together ist eine professionelle Umgebung für den effizienten Einsatz von Enterprise-Applikationen – und dies macht sich auch deutlich im Preis bemerkbar. Wenn Sie auf EJB und Web-Services verzichten können, eignet sich auch die »Solo«-Version für kleinere Projekte, aber beide Varianten sind definitiv nichts für Einsteiger.

Die Mac-Integration ist leider nicht so gut gelungen. Das Programmpaket wird in einem anderen Verzeichnis als der eigentliche Programmcode installiert. Außerdem benötigt die Software eigentlich Java 1.4, das Mac OS X-Programmpaket wird aber noch in einer Mac OS X 10.0-Version mitgeliefert und startet Together nur mit Java 1.3 – die entsprechende Warnung beim Start muss man ignorieren. All dies kann man zwar manuell anpassen, aber bei einer Software in dieser Preiskategorie darf man ein bisschen mehr Sorgfalt erwarten. Und auch bei Together gilt: Sie benötigen einen schnellen, aktuellen Mac mit viel RAM.

- ▶ Together Control Center 6.2 – ca. 5.000 US-\$
- ▶ Together Solo 6.2 – ca. 3.500 US-\$
<http://www.borland.com/together/>

2.8 Oracle JDeveloper

Auch wenn JDeveloper erst mit der kommenden Version Mac OS X offiziell unterstützen wird, lässt sich die aktuelle 10g-Version bereits problemlos einsetzen. Da die Entwicklungsumgebung vom Datenbank-Spezialisten Oracle stammt, werden natürlich ganz besonders Datenbankentwurf und generell das Arbeiten mit (PL/)SQL unterstützt. Aber auch die Enterprise-Entwicklung und Webapplikationen mit Servlets, JSP und Struts kommen nicht zu kurz.

Für die Installation unter Mac OS X laden Sie sich einfach das derzeit für Linux, Windows und Solaris angebotene ZIP-Archiv herunter. Benennen Sie dann die Datei `jdev` im Verzeichnis `jdev/bin` um in `jdev.command` – dadurch können Sie JDeveloper per Doppelklick auf diese Datei starten. Im erscheinenden Terminal-Fenster fragt Sie die Umgebung beim ersten Start nach dem Java-Pfad, geben Sie hier einfach `/usr` ein. (Diese Einstellung wird übrigens in der Datei `~/jdev_jdk` gespeichert). Nachdem die IDE komplett gestartet ist, sollten Sie noch den Menüpunkt **Help · Check for Updates...** aufrufen und das Plugin »JDeveloper on OSX Helper Addin« herunterladen. Dadurch ist unter anderem sichergestellt, dass beim Beenden der Umgebung alle Daten gespeichert werden.

Weitere Tipps zur Installation erhalten Sie direkt bei Oracle auf der Seite http://www.oracle.com/technology/products/jdev/tips/davison/jdev_mac.html. Einen allgemeinen Überblick über Oracles Mac-Unterstützung finden Sie unter der Adresse <http://www.oracle.com/technology/tech/macos/>.

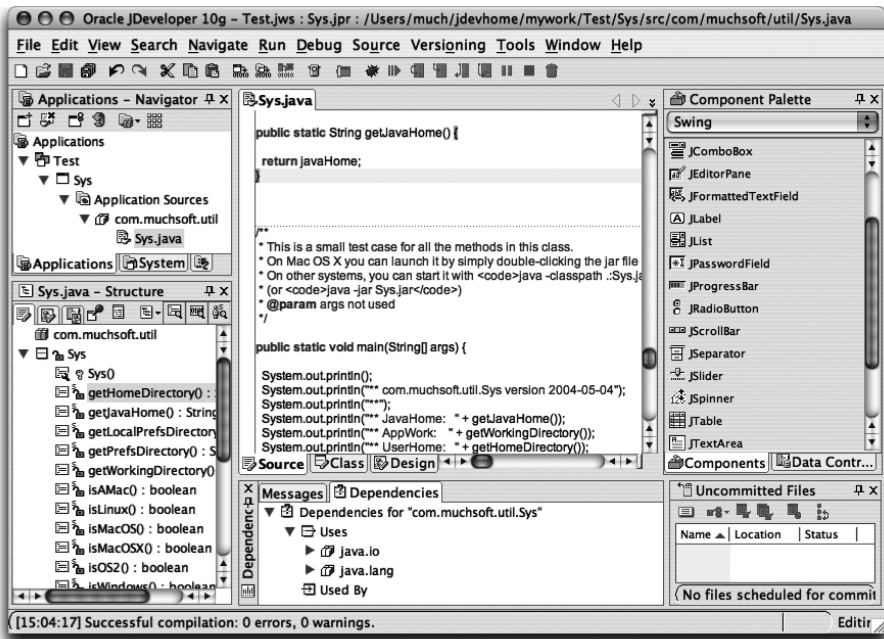


Abbildung 2.35 JDeveloper 10g 9.0.5

- ▶ Oracle JDeveloper 10g – kostenlos für nichtkommerzielle Zwecke <http://www.oracle.com/technology/products/jdev/>

2.9 Metrowerks CodeWarrior

Der CodeWarrior war viele Jahre lang die wichtigste Entwicklungsumgebung für Mac OS und anfangs auch für Mac OS X – nicht nur für Java, sondern vor allem auch für C und C++. Metrowerks ist bekannt als Compiler-Bauer (beispielsweise für den Embedded-Bereich), daher wurden alle Compiler selbst entwickelt. Der Java-Compiler unterstützt alle Versionen bis 1.4, der erzeugte Bytecode läuft dann in der Java-Laufzeitumgebung des Systems. CodeWarrior ist zum Kodieren (d. h. zum »normalen« Programmieren) eine produktive IDE, alle weiteren Fähigkeiten zur modernen Software-Entwicklung (z. B. Refactoring) fehlen aber.

In letzter Zeit wurde der CodeWarrior daher von den anderen Entwicklungsumgebungen deutlich überholt, und auch beim Compiler selbst darf man eine Java 1.5-Unterstützung wohl nicht so schnell erwarten – fremde Compiler (z. B. die von Sun bzw. Apple) lassen sich leider nicht sonderlich gut integrieren. Und da Xcode mittlerweile als Standardumgebung für Mac-spezifische Entwicklungen anzusehen ist, ist der CodeWarrior eigentlich nur noch für bestehende Projekte interessant.

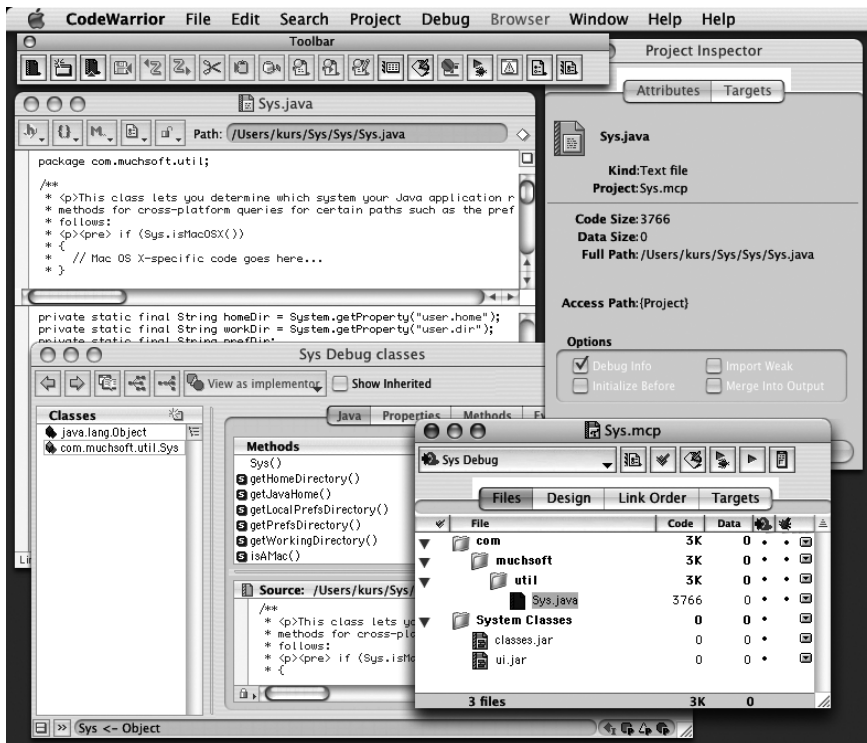


Abbildung 2.36 CodeWarrior v8

Die Entwicklung von Java-Anwendungen wird nur bis CodeWarrior v8 unterstützt, nicht aber im aktuellen CodeWarrior v9. Ob der Hersteller bei künftigen Versionen Java wieder berücksichtigt, ist nicht sicher.

- ▶ CodeWarrior Development Studio for Mac OS v8 – ab ca. 400 US-\$
<http://www.metrowerks.com/MW/Develop/Desktop/Macintosh/>

2.10 jEdit

Neben den Entwicklungsumgebungen gibt es eine Vielzahl von Programmiereditoren, die sich nicht so sehr auf die Projektverwaltung, sondern auf die möglichst effiziente Eingabe des Programmquelltextes konzentrieren. Für kleinere Projekte sind sie daher oft perfekt geeignet, und auch Einsteiger sehen sich nicht mit einem Dschungel von Konfigurationsmöglichkeiten konfrontiert. Die wichtigsten und interessantesten Editoren werden im Folgenden kurz vorgestellt.

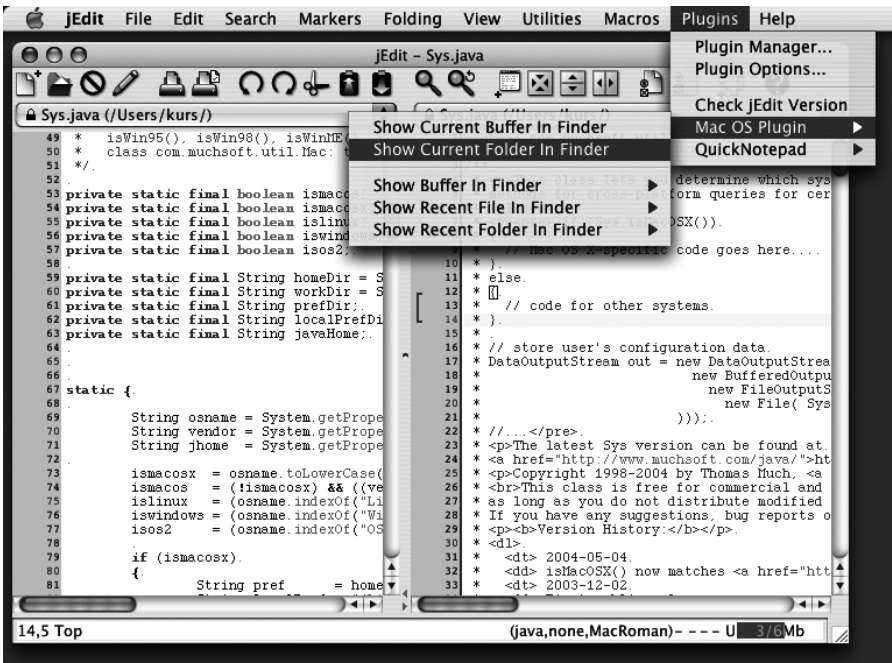


Abbildung 2.37 jEdit 4.2pre15

jEdit ist vielleicht der bekannteste systemübergreifende Programmiereditor. Er bietet Unterstützung für sehr viele Dateitypen, und über einen Plugin-Manager können – nicht nur für Java – viele Plugins nachinstalliert werden, die den Edi-

tor fast schon wieder zu einer kompletten Entwicklungsumgebung machen. An besonderen Eigenschaften bietet jEdit beliebige Zwischenpuffer, Lesezeichen (Marker) und das Einfalten von Quelltext-Blöcken.

- ▶ jEdit 4.2 – kostenlos
<http://www.jedit.org/>

2.11 Jext

Auch der Programmiereditor Jext wurde ursprünglich speziell für Java entwickelt, unterstützt aber mittlerweile recht viele Sprachen und ist brauchbar ins System integriert. Mit einem Doppelklick auf das Archiv `lib/jext-5.0.jar` können Sie Jext starten – alternativ ist der Editor auch als Java-Web-Startversion verfügbar (<http://www.jext.org/apps/jext.jnlp>). An speziellen Fähigkeiten bringt Jext Quelltext-Lesezeichen (Bookmarks) mit und hat einen Python-Interpreter fest eingebaut.

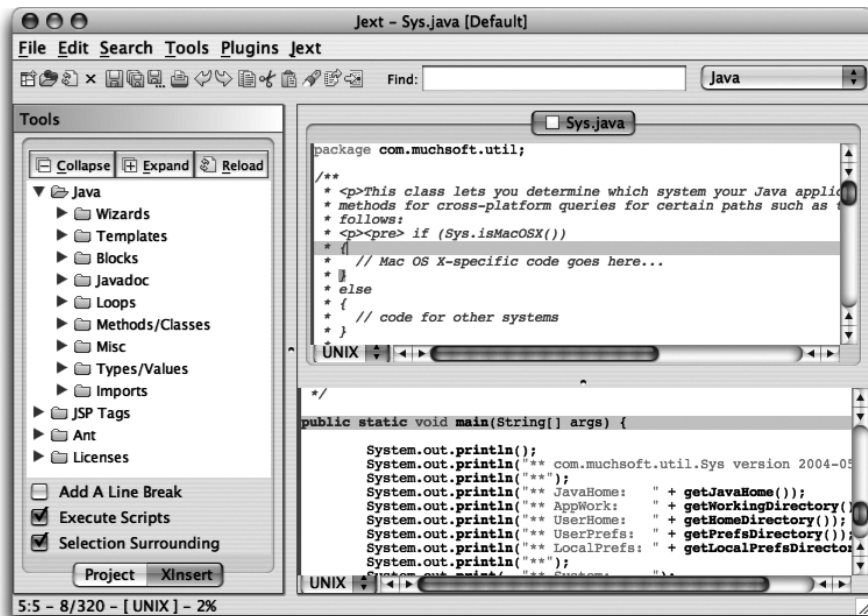


Abbildung 2.38 Jext 5.0

- ▶ Jext 5.0 – kostenlos
<http://www.jext.org/>

2.12 JEdit

JEdit ist ein recht einfacher Editor, mit dem Sie die bekannten Java-Befehle bequem aufrufen können. Kommandozeilenparameter können einfach angegeben und JAR-Archive sowie die Javadoc-Dokumentation auf Knopfdruck erzeugen werden. Zudem bietet JEdit eine rudimentäre Vervollständigung von Klassennamen während der Eingabe.

Bei Fehlermeldungen werden leider nur die Ausgaben von `javac` angezeigt, es gibt dabei keine Verknüpfung zum Quelltext. Dennoch ist dieser Editor für absolute Java-Neueinsteiger eine gute Wahl!



Abbildung 2.39 JEdit 1.1

► JEdit 1.1 – 5 US-\$

<http://homepage.mac.com/jmacmullin/>

2.13 BlueJ

BlueJ wurde von diversen Universitäten entwickelt, um objektorientierte Programmierung mit Java besser lehren zu können. Herausgekommen ist eine Entwicklungsumgebung, die einen ungewohnten Ansatz verfolgt, die aber speziell für OO-Einsteiger hervorragend geeignet ist!

Grundlage der Programmierung mit BlueJ ist ein UML-Klassendiagramm, das mit einem einfachen, aber vollkommen ausreichenden Editor entworfen wird (siehe Abbildung 2.40). Durch einen Doppelklick auf ein Klassen-Symbol wird ein Editor mit dem automatisch generierten Java-Quelltext geöffnet. Hier können Sie auch von der »Implementation« auf das »Interface« umschalten, wodurch die Javadoc-Dokumentation der Klasse erzeugt und angezeigt wird.

Zu jeder Klasse kann im Kontext-Popup-Menü ein JUnit-Testfall erzeugt werden, und jedes Klassendiagramm sieht standardmäßig ein Symbol für die Dokumentation vor.

Im Kontext-Popup-Menü können von jeder Klasse »live« Objekte in der ständig laufenden BlueJ Virtual Machine erzeugt werden, die dann in UML-Notation unten links im Modellierungsfenster auftauchen. Diese Objekte können Sie dann (wieder über das Kontext-Popup-Menü) inspizieren und per Mausklick Methoden darin aufrufen. Im interaktiven Ein-/Ausgabebereich rechts daneben können Sie Ausdrücke auswerten lassen und dabei natürlich auch auf die erzeugten Objekte zugreifen – die gerade zu Anfang schwer vermittelbare `main()`-Methode ist hier überhaupt nicht nötig.

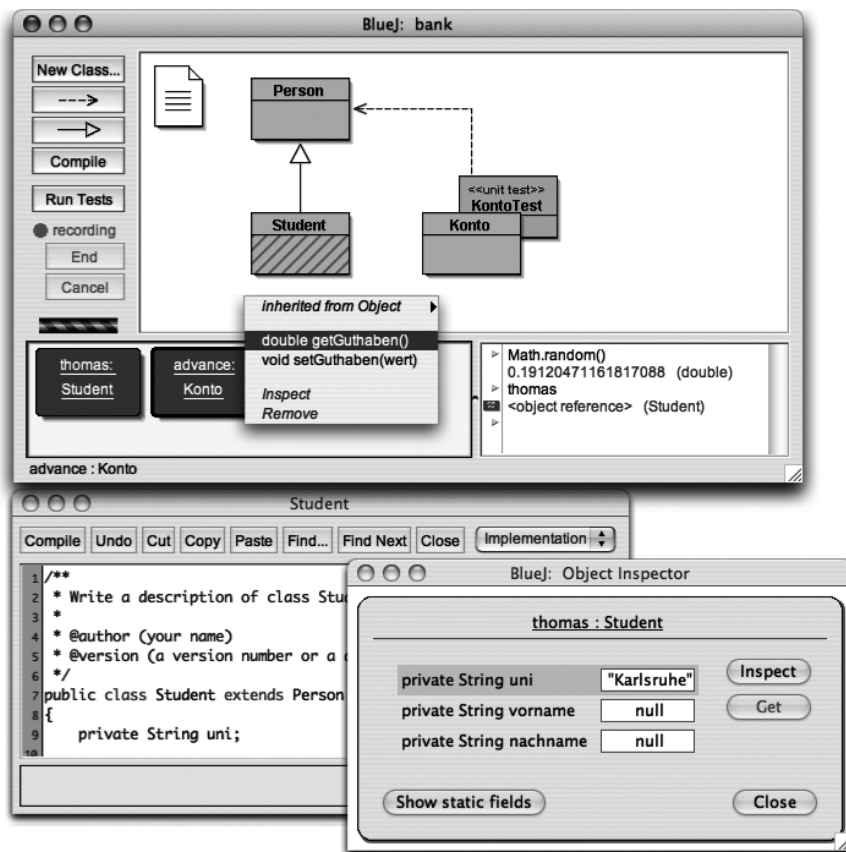


Abbildung 2.40 BlueJ 2.0 beta

- BlueJ 2.0 – kostenlos
<http://www.bluej.org/>

2.14 DrJava

Auch DrJava wurde im universitären Bereich zum Lernen von Java entwickelt. Verglichen mit BlueJ ist DrJava aber eher eine typische, gewohnte Entwicklungsumgebung. Der Quelltexteditor besitzt eine für Einsteiger hilfreiche Art, Blöcke zu markieren. Außerdem werden verschiedene Sprachstufen (Level) unterstützt – von sehr grundlegenden Java-Konstrukten bis hin zum vollen Sprachumfang. Schließlich sind auch ein interaktiver Ein-/Ausgabebereich, JUnit und ein einfacher Debugger integriert.

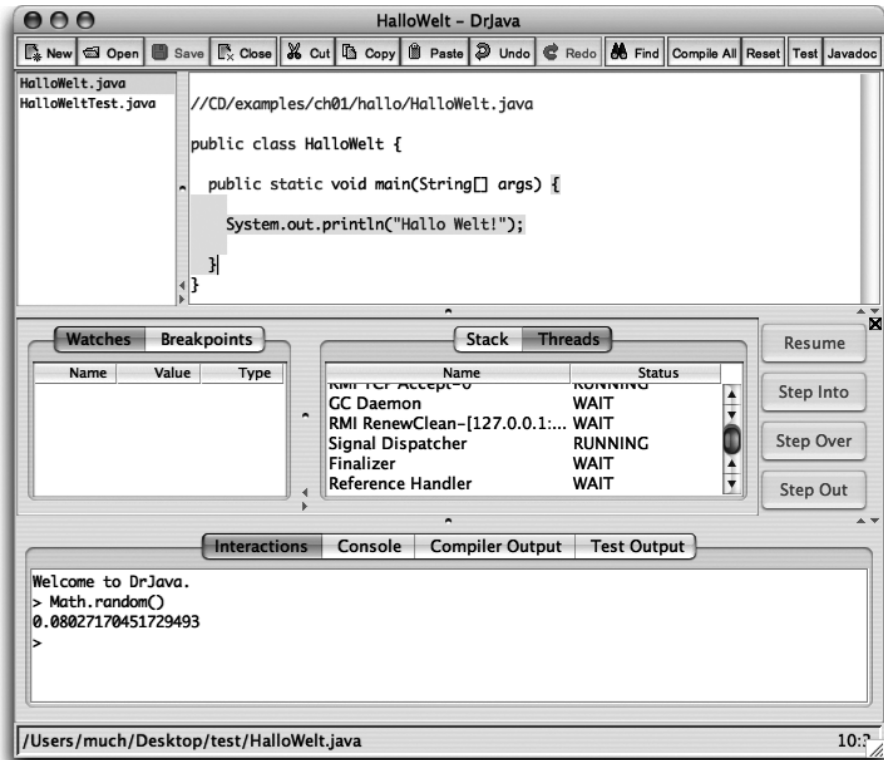


Abbildung 2.41 DrJava Version 20040707–3232

- ▶ DrJava – kostenlos
<http://drjava.org/>

2.15 Literatur & Links

► Project Builder und Xcode

- http://developer.apple.com/documentation/Java/Conceptual/Project_Builder_for_Java/

Dieses Dokument zeigt Schritt für Schritt die Entwicklung einer Java-Anwendung. Obwohl sich die Angaben auf den Project Builder beziehen, sind alle wesentlichen Informationen auch noch für Xcode gültig.

- <http://developer.apple.com/documentation/DeveloperTools/Conceptual/Xcode-QuickTour/>

Hier erhalten Sie einen schnellen Überblick über die Bedienung von Xcode. Für die praktischen Beispiele wird aber nicht Java, sondern Objective-C eingesetzt.

- <http://developer.apple.com/documentation/DeveloperTools/Conceptual/Xcode-Workflow/>

Dieses Dokument erklärt die Projektorganisation bzw. -struktur von Xcode – allerdings wieder nicht speziell für Java.

► Eclipse

- Manfred Borzechowski, »Eclipse 3 professionell«, Galileo Computing 2004
Dieses Buch ist ein umfassendes Lern- und Nachschlagewerk für den professionellen Einsatz von Eclipse 3, wobei auch die Plugin-Entwicklung behandelt wird.

- Ulrich Cuber, »Einstieg in Eclipse 3.0«, Galileo Computing 2004
Für den Einstieg in die Programmentwicklung mit der aktuellen Eclipse-Version ist dieses Buch geeignet – grundlegende Java-Kenntnisse werden aber vorausgesetzt.

► NetBeans

- Boudreau/Glick/Greene/Spurlin/Woehr, »NetBeans: The Definitive Guide«, O'Reilly 2002

Neben einem guten Überblick über die Möglichkeiten der Entwicklungsumgebung bietet dieses Buch eine umfassende Anleitung zur Programmierung von NetBeans-Plugins.

► BlueJ

- Barnes/Kölling, »Objektorientierte Programmierung mit Java«, Pearson Studium 2003

Dieses Buch bietet eine hervorragende Einführung in die objektorientierte Programmierung – wobei natürlich die besonderen Stärken der BlueJ-Umgebung genutzt werden. Nur für Java-Einsteiger.

- <http://www.bluej.org/doc/tutorial.html>

Das BlueJ-Handbuch ist online in vielen Sprachen verfügbar, nur leider noch nicht auf Deutsch.

3 Grafische Benutzungsoberflächen (GUI)

3.1	Das AWT und die Java 1.3-Apple-Erweiterungen ...	121
3.2	Swing und die Java 1.4-Apple-Erweiterungen	142
3.3	Fenster mit »Brushed Metal«-Aussehen	168
3.4	Drag & Drop	169
3.5	»Headless«-Applikationen ohne Benutzungsoberfläche	170
3.6	Literatur & Links	172

1 Grundlagen

2 Entwicklungsumgebungen

3 Grafische Benutzungsoberflächen (GUI)

4 Ausführbare Programme

5 Portable Programmierung

6 Mac OS X-spezifische Programmierung

7 Grafik und Multimedia

8 Werkzeuge

9 Datenbanken und JDBC

10 Servlets und JavaServer Pages (JSP)

11 J2EE und Enterprise JavaBeans (EJB)

12 J2ME und MIDP

A Kurzeinführung in die Programmiersprache Java

B Java auf Mac OS 8/9/Classic

C Java 1.5 »Tiger«

D System-Properties

E VM-Optionen

F Xcode- und Project Builder-Einstellungen

G Mac OS X- und Java-Versionen

H Glossar

I Die Buch-CD

3 Grafische Benutzungsoberflächen (GUI)

»Verlockend ist der äußere Schein, der Weise dringet tiefer ein.«
(Wilhelm Busch)

Apple-Rechner sind bekannt für ihre einfach zu bedienende grafische Benutzungsoberfläche (»Graphic User Interface«, GUI), die schon der erste Mac 1984 populär gemacht hat. Weil die Oberfläche für Desktop-Applikationen den wesentlichen Kontakt zum Anwender darstellt (und nicht irgendwelche Konfigurationsdateien), wird dieses Thema bereits an dieser Stelle im Buch behandelt – noch bevor Sie im nächsten Kapitel erfahren, wie ausführbare Java-Programme im Allgemeinen und Mac OS X-Anwendungen im Speziellen aufgebaut sind.

Falls Sie bisher noch keine Benutzungsoberflächen mit Java programmiert haben, erhalten Sie in den folgenden Abschnitten einen kurzen, praktischen Einstieg – allerdings liegt hierauf nicht der Schwerpunkt. Es geht bei den Oberflächen vor allem um die speziellen Anpassungen für Mac OS X, damit Java-Programme wie native Applikationen aussehen. Den Anwender interessiert es nicht, mit welcher Sprache ein Programm geschrieben wurde, er möchte einfach nur, dass eine Applikation so zu bedienen ist wie alle übrigen auf seinem Rechner! Außerdem erhalten Sie ein paar Programmiertipps, die unabhängig von Mac OS X sinnvoll sind, um Benutzungsoberflächen plattformunabhängig zu entwickeln.

Das Problem bei speziellen Systemanpassungen ist, dass die dabei verwendeten Java-Erweiterungen nicht immer auf anderen Systemen zur Verfügung stehen – die Programme sind dann also nicht mehr portabel. Allerdings können Sie Ihre Anwendungen so entwickeln, dass die speziellen Möglichkeiten auf fremden Systemen einfach nicht genutzt werden. In diesem Kapitel werden beide Arten vorgestellt: die nichtportable Anpassung, bei der die technischen Hintergründe der Anpassung besser zu verstehen sind, und anschließend die portable Anpassung, bei der die technischen Feinheiten in einer Hilfsbibliothek versteckt sind. Allgemeine Hinweise zur portablen und zur systemabhängigen Programmierung finden Sie dann in Kapitel 5, *Portable Programmierung*, und 6, *Mac OS X-spezifische Programmierung*.

Java bringt zwei Bibliotheken zur Oberflächenprogrammierung mit – das ältere »Abstract Windowing Toolkit« (AWT) und die neuere, deutlich umfangreichere Swing-Bibliothek. Beide sind Teil der so genannten »Java Foundation Classes« (JFC). Da Swing auf dem AWT aufbaut, werden die grundlegenden Konzepte im folgenden Abschnitt anhand des AWT erläutert. Apple stellt seit langem

Klassen zur Verfügung, mit denen Java-Programme auf systemspezifische Ereignisse reagieren können (beispielsweise das Anwählen bestimmter vordefinierter Menüeinträge oder das Öffnen einer Datei, die der Benutzer auf das Programmsymbol zieht). Leider werden für Java 1.3 und für Java 1.4 jeweils andere Klassen benötigt, daher werden hier beide Varianten vorgestellt.

Ganz am Ende dieses Kapitels erhalten Sie noch Informationen über Programme, die das genaue Gegenteil einer GUI-Anwendung sind – Server-Applikationen, die komplett im Hintergrund laufen und keine Menüs, ja nicht einmal ein Dock-Symbol besitzen.

Und bevor es mit der Programmierung losgeht, hier noch einige wichtige Begriffe und Hinweise für Einsteiger, die sowohl das AWT als auch Swing betreffen. Oberflächen setzen sich aus Komponenten und Containern zusammen. **Komponenten** (Unterklassen von `java.awt.Component` oder `javax.swing.JComponent`) repräsentieren Dialogelemente, die Sie sicherlich aus diversen Applikationen kennen: Schaltflächen (Buttons), Kontrollkästchen (Checkboxes), Listen, Texteingabefelder usw. **Container** (`java.awt.Container`) fassen Komponenten zusammen, um sie zu gruppieren und anzuordnen – jedes Fenster ist beispielsweise ein Container. Fenster werden bei Java »Frames« (`java.awt.Frame` bzw. `javax.swing.JFrame`) genannt und dienen unter anderem zur Darstellung von nichtmodalen Dialogformularen. Für modale Dialoge kommen die Klassen `java.awt.Dialog` bzw. `javax.swing.JDialog` zum Einsatz, wobei letztere häufig über die Klasse `javax.swing.JOptionPane` angesprochen wird. Nach Möglichkeit sollten Sie die nicht-modalen Frames verwenden. Container sind selbst Komponenten, d.h., ein Container kann auch weitere Container enthalten. Häufig wird dies bei Panels (`java.awt.Panel` bzw. `javax.swing.JPanel`) verwendet, die nur zur Zusammenfassung anderer Komponenten dienen und nicht selbst auf dem Desktop dargestellt werden können – dazu müssen die Panels in einen passenden Container eingefügt werden, z.B. in einen Frame. Durch das hinzufügen zu Containern ist zwar eine Gruppierung der Komponenten vorgegeben, nicht aber die Anordnung auf dem Bildschirm. Dazu verwendet jeder Container einen so genannten **Layout-Manager**, der anhand relativer Vorgaben (»links von«, »unterhalb« usw.) die Anordnung der Komponenten im Container berechnet – Sie müssen die Komponenten also nicht pixelgenau positionieren und gewinnen dadurch eine Auflösungsunabhängigkeit der Benutzungsoberfläche. Typische Layout-Manager sind `java.awt.BorderLayout` oder `java.awt.FlowLayout`. Es gibt auch Layout-Manager im Paket `javax.swing` (beispielsweise `BoxLayout`), und glücklicherweise können das AWT und Swing gegenseitig die Layout-Manager des anderen Pakets nutzen. Schließlich fehlt noch die Ereignis-Behandlung (»event handling«) – Darstellung und Interaktion

sind beim AWT und mehr noch bei Swing voneinander getrennt, da beide Bibliotheken das MVC-Muster umsetzen (»Model, View, Controller«; Datenmodell/Geschäftslogik, Darstellung, Steuerung). Die Ereignis-Behandlung wird nicht von den Komponenten selbst durchgeführt (beispielsweise wenn eine Schaltfläche angeklickt wird), sondern sie benachrichtigen stattdessen Steuerungsobjekte, die sich um die Interaktion kümmern. Diese Steuerungsobjekte werden **Listener** (»Lauscher«) genannt, und für jedes Ereignis kann es einen eigenen Listener geben (natürlich lassen sich Ereignisse auch zusammenfassen und dann von einem gemeinsamen Listener-Objekt behandeln). Damit eine Komponente weiß, dass sie bei einem Ereignis einen Listener benachrichtigen soll, müssen Sie den Listener an der Komponente registrieren, wofür die Komponenten – je nach unterstützten Ereignis-Typen – passende Methoden zur Verfügung stellen.

3.1 Das AWT und die Java 1.3-Apple-Erweiterungen

Das »Abstract Windowing Toolkit« (AWT) definiert Klassen im Paket `java.awt` und in zahlreichen Unterpaketen davon. Diese GUI-Bibliothek ist bereits seit Java 1.0 fester Bestandteil der Standard-Klassenbibliothek, allerdings sollten Sie bei neuen Projekten nach Möglichkeit die Swing-Bibliothek einsetzen, die deutlich umfassender ist – beispielsweise bietet das AWT standardmäßig keine Komponente zur Darstellung von Tabellen. Dennoch wird das AWT hier vorgestellt, da es die Grundlage für Swing bildet und die verwendeten Konzepte für beide Bibliotheken gültig sind.

Zusammen mit dem AWT werden die alten Apple-Erweiterungen beschrieben, mit denen eine Java-Anwendung besser ins Mac-System integriert werden kann. Die bis Java 1.3 unterstützten Erweiterungen stammen noch aus Mac OS (Classic)-Zeiten, wo Apples Java-Implementierung den Namen »MRJ« trug – daher stammt auch der Paketname `com.apple.mrj`. Auch wenn diese Erweiterungen derzeit noch mit Java 1.4 funktionieren, gelten sie als veraltet – nehmen Sie dort besser die neuen Erweiterungen, die im folgenden Abschnitt vorgestellt werden. Wenn Sie allerdings Anwendungen entwickeln, die noch mit Java 1.3 laufen müssen, haben Sie keine Wahl – die MRJ-Erweiterungen stellen dann die einzige Möglichkeit zur Systemintegration dar.

AWT-Anwendungen können auch die neuen Erweiterungen nutzen, und genauso können Sie die alten Erweiterungen auch zusammen mit Swing einsetzen – wichtig ist jeweils nur die Java-Version, mit der die Programme ausgeführt werden. Aus Gründen der Einfachheit und Übersichtlichkeit werden in diesem Kapitel aber nicht alle Kombinationen gezeigt.

3.1.1 Eine einfache AWT-Anwendung

Um das Grundgerüst für ein AWT-Programm zu bekommen, legen Sie in Xcode einfach ein neues Projekt mit dem Typ »Java AWT Application« an – die generierten Quelltexte können Sie sofort übersetzen und starten. Das folgende Beispiel verwendet diesen Projekttyp, um eine ausführbare Mac OS X-Applikation zu erzeugen, allerdings werden die generierten Quelltexte komplett durch die hier gezeigten Listings ersetzt. Die Dokumentation des Beispiels konzentriert sich auf die Oberflächenprogrammierung; wie der Projekttyp konfiguriert ist, um ein ausführbares Mac OS X-Programm zu erhalten, ist im folgenden Kapitel 4, *Ausführbare Programme, dokumentiert*.

Das Beispiel zeigt einen einfachen, nichtmodalen Dialog sowie eine Menüzeile, die alle Mac-typischen Einträge enthält (siehe Abbildung 3.1). Damit der Quelltext nicht zu groß und zu unübersichtlich wird, ist nicht für alle Elemente die Funktionalität ausprogrammiert. Die Listings sind sinnvoll gekürzt abgedruckt, aber auf der Buch-CD finden Sie natürlich das vollständige Beispiel. Beachten Sie, dass die Reihenfolge der gezeigten Quelltext-Abschnitte nicht zwingend die Reihenfolge im Projekt wiedergibt – logisch zusammenhängende Teile werden hier auch zusammen gezeigt und besprochen.



Abbildung 3.1 AWT-Beispielapplikation

```
//CD/examples/ch03/AWTTest/AWTTest.java
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import com.apple.mrj.*;
import com.muchsoft.util.Sys;

public class AWTTest extends Frame
    implements MRJAboutHandler, MRJOpenApplicationHandler,
```

```
MRJOpenDocumentHandler, MRJPrefsHandler,
MRJPrintDocumentHandler, MRJQuitHandler {
```

Die Hauptklasse ist eine Unterklasse von `Frame`, ist also selbst das Dialogfenster, das später angezeigt wird. Um die Anwendung in das System zu integrieren, muss sie auf bestimmte System-Ereignisse reagieren, wofür jeweils ein passendes Interface aus dem Paket `com.apple.mrj` implementiert wird – beispielsweise gibt es den `MRJQuitHandler` mit der Methode `handleQuit()`, die aufgerufen wird, wenn der Anwender den Eintrag »AWTTest beenden« im von Mac OS X verwalteten Programm-Menü aufruft. Die einzelnen Interfaces werden weiter unten bei ihrer Implementierung genauer beschrieben.

```
public AWTTest() {
    this.erzeugeDialog();
    this.erzeugeMenue();

    this.setResizable( false );
    this.setLocation( 20, 40 );
    this.pack();

    if (Sys.isMacOSX()) {
        MRJApplicationUtils.registerAboutHandler(this);
        MRJApplicationUtils.registerOpenApplicationHandler(this);
        MRJApplicationUtils.registerOpenDocumentHandler(this);
        MRJApplicationUtils.registerPrefsHandler(this);
        MRJApplicationUtils.registerPrintDocumentHandler(this);
        MRJApplicationUtils.registerQuitHandler(this);
    }

    this.addWindowListener( new FensterSchliesser() );
}
```

Der Konstruktor baut zunächst die Oberfläche zusammen, was in zwei Methoden ausgelagert ist. Dabei wird nicht nur der Dialog, sondern auch die Menüzeile erzeugt – jedes Menü muss einem `Frame` zugeordnet sein, und jeder `Frame` kann dementsprechend sein eigenes Menü mitbringen. Was dies für Anwendungen mit mehr als einem Fenster bedeutet, gerade wenn Sie Apples Gestaltungsrichtlinien befolgen wollen, wird im folgenden Abschnitt beim Swing-Beispiel erläutert. Anschließend wird das Fenster auf eine feste Pixel-Position gesetzt (ausgehend vom Nullpunkt links oben auf dem Bildschirm) und mit `pack()` auf seine minimal mögliche Größe – unter Beachtung der Komponenten-Anordnung – gebracht.

Danach geschieht die eigentliche Integration ins System. Mit diversen `registerXXXHandler()`-Methoden wird dem System mitgeteilt, welches Java-Objekt auf welches System-Ereignis reagiert. In diesem Fall werden alle Ereignisse an das Fenster-Objekt selbst gemeldet, das ja auch alle zugehörigen Interfaces implementiert. Die Abfrage auf Mac OS X ist hier eigentlich überflüssig, denn der vorgestellte Quelltext ist nicht portabel und kann sowieso nur auf Mac OS X eingesetzt werden – Sie sehen dadurch aber schnell, welche Stellen die Systemanpassung betreffen und was daher vom portablen Ansatz weiter unten in diesem Abschnitt abgeändert werden wird.

Zum Schluss wird ein Listener-Objekt am `Frame`-Objekt registriert, das darauf achtet, ob der Anwender das Schließfeld des Fensters angeklickt hat. Die Methode dafür heißt `addWindowListener()`, denn `Frame` ist eine Unterklasse von `Window` – letztere repräsentiert allerdings nur einen rechteckigen Bildschirmbereich ohne Fensterelemente, und Sie werden fast nie direkt mit `Window`-Objekten zu tun haben. Das Listener-Objekt wird von der inneren Klasse `FensterSchliesser` definiert:

```
class FensterSchliesser extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        schliesseFenster();
    }
}

private void schliesseFenster() {
    this.setVisible( false );
    this.dispose();
}
```

`WindowListener` müssen das gleichnamige Interface aus dem Paket `java.awt.event` implementieren, das sieben Methoden deklariert – wie Sie oben sehen, ist hier aber nur eine einzige Methode von Interesse: `windowClosing()`. Diese Methode wird aufgerufen, sobald der Anwender das Schließfeld des Fensters angeklickt hat. In dieser Methode können Sie nun entscheiden, ob das Fenster geschlossen werden darf – hier wird immer die Methode `schliesseFenster()` der umgebenden Klasse `AWTTest` aufgerufen, die das Fenster unsichtbar macht und die Fenster-Ressourcen freigibt. Damit Ihre Listener-Klasse nun nicht auch noch die sechs weiteren Methoden implementieren muss, die Sie gar nicht benötigen, lassen Sie den Listener von der passenden Adapter-Klasse (hier `WindowAdapter`) erben, die einfach alle Interface-Methoden leer implementiert – Sie müssen dann nur noch die gewünschte Methode sinnvoll überschreiben.


```
private void erzeugeDialog() {
    this.setLayout( new BorderLayout( 12, 12 ));

    Panel panelInhalt = new Panel( new GridLayout( 5,2 ));
    Panel panelButtons = new Panel( new GridLayout( 1,2,12,0 ));
    Panel panelSouth = new Panel(
        new FlowLayout( FlowLayout.RIGHT ));
}
```

Beim Erzeugen des Dialogs wird dem Fenster zunächst ein geeigneter Layout-Manager zugewiesen – ein `java.awt.BorderLayout` mit einem horizontalen und vertikalen Abstand von 12 Pixeln zwischen den Komponenten (ein `Frame` besitzt zwar standardmäßig ein `BorderLayout`, das aber keinen Abstand zwischen den Komponenten lässt). Weil dieser Layout-Manager nur fünf Elemente speichern kann (an den Positionen Nord, Süd, West, Ost und Mitte – ansprechbar über die `BorderLayout`-Konstanten `NORTH`, `SOUTH`, `WEST`, `EAST` und `CENTER`), werden ein paar Panels angelegt, in denen die eigentlichen Komponenten (Schaltflächen und Texte) verwaltet werden. Den Panels wird als Konstruktor-Parameter der gewünschte Layout-Manager übergeben, hier entweder das `GridLayout` (ein Gitter mit einer festen Anzahl von Zeilen und Spalten) oder das `FlowLayout`, das die Komponenten wie die Worte in einer Textverarbeitung von links nach rechts anordnet.

```
Checkbox cbMaus = new Checkbox( "Maus", true );
Checkbox cbMonitor = new Checkbox( "Monitor" );
// ...
```

Nun werden die Dialog-Komponenten erzeugt: diverse Kontrollkästchen, die entweder ein- oder ausgeschaltet sind, je nachdem ob der boolesche Wert `true` übergeben wird oder nicht.

```
CheckboxGroup cbgDAU = new CheckboxGroup();
Checkbox rbHoch = new Checkbox( "normal hoch",true,cbgDAU );
Checkbox rbMittel = new Checkbox( "mittel", false, cbgDAU );
// ...
```

Die Klasse `Checkbox` kann nicht nur Kontrollkästchen-Objekte erzeugen, sondern auch so genannte »Radio-Buttons«. Dazu wird allen zusammengehörigen Radio-Knöpfen dasselbe `CheckboxGroup`-Objekt übergeben. Von allen Knöpfen dieser Gruppe kann dann nur einer zur selben Zeit aktiv sein.

```
Button buttonAbbrechen = new Button( "Abbrechen" );
Button buttonOK = new Button( "OK" );
```

```

        buttonAbbrechen.addActionListener( new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                schliesseFenster();
            }
        });

```

Der Benutzer soll einen Dialog natürlich auch beenden können, wofür in diesem Fall die beiden Schaltflächen »Abbrechen« und »OK« angeboten werden. Achten Sie auf die Bezeichnung der ersten Schaltfläche: Beim Mac wird die Tätigkeit angegeben, nicht das Substantiv – aus diesem Grund heißt der OK-Knopf häufig auch »Drucken«, »Sichern« usw.

Dann wird ein Listener-Objekt am Abbrechen-Knopf registriert, so ähnlich, wie Sie es schon beim `WindowListener` weiter oben gesehen haben. Dieses Mal wird dazu aber keine separate (innere) Klasse, sondern eine so genannte anonyme Klasse verwendet. `new ActionListener() { ... }` erzeugt dabei ein neues Objekt einer nicht benannten Klasse, die das `ActionListener`-Interface implementiert. Der Quelltext für diese Klasse folgt unmittelbar darauf in geschweiften Klammern. Das Interface verlangt nur eine einzige Methode `actionPerformed()`, die einfach nur die bereits vorgestellte Methode `schliesseFenster()` aufruft.

```

        panelInhalt.add( new Label( "Prüfen:" ) );
        panelInhalt.add( cbMaus );
        panelInhalt.add( rbDAU );
        // ...

        panelButtons.add( buttonAbbrechen );
        panelButtons.add( buttonOK );

        panelSouth.add( panelButtons );

        this.add( panelInhalt, BorderLayout.CENTER );
        this.add( panelSouth, BorderLayout.SOUTH );
    }

```

Schließlich werden die erzeugten Komponenten mit der `add()`-Methode zu den Panels und die Panels zum `Frame` hinzugefügt. Wenn ein Layout-Manager eine Positionsangabe verlangt, wird sie bei `add()` als Parameter übergeben (hier beispielsweise `BorderLayout.CENTER`), ansonsten ist die Reihenfolge beim Hinzufügen wichtig. Beim Mac sollte die Schaltfläche, die zum erfolgreichen Verlassen des Dialogs führt, unten rechts im Dialog auftauchen, daher wird der Abbrechen-Knopf zuerst (und damit links vom OK-Knopf) eingefügt.

```
private void erzeugeMenue() {
    MenuBar mbar = new MenuBar();

    Menu menuAblage    = new Menu( "Ablage" );
    Menu menuBearbeiten = new Menu( "Bearbeiten" );
    Menu menuFenster   = new Menu( "Fenster" );
    Menu menuHilfe     = new Menu( "Hilfe" );

    mbar.add( menuAblage );
    mbar.add( menuBearbeiten );
    mbar.add( menuFenster );
    mbar.add( menuHilfe );
    mbar.setHelpMenu( menuHilfe );

    this.setMenuBar( mbar );
}
```

Für eine Menüzeile müssen Sie zunächst ein `MenuBar`- und für die darin enthaltenen Menüs jeweils ein `Menu`-Objekt erzeugen. Die hier gezeigten vier Menüs sollte jede Mac-Anwendung zusätzlich zum automatisch verwalteten Programm-Menü besitzen. Danach fügen Sie die Menüs mit `add()` in die Menüzeile ein.

Mit `setHelpMenu()` können Sie der Menüzeile sagen, welches Menü das Hilfe-Menü ist. MacOS X nutzt diese Information zwar nicht, es gibt aber Systeme, die dieses Menü dann speziell darstellen (beispielsweise ganz rechts in der Menüzeile).

Damit das Fenster seine Menüzeile kennt, müssen Sie diese schließlich noch mit `setMenuBar()` beim `Frame` anmelden.

```
MenuItem miOpen = new MenuItem( "Öffnen...",
                                new MenuShortcut( KeyEvent.VK_O ) );
MenuItem miClose = new MenuItem( "Schließen",
                                  new MenuShortcut( KeyEvent.VK_W ) );
menuAblage.add( miOpen );
menuAblage.add( miClose );
// ...

miClose.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        schliesseFenster();
    }
}
```

```

));
// ...

```

Pro Menüpunkt wird nun ein `MenuItem`-Objekt erzeugt. Zusätzlich zur Beschriftung können Sie einen `MenuShortcut` übergeben, das Tastaturkürzel für diesen Eintrag. Das Kürzel wird dabei mit den `VK_XXX`-Konstanten (»virtual keys«) aus der Klasse `KeyEvent` festgelegt. Welche Umschalttaste zusätzlich zu diesem Buchstaben gedrückt werden muss, hängt vom Betriebssystem ab (`Strg` bei Windows, `⌘` bei Mac OS X). Wenn beim Tastaturkürzel zusätzlich `⇧` gedrückt werden soll, übergeben Sie an `MenuShortcut` den Wert `true` als zweiten Konstruktor-Parameter.

Wie schon beim Abbrechen-Knopf wird dann noch ein Listener-Objekt pro Menüeintrag registriert.

```

MenuItem miDock = new MenuItem( "Im Dock ablegen",
                                new MenuShortcut( KeyEvent.VK_M ));
menuFenster.add( miDock );
miDock.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        setState( Frame.ICONIFIED );
    }
});

```

Mit dem Menüpunkt »Im Dock ablegen« wird das Fenster minimiert – denselben Effekt erzielen Sie, wenn Sie den gelben Knopf oben links im Fensterahmen anwählen. Während hier `setState()` verwendet wird, sollten Sie ab Java 1.4 `setExtendedState()` nutzen.

```

if (!Sys.isMacOSX()) {
    MenuItem miAbout = new MenuItem( "Über AWTTTest" );
    miAbout.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            ProgrammInfo.zeigeInfo();
        }
    });
    menuHilfe.addSeparator();
    menuHilfe.add( miAbout );
}
}

```

Nun folgt eine Anpassung für fremde Systeme. Während Mac OS X das Programm-Menü mit dem »Über...«-Eintrag automatisch verwaltet, müssen Sie anderswo selber einen passenden Menüpunkt bereitstellen. Dazu wird in der

Regel ein Eintrag an das Hilfe-Menü angehängt. Und damit sich die Programm-Informationen deutlicher von den übrigen Hilfe-Einträgen abheben, wird mit `addSeparator()` noch eine Trennlinie eingefügt. Der Menüpunkt ruft bei seiner Anwahl eine statische Methode der Klasse `ProgrammInfo` auf, die sich um die Verwaltung des Über-Dialogs kümmert. Zum Erzeugen des Dialogs wird intern folgender Konstruktor genutzt:

```
//CD/examples/ch03/AWTTTest/ProgrammInfo.java
// ...
private ProgrammInfo() {
    // ...
    this.add( new Label( "Copyright \u00a92004 Thomas Much",
                        Label.CENTER ), BorderLayout.SOUTH );

    this.pack();
    Dimension screen = this.getToolkit().getScreenSize();
    Dimension dialog = this.getSize();
    this.setLocation( (screen.width - dialog.width) / 2,
                     (screen.height - dialog.height) / 2 );

    // ...
}
```

Unter anderem wird ein Copyright-Hinweis dargestellt. Sonderzeichen, die nicht direkt als einzelne Taste zur Verfügung stehen, müssen dabei mit ihrer Unicode-Escape-Sequenz eingegeben werden – in diesem Fall wird mit `\u00a9` das Copyright-Zeichen »©« kodiert.¹

Danach wird der Dialog auf dem Bildschirm zentriert. Mit `Toolkit.getScreenSize()` fragen Sie die Bildschirmgröße ab, mit `Frame.getSize()` die Größe des Dialogs. Aus beiden Werten können Sie die gewünschten Pixelwerte für die linke obere Ecke des Fensters errechnen.

Zurück zum `AWTTTest`-Quelltext. Damit der Dialog die Komponenten nicht direkt an den Fensterrändern darstellt, überschreiben Sie die Methode `getInsets()` und fügen einige Pixel Abstand ein – hier zwölf Pixel an jeder Seite:

```
//CD/examples/ch03/AWTTTest/AWTTTest.java
// ...
public Insets getInsets() {
    Insets i = super.getInsets();
    i.top += 12; i.left += 12; i.bottom += 12; i.right += 12;
}
```

¹ Diverse Listen mit den erlaubten Zeichenwerten finden Sie auf <http://www.unicode.org/>.

```

        return i;
    }

    public void handleAbout() {
        new Thread() {
            public void run() {
                ProgrammInfo.zeigeInfo();
            }
        }.start();
    }
}

```

Jetzt folgt die Methode `handleAbout()`, die bei Anwahl des »Über...«-Menüpunkts im Programm-Menü aufgerufen wird. Dass diese Methode aufgerufen wird, haben Sie im Konstruktor mit `registerAboutHandler()` festgelegt. An dieser Stelle wird wieder der `ProgrammInfo`-Dialog angezeigt, den Sie gerade eben schon beim »Über...«-Eintrag für fremde Systeme genutzt haben. Wenn Sie den `MRJAboutHandler` nicht registrieren, zeigt dieser Menüpunkt einen Standarddialog mit dem Programmnamen an.

Auffallend ist hier der `Thread`, in dem der Dialog-Aufruf stattfindet. Dies ist ganz einfach eine Korrektur für einen Fehler, der bis `Mac OS X 10.1` zu einem Programm-Hänger führen konnte, wenn in einer Handler-Methode ein modaler Dialog angezeigt wurde. Auch wenn der Fehler seit `Mac OS X 10.2` nicht mehr auftritt und Sie eventuell gar keine modalen Dialoge verwenden, ist es gängige Praxis geworden, diese Technik bei Apples `Java 1.3`-Erweiterungen immer zu nutzen – schaden tut der Code bei neueren Versionen nicht.

Damit der »Über...«-Menüpunkt im Programm-Menü auftaucht, mussten Sie bis `Mac OS X 10.1` nicht nur den `MRJAboutHandler` registrieren, sondern auch die System-Property `com.apple.mrj.application.apple.menu.about.name` auf den Programmnamen setzen. Seit `Java 1.4` gilt diese Property allerdings als veraltet! Zum Glück gibt es bereits seit `Mac OS X 10.2` eine bessere Art, den Programmnamen anzugeben: Wie im folgenden Kapitel ausführlich beschrieben wird, setzen Sie einfach in der Datei `Info.plist` eines Programmpakets den Schlüssel `CFBundleName` auf die gewünschte Zeichenkette. Wenn Sie ein `Xcode`-»Java Application«-Projekt verwenden, brauchen Sie sich darum aber gar nicht zu kümmern. In den Target-Einstellungen ist dann unter **Info.plist Entries · Simple View · Display Information · Display name** automatisch der Projektname als Programmname eingetragen.

```

public void handlePrefs() {
    new Thread() {

```

```

        public void run() {
            // Einstellungen-Dialog öffnen...
        }
    }.start();
}

```

Im Programm-Menü einer Mac-Applikation gibt es üblicherweise auch einen Eintrag »Einstellungen...«. Java-Programme zeigen diesen Eintrag nur an, wenn Sie einen `MRJPrefsHandler` registrieren! Dieser Handler kam erst mit dem Java 1.3.1 Update 1 hinzu, steht also auf dem ursprünglichen Mac OS X 10.1 und früher nicht zur Verfügung.

Wenn Sie sich um solche Versionsprobleme – welche Methode kann seit welchem Java-Update genutzt werden? – und um die Einzelheiten der Systemintegration nicht kümmern möchten, können Sie auf diverse Bibliotheken zurückgreifen, die Ihnen die Überprüfung abnehmen und gegebenenfalls Ersatzroutinen zur Verfügung stellen. Die bekannteste Bibliothek ist sicherlich der **MRJAdapter**, den Sie von der Seite <http://www.roydesign.net/mrjadapter/> herunterladen können.

```

public void handleOpenApplication() {
    System.out.println( "Applikation ohne Parameter gestartet" );
}

public void handleOpenFile(File datei) {
    System.out.println( "Datei " + datei + " öffnen..." );
}

```

Zwei weitere Methoden, die von den Schnittstellen `MRJOpenApplicationHandler` und `MRJOpenDocumentHandler` deklariert werden, werden beim Programmstart aufgerufen bzw. immer dann, wenn der Anwender Dateien auf das Programmsymbol zieht. Damit das Programmsymbol darauf gezogene Dateien akzeptiert, müssen Sie in Xcode in den Target-Einstellungen festlegen, welche Dateitypen erlaubt sind (siehe Abbildung 3.2). Die Konfiguration der Dokumenttypen wird in Kapitel 4, *Ausführbare Programme*, noch ausführlich beschrieben – hier reicht es zunächst vollkommen aus, wenn Sie einfach das Sternchen »*« im Feld »Extensions« eintragen, d.h., Ihr Programm akzeptiert alle Dateitypen.

Ein kleines Problem ist die Zuverlässigkeit dieser Methoden. Während bei Java 1.3 die Aufrufe wie erwartet erfolgen (`handleOpenApplication()` einmal

beim Programmstart; `handleOpenFile()` jedes Mal, wenn der Benutzer eine Datei auf das Programmsymbol zieht – auch beim Programmstart), verhält sich Java 1.4 recht seltsam – einzig `handleOpenFile()` wird vernünftig gemeldet, wenn das Programm bereits läuft. Andererseits gelten diese alten Handler ab Java 1.4 für veraltet, und die neuen Erweiterungen funktionieren mit Java 1.4 problemlos.

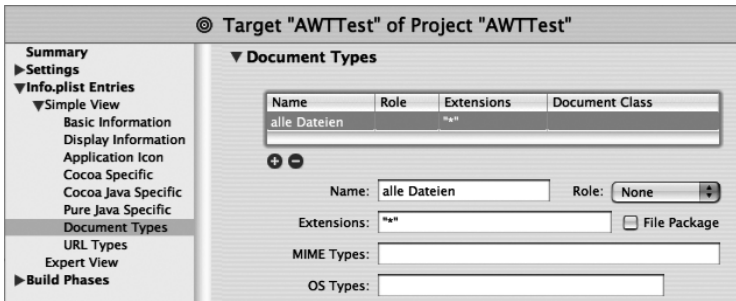


Abbildung 3.2 Dokumenttypen für `handleOpenFile()` definieren

Und ein Hinweis noch zu den Dateien, die der Anwender beim Starten des Programms auf das Programmsymbol zieht: Diese werden nur mit `handleOpenFile()` gemeldet. Das Argument-Array der `main()`-Methode ist dagegen immer leer!

```
public void handleQuit() {
    new Thread() {
        public void run() {
            // evtl. Abfrage, ob beendet werden soll
            beendeProgramm();
        }
    }.start();
    throw new IllegalStateException();
}

private void beendeProgramm() {
    // Aufräumarbeiten
    schliesseFenster();
    System.exit(0);
}
```

Jedes Programm muss irgendwann einmal beendet werden. Damit Sie mitbekommen, wenn der Anwender den »Beenden«-Menüpunkt im Programm-Menü anwählt, müssen Sie einen `MRJQuitHandler` registrieren. Wie bei

`handleAbout()` sollten Sie auch hier wieder den Thread-Workaround einsetzen. Damit das System weiß, dass Sie in einem separaten Thread entscheiden, ob das Programm wirklich beendet wird, werfen Sie eine `IllegalStateException` – Java 1.3 würde die Anwendung sonst nach dem Verlassen von `handleQuit()` beenden. Java 1.4 verhält sich hier etwas anders und beendet bei installiertem Handler das Programm niemals von sich aus. Wenn Sie das Programm direkt in `handleQuit()` ohne weiteren Thread beenden, ist die `IllegalStateException` unnötig.

Falls Sie `handleQuit()` in einer Swing-Applikation einsetzen, können Sie anstelle des Threads auch die Methode `invokeLater()` aus der Klasse `javax.swing.SwingUtilities` verwenden. Der Vorteil dabei ist, dass Oberflächenelemente – hier eine `JOptionPane` – dann automatisch vom korrekten Swing-Thread verarbeitet werden:

```
// alternativer Quit-Handler für Swing-Applikationen
public void handleQuit() {
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            int erg = JOptionPane.showConfirmDialog(
                null,
                "Wollen Sie das Programm beenden?",
                "Beenden",
                JOptionPane.YES_NO_OPTION,
                JOptionPane.QUESTION_MESSAGE
            );
            if (erg == JOptionPane.YES_OPTION) {
                beendeProgramm();
            }
        }
    });
    throw new IllegalStateException();
}

public static void main(String args[]) {
    Frame f = new AWTTest();
    f.setVisible(true);
}
}
```

Ganz am Schluss wird in der `main()`-Methode das Fenster-Objekt erzeugt und sichtbar gemacht. Oft finden Sie die beiden Zeilen auch zusammengefasst in

einer Anweisung `new AWTTest().setVisible(true);` oder noch kürzer `new AWTTest().show();`. Wenn Sie das Programm nun starten, sehen Sie die Abbildung 3.1, die bereits zu Anfang des Abschnitts vorgestellt wurde. Wenn Sie bisher auf anderen Systemen programmiert haben, beachten Sie bitte, dass sich die Menüzeile oben am Bildschirmrand befindet und nicht innerhalb des Fensters! Da AWT-Komponenten direkt vom System und nicht von der Java-Bibliothek gezeichnet werden, ist es tatsächlich Sache des Systems, wie und wo es die Komponenten darstellt – und ganz nebenbei erhält Ihre Java-Anwendung so ohne weiteres Zutun ein Mac-typisches Aussehen.

3.1.2 Fensterhintergrund

Der in Abbildung 3.1 gezeigte gestreifte Aqua-Hintergrund erscheint nur, wenn Sie das Programm mit Java 1.4 starten – bei Java 1.3 ist der Fensterhintergrund weiß. Auch mit Java 1.4 können Sie natürlich einen weißen Hintergrund einstellen, indem Sie `setBackground(Color.WHITE)` im `Frame`-Konstruktor aufrufen. Das Aqua-Aussehen erhalten Sie dann wieder mit `setBackground(null)`. Die Hintergrundfarbe kann auch ganz oder teilweise transparent sein. Mit

```
this.setBackground( new Color( 1.0F, 1.0F, 1.0F, 0.5F ) );
```

wird als letzter Parameter ein Alpha-Wert von 0.5 gesetzt, d.h., die Farbe ist nur zu 50% deckend (siehe Abbildung 3.3). Mit einem Alpha-Wert 0.0 bekommen Sie einen ganz unsichtbaren Hintergrund – die Komponenten »schweben« dann über anderen Fenstern und Applikationen. Wenn Sie solche Spielereien einsetzen, testen Sie diese aber gut auf allen Zielsystemen, denn dieses Verhalten ist nicht systemübergreifend garantiert.²

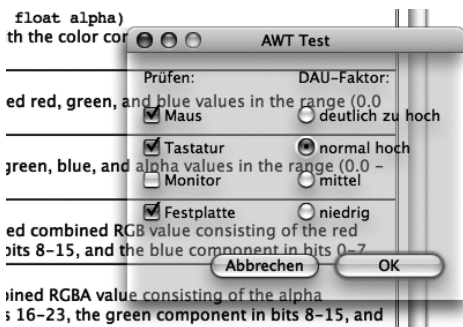


Abbildung 3.3 Transparenter Fensterhintergrund

2 Um den `Frame`-Rahmen ganz verschwinden zu lassen, müssen Sie `setUndecorated(true)` verwenden. Bei einem `JFrame` sollten Sie zusätzlich `getRootPane().setWindowDecorationStyle(JRootPane.PLAIN_DIALOG)` aufrufen.

3.1.3 Kontext-Popup-Menüs

Kontext-Popup-Menüs, die der Anwender mit einem Rechtsklick (bzw. mit `Ctrl`+Klick) auf ein Dialogelement aufrufen kann, sind aus modernen Anwendungen nicht mehr wegzudenken. Das Beispielprogramm besitzt ein solches Menü hinter dem Text »DAU-Faktor« (siehe Abbildung 3.4). Der folgende Quelltext stammt aus der Methode `erzeugeDialog()` in der Klasse `AWTTest`:

```
final PopupMenu popupFaktor = new PopupMenu();

popupFaktor.add( "Information" );
popupFaktor.addSeparator();
popupFaktor.add( "BoFH benachrichtigen" );
popupFaktor.add( "DAU-Alarm" );

labelFaktor.add( popupFaktor );
```

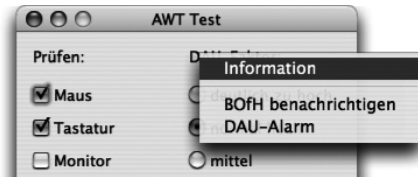


Abbildung 3.4 Kontext-Popup-Menü

Das Zusammenbauen eines Kontext-Menüs ist relativ geradlinig. Zunächst erzeugen Sie ein `PopupMenu`-Objekt, zu dem Sie die gewünschten Einträge (und Trennlinien) hinzufügen. Wichtig ist dann, dass Sie das Menü einer Komponente des Dialogs zuordnen, was hier mit `labelFaktor.add()` geschieht.

```
labelFaktor.addMouseListener( new MouseAdapter() {
    public void mousePressed(MouseEvent e) {
        zeigePopup(e);
    }

    public void mouseReleased(MouseEvent e) {
        zeigePopup(e);
    }

    private void zeigePopup(MouseEvent e) {
        if (e.isPopupTrigger()) {
            popupFaktor.show( labelFaktor, 10, 10 );
        }
    }
}
```

```
    }  
    });
```

Damit Sie das Anklicken der `Label`-Komponente mitbekommen, registrieren Sie einen `MouseListener` daran. Interessant sind dabei die Methoden `mousePressed()` und `mouseReleased()`, die beide versuchen, das Kontext-Popup-Menü anzuzeigen. Je nach Betriebssystem wird der Kontext-Klick nämlich beim Drücken (Mac) oder beim Loslassen (Windows) der Maustaste gemeldet. Ob wirklich ein Kontext-Klick vorliegt, können Sie nur mit der Methode `isPopupTrigger()` aus dem übergebenen `MouseEvent`-Objekt feststellen – fragen Sie niemals irgendwelche Umschalttasten oder rechten Maustasten direkt ab! Die Anzeige des Menüs erfolgt dann mit der Methode `show()`, der Sie eine Bezugskomponente und eine relative Position übergeben müssen – hier wird die linke obere Ecke des Menüs je zehn Pixel rechts und unterhalb der `labelFaktor`-Komponente dargestellt.

Apples Gestaltungsrichtlinien zu Kontext-Popup-Menüs sagen aus, dass Sie in diesen Menüs nichts anbieten sollten, was nicht auch über einen anderen Weg – beispielsweise die Menüzelle – erreicht werden kann. Das Kontext-Menü stellt nur eine *zusätzliche* Möglichkeit für den geübten Anwender dar, gewisse Funktionen schneller aufzurufen.

Wenn Sie bereits etwas intensiver mit dem AWT programmieren und die Methode `processMouseEvent()` überschreiben, achten Sie darauf, dass Sie die überschriebene Methode gleich zu Anfang Ihrer Methode aufrufen, weil es ansonsten zu Problemen mit der Kontext-Klick-Erkennung kommen kann.

3.1.4 Dateiauswahl-Dialoge

Um Anwender Dateien zum Laden oder Sichern auswählen zu lassen, stellt das AWT einen Dateiauswahl-Dialog mit der Klasse `java.awt.FileDialog` zur Verfügung. Wie beim AWT üblich, wird auch diese Komponente direkt vom System dargestellt und ist somit der bekannte Mac OS X-Dateiauswahl-Dialog (siehe Abbildung 3.5).

Der Dateiauswahl-Dialog lässt sich mit folgenden Tastenkombinationen schnell bedienen: `⌘+⌘+H` zeigt das Benutzerverzeichnis (»Home«) an, `⌘+D` den Schreibtisch (»Desktop«). Mit `⌘+⌘+G` (»Goto«) rufen Sie eine Textzeile auf, in der Sie das gewünschte Verzeichnis eintippen können.



Abbildung 3.5 Dateiauswahl-Dialog

```
private void oeffneDokument() {
    System.setProperty(
        "com.apple.macos.use-file-dialog-packages",
        "true" );
    FileDialog fd = new FileDialog( this,
                                    "Dokument öffnen",
                                    FileDialog.LOAD );
    fd.setDirectory( "~/Documents" );
    fd.setVisible(true);

    String pfad = fd.getDirectory();
    String datei = fd.getFile();
    if ((pfad != null) && (datei != null)) {
        // laden...
    }
}
```

Um einen Dateiauswahl-Dialog anzuzeigen, erzeugen Sie zunächst ein `FileDialog`-Objekt, dem Sie als Parameter den zugehörigen `Frame`, einen Titel und eine Konstante übergeben, ob der Dialog zum Laden (`FileDialog.LOAD`) oder zum Speichern (`FileDialog.SAVE`) verwendet wird. `setVisible()` macht den Dialog sichtbar und kehrt erst dann wieder zur Aufrufstelle zurück, wenn der Anwender eine Auswahl getroffen hat – Sie können dann die ausgewählte Datei und das zugehörige Verzeichnis abfragen. Wenn der Benutzer den Dialog abgebrochen hat, liefert `getFile()` den Wert `null` zurück.

Mit `setDirectory()` können Sie das Verzeichnis festlegen, das zu Anfang im Dialog angezeigt wird. Dies funktioniert unter MacOS X allerdings nur, wenn die Anwendung mit Java 1.4 läuft – unter Java 1.3 merkt sich der Dialog immer den zuletzt eingestellten Pfad. `setFile()` hat bei MacOS X überhaupt keine Auswirkung, da der Dateiauswahl-Dialog kein Textfeld für den Dateinamen besitzt.

Mac OS X-Programme haben eine Eigenheit: Es handelt sich dabei oft nicht nur um eine einzelne Programmdatei, sondern um eine ganze Verzeichnishierarchie, ein Programmpaket (»Bundle«). Damit der Anwender anstatt der Verzeichnisse nur eine Programmdatei sieht, setzen Sie vor dem Aufruf des Dialogs einfach die System-Property `com.apple.macos.use-file-dialog-packages` auf »true«.

Ein weiteres Problem entsteht, wenn der Dialog zur Auswahl eines Verzeichnisses verwendet werden soll – dies ist bei Suns Implementierung schlicht nicht vorgesehen (und Swings `JFileChooser`, der dies beherrscht, passt nicht zum Aqua-Aussehen, wie Sie noch sehen werden). Speziell unter MacOS X können Sie ein solches Verhalten bis Java 1.3 dadurch aktivieren, dass Sie als Dialog-Modus anstelle von `LOAD` oder `SAVE` den Wert `3` übergeben. Ab Java 1.4 steuern Sie dies mit der System-Property `apple.awt.fileDialogForDirectories`, so dass sich folgender Code für beide Java-Versionen anbietet:

```
System.setProperty( "apple.awt.fileDialogForDirectories", "true" );
FileDialog fd = new FileDialog( frame, "", FileDialog.LOAD );
try {
    fd.setMode( 3 );
}
catch (IllegalArgumentException e) { }
// ...
```

Listing 3.1 Dateialog zur Auswahl von Verzeichnissen

Die Klasse `FileDialog` bietet mit der Methode `setFilenameFilter()` die Möglichkeit, im Dateiauswahl-Dialog nur Dateien anzuzeigen, die bestimmten Kriterien eines `java.io.FilenameFilter` entsprechen. Während dies mit MacOS X problemlos funktioniert, ist Suns Implementierung für Windows fehlerhaft. Bei einem portablen Java-Programm können Sie daher diese Funktionalität leider nicht nutzen.

3.1.5 Portables Einbinden der Java 1.3-Apple-Erweiterungen

Wenn Sie eine an MacOS X angepasste Java-Anwendung auf anderen Systemen laufen lassen wollen, haben Sie ein Problem: Die Erweiterungsklassen aus dem Paket `com.apple.mrj` gibt es dort nicht. Es reicht auch nicht aus, die Apple-spezifischen Aufrufe mit `if (Sys.isMacOSX()) { ... }` zu kapseln, denn sobald Sie eine Klasse wie `MRJApplicationUtils` im Quelltext direkt ansprechen, versucht der Klassenlader zur Laufzeit auch, diese Klasse in den Speicher zu laden – und bricht mit einer Fehlermeldung ab, wenn er sie nicht findet.

Eine einfache Lösung besteht darin, so genannte »Stub«-Klassen (Stummel) mit Ihrem Programm auszuliefern. Die Stubs enthalten keinen sinnvollen Code, sondern bilden nur die Schnittstelle einer Klasse ab, damit diese vom Klassenlader gefahrlos geladen werden kann. Für die MRJ-Erweiterungen bietet Apple auf der Seite <http://developer.apple.com/samplecode/MRJToolkitStubsOld/MRJToolkitStubsOld.html> ein Archiv an, das alle notwendigen Stubs enthält und das Sie mit Ihrer Applikation ausliefern dürfen. Dieses Archiv müssen Sie auch zum Compiler-Klassenpfad hinzufügen, wenn Sie an MacOS X angepasste Java-Programme auf einem fremden System übersetzen wollen.

Eine technisch bessere Lösung, die auf die Stub-Klassen verzichtet, besteht darin, die speziellen Apple-Erweiterungen nicht direkt mit ihrem Klassennamen, sondern indirekt über »Java Reflection« anzusprechen. Damit Sie sich um die Feinheiten nicht kümmern müssen, greifen Sie auf das »Sys«-Projekt zurück, das mit dem Paket `com.muchsoft.util.mac` eine einfache Möglichkeit zur portablen Integration bietet. Die Hauptklasse `AWTTest` muss dafür nur minimal abgeändert werden:

```
//CD/examples/ch03/AWTPortabel/AWTPortabel.java
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import com.muchsoft.util.Sys;
import com.muchsoft.util.mac.*;

public class AWTPortabel extends Frame implements Java13Handler {

    public AWTPortabel() {
        this.erzeugeDialog();
        this.erzeugeMenue();
    }
}
```

```

        this.setResizable( false );
        this.setLocation( 20, 40 );
        this.pack();

        this.addWindowListener( new FensterSchliesser() );

        Javal3Adapter.registerJaval3Handler( this );
    }
    //...
}

```

Listing 3.2 Portables integrieren der Apple-Routinen für Java 1.3

Die Klasse implementiert nun nicht sehr viele einzelne Schnittstellen, sondern nur eine einzige: `com.muchsoft.util.mac.Javal3Handler`. Dieses Interface enthält die Methodensignaturen aller MRJ-Schnittstellen, ohne direkt auf die ursprünglichen Deklarationen zuzugreifen. Und im Konstruktor wird nun – ohne besondere System-Abfrage – einmal `registerJaval3Handler()` anstelle der zahlreichen `MRJApplicationUtils.registerXXXHandler()` aufgerufen. Das waren auch schon alle Änderungen! Wie funktioniert dies im Detail?

```

//CD/utility/sys/source/Javal3Adapter.java
package com.muchsoft.util.mac;
import java.io.File;
import com.muchsoft.util.Sys;

public class Javal3Adapter implements Javal3Handler {

    public void handleAbout() { }
    public void handleOpenApplication() { }
    public void handleOpenFile(File file) { }
    public void handlePrefs() { }
    public void handlePrintFile(File file) { }
    public void handleQuit() { }

    public static void registerJaval3Handler(Javal3Handler handler) {
        if (Sys.isMacOSX()) {
            try {
                Class integration =
                    Class.forName(
                        "com.muchsoft.util.mac.Javal3Integration"
                    );
            }
        }
    }
}

```



```

        java.lang.reflect.Constructor constr =
            integration.getConstructor(
                new Class[] { Javal3Handler.class }
            );

        constr.newInstance( new Object[] { handler } );
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
}
}

```

Listing 3.3 Java Reflection für portable Programmierung

Die Klassenmethode `registerJaval3Handler()` testet zunächst auf Mac OS X – aber das hatte das erste Beispielprogramm auch gemacht. Portabel wird diese Methode dadurch, dass sie auf die Implementierungsklasse `com.muchsoft.util.mac.Javal3Integration`, die die speziellen Apple-Aufrufe durchführt, nicht direkt zugreift, sondern diesen Namen bei `Class.forName()` als Zeichenkette angibt – der Klassenlader lädt die Klasse dadurch nur noch dann, wenn er mit `forName()` dazu aufgefordert wird.

Danach wird dann dynamisch ein passender Konstruktor gesucht und eine neue Instanz (ein neues Exemplar) der Klasse erzeugt. Ausnahmefehler sollte es in diesem Code nicht geben, denn es wurde ja sichergestellt, dass der Code nur unter Mac OS X durchlaufen wird.

Bei Interesse können Sie sich auch den Quelltext der Implementierungsklasse `Javal3Integration` ansehen, den Sie auf der Buch-CD im Verzeichnis `/utility/sys/source/` finden.

Was fehlt dem Beispielprogramm noch? Zum einen ist die Anwendung noch nicht zur Verwaltung mehrerer Dokumente und damit mehrerer Fenster geeignet. Dann sollten auch andere Lokalisierungen als Deutsch vorliegen (zumindest noch eine englische) – wobei keine Änderungen am Quelltext notwendig sein sollten, nur weil sich eine Beschriftung geändert hat. Und schließlich merkt man dem Programm an, dass es keine native Mac OS X-Anwendung ist – spätestens, wenn Sie das Fenster schließen. Dann nämlich ändert sich die Menüzeile, die dann nur noch das Apfel- und das Programm-Menü umfasst. Typischerweise sollte aber immer die komplette Menüzeile sichtbar sein,

solange das Programm läuft; nicht verfügbare Menüpunkte werden dann nicht anwählbar (grau) dargestellt. All dies kann bei AWT-Applikationen realisiert werden, es ist aber im folgenden Abschnitt für Swing-Anwendungen dokumentiert.

3.2 Swing und die Java 1.4-Apple-Erweiterungen

Während beim AWT die Komponenten vom System dargestellt werden, ist bei der Swing-Bibliothek Java für das Zeichnen der Elemente verantwortlich. Dadurch kann zum einen sichergestellt werden, dass die Oberfläche auf jedem System exakt gleich aussieht – dies ist wichtig, wenn ein Entwickler pixelgenaue Masken und Formulare programmiert und sich nicht um Auflösungsunabhängigkeit kümmert. Zum anderen ist aber auch das genaue Gegenteil möglich: Das Aussehen der Benutzungsoberfläche kann in der Java-Anwendung umgeschaltet werden. Dazu bringt Swing einige austauschbare Oberflächendesigns mit (»Pluggable Look & Feel«, kurz PLAF; oft auch nur mit LAF oder L&F abgekürzt), und weitere lassen sich einfach als JAR-Archiv installieren.

Ein Beispiel für die Mächtigkeit von Swing finden Sie auf Ihrer Festplatte im Verzeichnis `/Developer/Examples/Java/JFC/SwingSet2/` – starten Sie das dortige JAR-Archiv mit einem Doppelklick. Über das »Look & Feel«-Menü können Sie das komplette Aussehen des Programms zur Laufzeit ändern. Das plattformübergreifende Aussehen ist das »Java-Look & Feel« (auch »Metal« genannt), das in Abbildung 3.6 dargestellt ist.

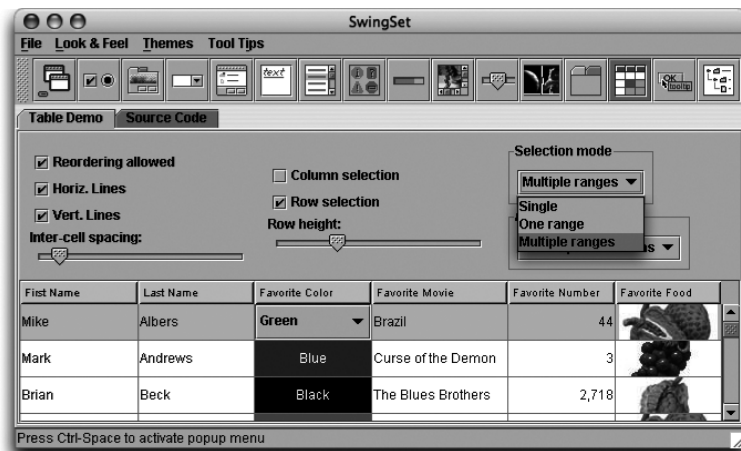


Abbildung 3.6 »Metal«-Look & Feel

Das »Macintosh-Look & Feel« (auch »Aqua-Look & Feel« genannt) steht nur auf Apple-Systemen zur Verfügung – dort sollten Sie dieses Aussehen nach Möglichkeit aber auch nutzen (siehe Abbildung 3.7). Die Komponenten belegen dabei etwas mehr Platz als beim Java-Aussehen, daher sollten Sie Ihre Applikation tatsächlich unter Mac OS X getestet haben, bevor Sie die Konfiguration entsprechend umstellen. Beachten Sie, dass sich die Menüleiste nun oben am Bildschirmrand befindet – so, wie Sie es von Mac OS X-Programmen und von Java-AWT-Anwendungen kennen. Bei Swing hängt die Position der Menüleiste vom Look & Feel ab.

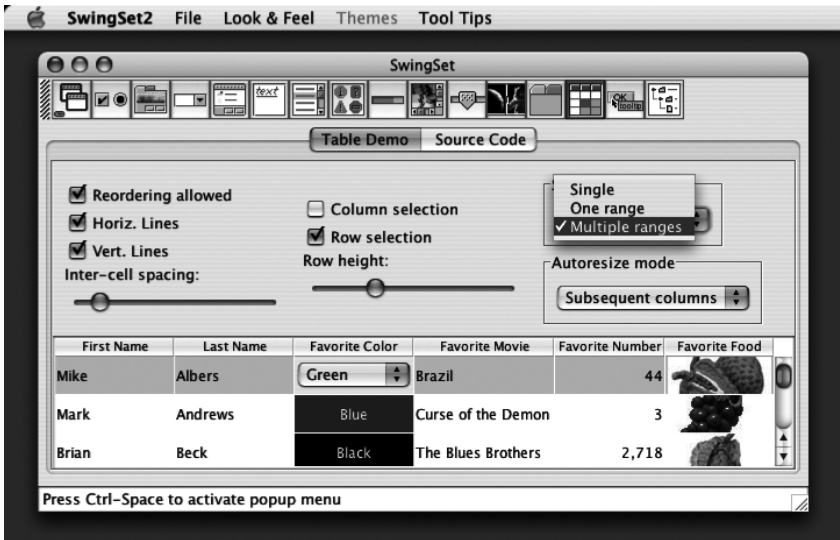


Abbildung 3.7 Aqua-Look & Feel

Falls Sie sich schon mit Swing auf anderen Systemen auskennen, sollten Sie bei künftigen Projekten bedenken, dass MDI-Anwendungen (»Multiple Document Interface« – mehrere Dokumentfenster innerhalb eines übergeordneten Fensters) und damit insbesondere die Klassen `javax.swing.JInternalFrame` und `javax.swing.JDesktopPane` nicht zur Apple-Philosophie passen. Beim Mac ist pro Dokument ein komplett eigenständiges Fenster vorgesehen.

Wenn Sie dies nicht von Anfang an in Ihrer Applikation berücksichtigt haben, ist eine Umstellung auf einer Mehr-Fenster-Anwendung – eventuell sogar nur speziell unter MacOS X – natürlich schwierig umzusetzen. Bei bestehenden MDI-Anwendungen können Sie sich daher der Einfachheit halber zunächst auf die voll funktionstüchtige Apple-Implementierung der internen Fenster verlassen – wie das `SwingSet2` beim »Internal Frames Demo« sehr schön zeigt.

3.2.1 Konfiguration des Aussehens (Look & Feel)

Welches Aussehen standardmäßig verwendet wird, ist bei eigentlich allen Java-Implementierungen in einer Datei `swing.properties` festgelegt. Bei MacOS X finden Sie diese für Java 1.4 im Verzeichnis `/Library/Java/Home/lib/` mit folgendem Inhalt:

```
swing.defaultlaf=apple.laf.AquaLookAndFeel
```

Für Java 1.3 liegt diese Datei im Verzeichnis `/System/Library/Frameworks/JavaVM.framework/Versions/1.3.1/Home/lib/` und hat eine andere Vorgabe:

```
swing.defaultlaf=com.apple.mrj.swing.MacLookAndFeel
```

Beide Versionen nutzen also normalerweise Apples Aqua-Look & Feel, auch wenn sich die Klasse für die Implementierung geändert hat. Die Einträge in diesen Dateien sollten Sie nur zur Information nutzen und niemals ungefragt von einem Programm abändern lassen – die Vorgabe ist Sache des jeweiligen Rechner-Administrators.

Ein nützliches Werkzeug zum Verwalten der `swing.properties`-Datei ist das »TKPLAFUtility«, das Sie von der Seite <http://www.moniundthomaskueneth.de/tkplafutility/> herunterladen können.

Um eine Anwendung mit einem bestimmten Aussehen zu starten, können Sie die Property `swing.defaultlaf` beim Aufruf in der Kommandozeile angeben:

```
java -Dswing.defaultlaf=javax.swing.plaf.metal.MetalLookAndFeel
    MeineApp
```

Diese Angabe hat dann Vorrang vor dem in der Datei `swing.properties` festgelegten Standardaussehen. Das hier verwendete `MetalLookAndFeel` ist das Java-Look & Feel, das Sie weiter oben schon gesehen haben. So können Sie zahlreiche weitere PLAF-Klassen aus dem Internet laden und verwenden – das jeweilige JAR-Archiv muss sich dafür nur auf dem Klassenpfad befinden (z.B. in einem der Erweiterungsverzeichnisse). Allerdings sind nicht alle PLAFs auf jedem System nutzbar – immer dann nicht, wenn sie auf native Systemfunktionen zurückgreifen oder wenn die Verwendung auf bestimmten Systemen vom PLAF-Code verhindert wird.

Das Aussehen kann auch im Programmcode eingestellt werden, was dann Vorrang vor allen anderen Konfigurationsmöglichkeiten hat. Das folgende Listing zeigt, wie unter Mac OS X das Aqua-Aussehen und unter Windows das Windows-Aussehen aktiviert wird:

```
public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel(
            UIManager.getSystemLookAndFeelClassName()
        );
    }
    catch (UnsupportedLookAndFeelException e) {
        // L&F wird von diesem System nicht unterstützt
    }
    catch (Exception e) {
        // L&F konnte nicht geladen werden
    }

    // erst danach UI-Elemente (Fenster usw.) erzeugen
    new JFrame().setVisible(true);
}
```

Listing 3.4 Setzen des Look & Feel im Programmcode

Die Klasse `javax.swing.UIManager` besitzt einige statische Methoden, mit denen das Look & Feel abgefragt und gesetzt werden kann. Wichtig dabei ist, dass Sie das Aussehen so früh wie möglich einstellen, auf jeden Fall aber bevor

irgendein GUI-Element erzeugt wird. Am besten führen Sie die Anweisungen – wie hier gezeigt – zu Beginn der `main()`-Methode aus.

Mit `getSystemLookAndFeelClassName()` fragen Sie den Namen des Systemaussehens ab – bei MacOS X und Java 1.4 beispielsweise `apple.laf.AquaLookAndFeel`, was ja auch in der Datei `swing.properties` eingetragen ist – und setzen diesen Namen dann mit `setLookAndFeel()`.

Wenn Sie das Aussehen nachträglich zur Laufzeit ändern wollen, also nachdem bereits Oberflächenelemente erzeugt und angezeigt wurden, müssen Sie nach dem Setzen des neuen Look & Feel alle Fenster mit `SwingUtilities.updateComponentTreeUI()` aktualisieren und die Komponenten anschließend neu anordnen lassen:

```
UIManager.setLookAndFeel(  
    UIManager.getCrossPlatformLookAndFeelClassName()  
);  
  
SwingUtilities.updateComponentTreeUI( frame );  
frame.pack();
```

Listing 3.5 Nachträgliches Ändern des Look & Feel zur Laufzeit

Die hier verwendete Methode `getCrossPlatformLookAndFeelClassName()` liefert den Namen den plattformübergreifenden Aussehens zurück, also das Java-Look & Feel (»Metal«). Weitere Infos zum Setzen des Aussehens erhalten Sie bei Sun auf der Seite <http://java.sun.com/docs/books/tutorial/uiswing/misc/plaf.html>.

3.2.2 Ein Swing-Beispiel mit mehreren Dokument-Fenstern

Mit Java 1.4 haben sich nicht nur Apples interne PLAF-Klassen geändert, sondern auch die Apple-Erweiterungen zur besseren Integration eines Programms ins Betriebssystem. Im Gegensatz zu den Java 1.3-Erweiterungen handelt es sich nun um richtige Listener, die auch mehrfach registriert werden können. Die Klassen gehören nun zum Paket `com.apple.eawt`. Verwenden Sie bei neuen Projekten, die Java 1.4 voraussetzen können, nach Möglichkeit immer diese neuen Erweiterungen.

Im Folgenden wird als Beispielprogramm ein kleiner Minimal-Editor entworfen. Der Editor kann zwar keine Dokumente laden und speichern, aber Sie sehen daran recht gut, wie mehrere Fenster verwaltet werden, wo die Unterschiede zwischen der AWT- und der Swing-Programmierung liegen und natürlich wie die Oberfläche mit den neuen Erweiterungen ins System integriert

wird. In Xcode können Sie dafür den Projekttyp »Java Swing Application« verwenden, der komplett konfiguriert ist und alle wesentlichen Dateien bereits mitbringt. Wie beim AWT-Beispiel sehen Sie zunächst wieder den direkten Zugriff auf die Apple-Erweiterungen, danach wird der Quelltext dann portabel gemacht.

Die vorgestellten Quelltexte sind speziell für die Beispielanwendung geschrieben, es handelt sich nicht um eine universell einsetzbare Rahmenanwendung – allerdings sollte es recht einfach sein, die Quelltexte an eigene Bedürfnisse anzupassen. Durch die spezielle Ausrichtung können die Quelltexte übersichtlicher gehalten werden und sind leichter zu verstehen. Aus demselben Grund wurde das Programm auch nicht immer optimiert bezüglich der Anzahl der erzeugten Objekte (beispielsweise bei den Listener-Objekten).

Wenn Sie einen universellen Ansatz suchen, der auch noch ältere Mac-Systeme unterstützt, bietet sich wieder der **MRJAdapter** an (<http://www.roydesign.net/mrjadapter/>).

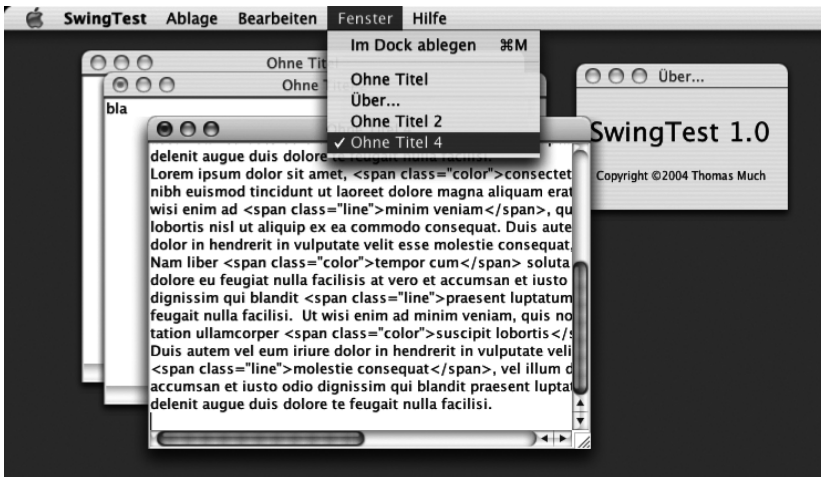


Abbildung 3.8 Eine Java-Swing-Anwendung, die zu Aqua passt

```
//CD/examples/ch03/SwingTest/BasisFenster.java
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
```

```

import com.muchsoft.util.Sys;

public class BasisFenster extends JFrame {
    protected JMenuItem miNeu, miOeffnen, miSchliessen /* ... */ ;
    private JMenu menuFenster;

    private static Vector fenster = new Vector();
    public static String INVISIBLE = "<invisible window ... >";

```

Los geht es mit einer Basisklasse, die Grundfunktionen für sämtliche Fenster des Programms bereitstellt. Auch wenn noch einige Hilfsklassen aus den AWT-Paketen verwendet werden, stammen die Oberflächenelemente nun aus dem Paket `javax.swing` – so wird hier beispielsweise ein `javax.swing.JFrame` anstelle eines `java.awt.Frame` eingesetzt (abgesehen vom Paket ist das »J« am Anfang eines Klassennamens der offensichtlichste Unterschied zwischen AWT- und Swing-Komponenten).

Diese Klasse kümmert sich auch um eine Besonderheit von Mac-Applikationen: Die Menüzeile, die am oberen Bildschirmrand dargestellt wird, bleibt immer gleich – egal, welches Fenster gerade das oberste ist. Menüeinträge, die nicht zum aktuellen Fenster passen, werden deaktiviert (»disabled«). Weil aber Swing prinzipiell vorsieht, dass jedes Fenster eine eigene Menüzeile hat, schaltet Apples Java-Implementierung die Menüzeile oben am Bildschirm je nach aktivem Fenster um. Daher erzeugt diese Klasse einfach eine komplette, immer gleiche Menüzeile pro Fenster. Jedes Fenster kümmert sich dann selbst darum, die unpassenden Menüpunkte zu deaktivieren, wofür die `JMenuItem`-Objektvariablen genutzt werden.

Außerdem wird von dieser Klasse das Fenster-Menü verwaltet (siehe Abbildung 3.8). In diesem Menü tauchen die Titel aller offenen Fenster (in der Reihenfolge des Öffnens) auf, das oberste Fenster hat ein Häkchen vor seinem Namen. Durch Anwahl eines Fenster-Namens wird dieses Fenster nach vorne geholt. Dazu trägt sich jedes `BasisFenster` in das statische `java.util.Vector`-Objekt `fenster` ein, und jedes Fenster-Objekt erzeugt sich aus dieser Liste ein eigenes, komplettes Fenster-Menü `menuFenster`. Beim Schließen eines Fensters entfernt sich dieses wieder selbst aus der Liste.

Die Konstante `INVISIBLE` wird für eine weitere Mac-Besonderheit benötigt. Wenn eine Applikation läuft, besitzt sie auch dann eine Menüzeile, wenn gar kein Fenster offen ist! Mit dem Menüpunkt **Ablage · Neu** kann der Benutzer dann ein neues Dokument-Fenster öffnen. Wird dem `BasisFenster`-Konstruktor diese Konstante als Titel übergeben, taucht das Fenster nicht im Fenster-Menü auf und wird unsichtbar außerhalb des Bildschirms erzeugt – eine

Menüzeile kann dieses unsichtbare Fenster aber dennoch besitzen. Mit diesem kleinen (und offiziell von Apple abgesegneten) Trick verhält sich die Java-Menüzeile genauso wie die einer nativen MacOS X-Applikation.

```
public BasisFenster(String titel) {
    super(titel);
    ResourceBundle resbundle = SwingTest.getResBundle();

    JMenuBar mbar = new JMenuBar();
    JMenu menuAblage =
        new JMenu( resbundle.getString("menuFile") );
    JMenu menuBearbeiten =
        new JMenu( resbundle.getString("menuEdit") );
    menuFenster =
        new JMenu( resbundle.getString("menuWindow") );

    JMenu menuHilfe =
        new JMenu( resbundle.getString("menuHelp") );

    mbar.add( menuAblage );
    mbar.add( menuBearbeiten );
    mbar.add( menuFenster );
    mbar.add( menuHilfe );

    this.setJMenuBar( mbar );
}
```

Der Konstruktor erfragt von der Hauptklasse zunächst ein `java.util.ResourceBundle`. Resource-Bundles sind Textdateien mit Schlüssel-Wert-Paaren, die zur Lokalisierung eines Programms genutzt werden. Wie diese Textdateien aufgebaut sind und wie sie geladen werden, wird weiter unten bei der `main()`-Methode beschrieben. Die Verwendung der übersetzten Ressourcen ist denkbar einfach: Sie rufen auf dem `ResourceBundle`-Objekt die Methode `getString()` auf, der Sie den von Ihnen festgelegten Schlüssel übergeben, und erhalten die passende Zeichenkette zurück. Hier sehen Sie am Beispiel der `JMenu`-Objekte, wie diese ihren Titel ermitteln.



```
miSichern =
    new JMenuItem( resbundle.getString("menuFileSave") );
miSichernUnter =
    new JMenuItem( resbundle.getString("menuFileSaveAs") );
// ...
menuAblage.addSeparator();
```

```

menuAblage.add( miSichern );
menuAblage.add( miSichernUnter );
// ...

final int mask =
    Toolkit.getDefaultToolkit(). getMenuShortcutKeyMask();
miSichern.setAccelerator(
    KeyStroke.getKeyStroke( KeyEvent.VK_S, mask ));
miSichernUnter.setAccelerator(
    KeyStroke.getKeyStroke( KeyEvent.VK_S,
                            mask | InputEvent.SHIFT_MASK ));
// ...

```

Auch die einzelnen JMenuItem-Menüeinträge erfragen ihre Beschriftung aus dem Resource-Bundle. Einen wichtigen Unterschied zwischen dem AWT und Swing gibt es dann beim Setzen der Tastaturkürzel, wofür nun die Methode `setAccelerator()` verwendet wird. Die übergebenen `javax.swing.KeyStroke`-Objekte erhalten dabei als Parameter nicht nur die `VK_XXX`-Konstante für den Buchstaben des Kürzels, sondern auch eine Bitmaske für die zu drückenden Umschalttasten. Ermitteln Sie diese Bitmaske immer mit `getMenuShortcutKeyMask()`! Auch wenn man leicht feststellen kann, dass Windows hier `CTRL_MASK` und Mac OS X `META_MASK` liefert, ist die `Toolkit`-Methode der einzig portable Weg. Wenn das Tastenkürzel  oder  enthalten soll, »verodern« Sie die Bitmaske mit `InputEvent.SHIFT_MASK` bzw. `ALT_MASK`.

```

if (!Sys.isMacOSX()) {
    JMenuItem miExit =
        new JMenuItem( resbundle.getString("menuFileExit") );
    miExit.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // Methoden zum Sichern der Daten aufrufen
            System.exit(0);
        }
    });
    menuAblage.addSeparator();
    menuAblage.add( miExit );

    JMenuItem miAbout =
        new JMenuItem( resbundle.getString("menuHelpAbout") );
    miAbout.addActionListener( new ActionListener() {

```

```

        public void actionPerformed(ActionEvent e) {
            ProgrammInfo.zeigeInfo();
        }
    });
    menuHilfe.addSeparator();
    menuHilfe.add( miAbout );
}

```

Nun folgen die Anpassungen für andere Systeme, die kein automatisch verwaltetes Programm-Menü mit einem »Über...«- und einem »Beenden«-Menüeintrag besitzen. Am Ende des Ablage-Menüs wird ein Menüpunkt zum Beenden des Programms hinzugefügt, am Ende des Hilfe-Menüs ein Eintrag für die Programm-Informationen. Bei MacOS X wird diese Funktionalität von den Apple-Erweiterungen geliefert, die in der weiter unten beschriebenen Klasse `SwingTest` genutzt werden.

```

miNeu.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        new DokumentFenster().setVisible( true );
    }
});

miSchliessen.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        schliesseFenster();
    }
});

```

Die Menüeinträge werden mit Listener-Objekten verknüpft – exakt so, wie Sie dies schon beim AWT gesehen haben. Jedes Fenster erzeugt mit dem Menüpunkt **Ablage • Neu** ein neues Dokument-Fenster (dessen Beschreibung folgt unten) und lässt sich mit dem entsprechenden Eintrag im Ablage-Menü schließen.

```

miOeffnen.setEnabled( false );
miSichernUnter.setEnabled( false );
// ...

if (titel != INVISIBLE) {
    fenster.add( this );
}

```

```

        this.updateAllWindows();
    }

```

Am Ende des Konstruktors werden alle normalerweise nicht verfügbaren Menüeinträge deaktiviert – in diesem Fall sind dies alle Punkte, die bei der Beispielanwendung nicht ausprogrammiert wurden. Danach trägt sich das `BasisFenster` selber in die Liste aller Fenster ein und informiert alle bereits offenen Fenster, damit diese ihre Fenster-Menüs aktualisieren können.

```

protected void schliesseFenster() {
    fenster.remove( this );
    this.updateAllWindows();
    this.setVisible(false);
    this.dispose();
}

```

Beim Schließen eines Fensters entfernt sich dieses aus der Liste aller Fenster und informiert wieder alle noch offenen Fenster, damit sie ihre Fenster-Menüs anpassen können.

```

protected void updateAllWindows() {
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            Enumeration e = fenster.elements();
            while ( e.hasMoreElements() ) {
                BasisFenster bf = (BasisFenster)e.nextElement();
                bf.updateFensterMenu();
            }
            ((BasisFenster)SwingTest.getApp()).updateFensterMenu();
        }
    });
}

```

Um alle Fenster zu benachrichtigen, dass sich die Liste der offenen Fenster geändert hat, durchläuft die Methode `updateAllWindows()` einfach alle Objekte im `fenster`-Vektor und ruft in jedem Fenster die Methode `updateFensterMenu()` auf. Danach wird das unsichtbare Fenster, das zur Anzeige der Menüzelle ohne sichtbare Fenster genutzt wird, mit `SwingTest.getApp()` abgefragt und ebenfalls zum Aktualisieren des Fenster-Menüs aufgefordert.

Das Ganze geschieht verzögert mit `SwingUtilities.invokeLater()`, wodurch die Bearbeitung im korrekten Ereignis-Verarbeitungs-Thread erfolgt –

ansonsten könnten nämlich die Titel neu erzeugter Fenster unter Umständen noch nicht abgefragt werden.

```
private void updateFensterMenu() {
    int mcc = menuFenster.getMenuComponentCount();
    for (int i = mcc - 1; i > 0 ; i--) {
        menuFenster.remove( i );
    }

    int fs = fenster.size();
    if ( fs > 0 ) {
        menuFenster.addSeparator();
        for (int i = 0; i < fs; i++) {
            BasisFenster bf = (BasisFenster)fenster.elementAt(i);
            JMenuItem jmi;
            if (this == bf) {
                jmi =
                    new JCheckBoxMenuItem( this.getTitle(), true);
            }
            else {
                jmi = new JMenuItem( bf.getTitle() );
            }
            menuFenster.add( jmi );
            jmi.addActionListener( new ToFront( i ) );
        }
    }
}
```

Mit dieser Methode `updateFensterMenu()` aktualisiert jedes Fenster sein eigenes Fenster-Menü. Zunächst werden alle Fenster-Namen (inklusive Trennlinie) aus dem Menü entfernt. Danach werden – wenn überhaupt noch Fenster offen sind – wieder die Trennlinie und anschließend alle aktuellen Fenster-Namen in das Menü eingefügt.

Wenn die Methode in der Schleife feststellt, dass sie gerade den eigenen Fenster-Namen hinzufügt, wird ein `JCheckBoxMenuItem`-Eintrag verwendet, damit ein Häkchen vor dem Namen erscheint. Ansonsten wird ein ganz normales `JMenuItem`-Objekt verwendet. Der Eintrag wird dann ans Ende des Menüs angehängt und ein Listener-Objekt daran registriert, das von der inneren Klasse `ToFront` definiert wird:

```
private class ToFront implements ActionListener {
```

```

private int nr;

public ToFront(int nr) {
    this.nr = nr;
}

public void actionPerformed(ActionEvent e) {
    JMenuItem source = (JMenuItem)e.getSource();
    try {
        ((JCheckBoxMenuItem)source).setState( true );
    }
    catch (ClassCastException ex) { /* erwartete Ausnahme */ }

    BasisFenster bf = (BasisFenster)fenster.elementAt( nr );
    if (bf.getExtendedState() == Frame.ICONIFIED) {
        bf.setExtendedState( Frame.NORMAL );
    }
    bf.toFront();
}
}
}
}

```

Wenn der Benutzer einen Menüpunkt im Fenster-Menü auswählt, wird die Methode `actionPerformed()` des Listeners aufgerufen. Als Erstes wird darin aus dem übergebenen `ActionEvent`-Objekt die Ereignis-Quelle ermittelt, also das Menüeintrag-Objekt. Mit einer Typumwandlung – die bewusst schief gehen kann – wird dann versucht, das Häkchen vor dem Eintrag zu setzen. Dadurch wird sichergestellt, dass das Häkchen wieder gesetzt wird, wenn der Eintrag für das oberste (aktive) Fenster angewählt wird.

In jedem Fall wird das ausgewählte Fenster dann gegebenenfalls aus dem Dock hervorgeholt und anschließend mit `toFront()` nach vorne gebracht.

Nun folgt die Hauptklasse, die die Mac OS X-Integration mithilfe der Apple-Erweiterungen vornimmt:

```

//CD/examples/ch03/SwingTest/SwingTest.java
import java.awt.*;
import java.util.*;
import javax.swing.*;
import com.apple.eawt.*;
import com.muchsoft.util.Sys;

```

```
public class SwingTest extends BasisFenster
    implements ApplicationListener {
    private static SwingTest    app;
    private static ResourceBundle resbundle;
    private      Application    eawtApp;
```

Die Klasse `SwingTest` implementiert die Schnittstelle `com.apple.eawt.ApplicationListener`, die zahlreiche Methoden zur Systemintegration deklariert (bei den Java 1.3-Erweiterungen waren dies noch viele einzelne Schnittstellen). Die Registrierung des Listeners erfolgt im Konstruktor:

```
public SwingTest() {
    super( BasisFenster.INVISIBLE );

    miSchliessen.setEnabled( false );
    miSichern.setEnabled( false );
    miDock.setEnabled( false );

    eawtApp = new Application();
    eawtApp.addApplicationListener( this );
    eawtApp.setEnabledPreferencesMenu( true );
    eawtApp.setEnabledPreferencesMenu( false );

    this.setBackground( new Color( 1.0F, 1.0F, 1.0F, 0.0F ) );
    this.setUndecorated( true );
    this.setBounds( 30000, -30000, 0, 0 );
    this.pack();
    if (Sys.isMacOSX()) {
        this.setVisible( true );
    }
}
```

Der Konstruktor setzt zunächst die spezielle Titel-Konstante `INVISIBLE`, damit dieses Fenster zwar eine Menüleiste darstellt, aber selbst nicht sichtbar ist. Dazu muss das Fenster noch mit ein paar Kniffen manipuliert werden (hier am Ende des Konstruktors): Die Hintergrundfarbe muss einen Alpha-Wert von 0.0 haben, muss also vollständig transparent sein. Außerdem muss das Fenster durch `setUndecorated(true)` ohne Titelzeile und Fensterrahmen dargestellt werden. Schließlich wird das Fenster noch außerhalb des Bildschirms positioniert. Damit das nun unsichtbare Fenster trotzdem eine Menüleiste anzeigt,

müssen Sie das Fenster mit `setVisible()` »sichtbar« machen – dies geschieht allerdings nur unter Mac OS X, andere Systeme benötigen ja gar keine Menüzeile ohne sichtbares Fenster.

Dazwischen wird ein `com.apple.eawt.Application`-Objekt erzeugt, das für die Systemintegration verantwortlich ist. Mit `addApplicationListener()` wird die Verknüpfung zwischen dem `Application`-Objekt und dem `Swing-Test`-Objekt hergestellt.

Die beiden `setEnabledPreferencesMenu()`-Aufrufe sind hier eigentlich unnötig. Sie dienen nur dazu, den »Einstellungen...«-Menüeintrag im Programm-Menü einzublenden und sofort wieder zu deaktivieren. Dadurch sieht man immerhin, dass es einen solchen Menüpunkt geben könnte.

```
public void handleAbout(ApplicationEvent event) {
    ProgrammInfo.zeigeInfo();
    event.setHandled( true );
}

public void handleQuit(ApplicationEvent event) {
    event.setHandled( false );
    // Methoden zum Sichern der Daten aufrufen
    System.exit(0);
}

public void handlePreferences(ApplicationEvent event) { }
public void handleOpenApplication(ApplicationEvent event) { }
public void handleReOpenApplication(ApplicationEvent event) { }
public void handleOpenFile(ApplicationEvent event) { }
public void handlePrintFile(ApplicationEvent event) { }
```

Die Methoden der `ApplicationListener`-Schnittstelle sehen denen der `MRJ-Interfaces` recht ähnlich. Allerdings wird jeder Methode beim Aufruf ein `com.apple.eawt.ApplicationEvent`-Objekt übergeben, mit dem Sie – je nach Ereignis – beispielsweise einen Dateinamen ermitteln können.

Wichtig ist, dass Sie dem System mit `setHandled()` sagen, ob Sie das Ereignis verarbeitet haben. Falls Sie diesen Aufruf vergessen, erscheint z.B. nach `handleAbout()` zusätzlich zu Ihrem `ProgrammInfo`-Dialog immer auch noch ein vom System generierter `Info`-Dialog. Bei `handleQuit()` legen Sie mit `setHandled()` fest, ob das Programm automatisch beendet werden soll (`true`) oder ob Sie erst noch Daten speichern wollen und das Programm dann selbst beenden (`false`).

Während die Handler der alten Erweiterungen unter Java 1.4 teilweise nicht zuverlässig funktionieren, kann man sich auf die neuen Handler verlassen. `handleOpenApplication()` wird einmal beim Programmstart aufgerufen, `handleOpenFile()` jedes Mal, wenn der Anwender eine Datei auf das Programmsymbol zieht (auch beim Programmstart). `handleReOpenApplication()` wird ausgelöst, wenn der Benutzer auf das Dock-Symbol der laufenden Anwendung klickt – eine typische Reaktion darauf ist, ein neues Dokument-Fenster zu öffnen, wenn überhaupt keines mehr offen ist.

```

public static SwingTest getApp() {
    return app;
}

public static ResourceBundle getResBundle() {
    return resbundle;
}

public static void main(String[] args) {
    // Look&Feel setzen...
    resbundle =
        ResourceBundle.getBundle( "SwingTeststrings",
                                Locale.getDefault() );
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            app = new SwingTest();
            new DokumentFenster().setVisible( true );
        }
    });
}
}

```

In der `main()`-Methode wird zu Beginn das System-Look & Feel gesetzt, wie Sie es in Abschnitt 3.2.1 gesehen haben. Am Ende werden zwei Fenster erzeugt – ein unsichtbares `SwingTest`-Fenster und ein erstes `DokumentFenster`. Dies passiert mit `invokeLater()` wieder im Ereignis-Verarbeitungs-Thread, wo sämtliche Realisierungen von Oberflächenelementen stattfinden sollten.³

³ Sun hat dies erst ziemlich spät auf <http://java.sun.com/developer/JDCTechTips/2003/tt1208.html#1> dokumentiert. Mittlerweile geht man bei Sun sogar dazu über, das gesamte GUI innerhalb des Ereignis-Verarbeitungs-Threads zu erzeugen, siehe <http://java.sun.com/developer/JDCTechTips/2004/tt0518.html#1>.

Dazwischen wird das `ResourceBundle` mit den lokalisierten Zeichenketten für Menütitel, -einträge usw. geladen. An die Methode `getBundle()` übergeben Sie dafür einen Basis-Dateinamen, an den bei Bedarf Landeskennungen angehängt werden. Hier wird mit `Locale.getDefault()` die im System eingestellte Landeskenung abgefragt, beispielsweise »de_DE« (die ersten Buchstaben stehen für die Sprache, die letzten für das Land). In der Datei definieren Sie dann beliebige Schlüssel-Wert-Paare:

```
# CD/examples/ch03/SwingTest/SwingTeststrings_de.properties
appName=SwingTest
appVersion=1.0
dialogAboutTitle=Über...
dialogAboutCopyright=Copyright \u00a92004 Thomas Much
documentTitle=Ohne Titel
menuFile=Ablage
menuFileNew=Neu
menuFileOpen=Öffnen...
menuFileClose=Schließen
menuFileSave=Sichern
# ...
```

Listing 3.6 Lokalisierte Datei `SwingTeststrings_de.properties`

Beachten Sie, dass an den Basis-Dateinamen nur »de« (und nicht »de_DE«) angehängt wurde – hier interessiert nur die Sprache, nicht das Land, und Sie müssen die Landeskenung nur so genau angeben, wie Sie benötigen. Die Dateinamenerweiterung `.properties` ist zwingend vorgeschrieben.

Sie können beliebig viele weitere Dateien für andere Lokalisierungen anlegen, die dann – je nach Systemeinstellung – automatisch geladen werden. Auf jeden Fall sollten Sie aber die Standard-Lokalisierungsdatei ohne spezielle Landeskenung mitliefern (hier also `SwingTeststrings.properties`), die dann verwendet wird, wenn keine andere passende Lokalisierung gefunden wird. Dementsprechend enthält diese Datei üblicherweise englische Texte.

Wenn Sie in der Properties-Datei Umlaute verwenden wollen, müssen Sie unbedingt darauf achten, dass als Textkodierung »Western (ISO Latin 1)« eingestellt ist! In Xcode erreichen Sie dies im Info-Dialog unter »File Encoding«. Ansonsten können Sie sämtliche Sonderzeichen wie bei Java üblich mit den Unicode-Escape-Sequenzen angeben – hier beispielsweise `\u00a9` für das Copyright-Symbol.

```
//CD/examples/ch03/SwingTest/DokumentFenster.java
```

```

import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class DokumentFenster extends BasisFenster {
    private static int nextNr = 1;
    private static int nextX  = 70;
    private static int nextY  = 40;
    private boolean dirty = false;

    public DokumentFenster() {
        String titel =
            SwingTest.getResBundle().getString("documentTitle");

        if (nextNr > 1) {
            titel = titel + " " + nextNr;
        }
        this.setTitle( titel );
        this.setBounds( nextX, nextY, 400, 300 );

        nextX += 20;
        nextY += 20;
        nextNr++;
    }
}

```

Das Dokument-Fenster zählt mit jedem neuen Dokument die Titelnummer und die Fensterposition hoch. Weitere Grundfunktionen müssen hier nicht implementiert werden, da die Oberklasse `BasisFenster` alles Nötige zur Verfügung stellt.

```

JTextArea text = new JTextArea();
JScrollPane scrollpane =
    new JScrollPane( text,
        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
        JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS );
this.getContentPane().add( scrollpane );

```

Im Gegensatz zum AWT können Sie bei Swing nicht einfach die `add()`-Methode eines `JFrame`-Fensters aufrufen, um eine Komponente hinzuzufügen, da Swing mehrere Komponenten-Ebenen in einem Fenster anbietet. Sie müssen erst mit `getContentPane()` die Inhalts-Ebene abfragen und rufen darauf dann `add()` auf.⁴

Eine Mac-Besonderheit sehen Sie bei der `JTextArea`, die in eine `JScrollPane` eingebettet wird. Während viele Systeme Bildlaufleisten (»Scrollbars«) nur dann anzeigen, wenn sie wirklich benötigt werden, soll der Anwender bei der Apple-Philosophie immer die gesamte Funktionalität sehen, die er aufrufen kann – selbst wenn die Elemente meistens deaktiviert sind. Daher werden der `JScrollPane` die `XXX_SCROLLBAR_ALWAYS`-Konstanten übergeben. Für andere Systeme könnten Sie hier – eventuell mit bedingtem Code – die `XXX_SCROLLBAR_AS_NEEDED`-Konstanten verwenden.

```
text.getDocument().addDocumentListener(  
    new DocumentListener() {  
        public void changedUpdate(DocumentEvent e) {  
            setDirty();  
        }  
  
        public void insertUpdate(DocumentEvent e) {  
            setDirty();  
        }  
  
        public void removeUpdate(DocumentEvent e) {  
            setDirty();  
        }  
    });  
  
miSichern.addActionListener( new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        // Dokument sichern  
        cleanDirty();  
    }  
});  
}
```

An der `JTextArea` wird nun ein `DocumentListener` registriert, der bei jeder Änderung des Textinhalts mit einer seiner drei Methoden aufgerufen wird. Hier wird einfach die gleich dokumentierte Methode `setDirty()` aufgerufen, in der das Dokument-Fenster ein Änderungssymbol im Fenstertitel darstellt. Wenn der Anwender den »Sichern«-Menüpunkt wählt, wird das Dokument

4 Bei Java 1.5 hat Sun endlich diesen Normalfall erkannt. Wenn Sie dort `JFrame.add()` aufrufen, landen Sie automatisch in der Inhalts-Ebene.

bei dieser Beispielanwendung zwar nicht gespeichert, aber zumindest wird das Änderungssymbol wieder entfernt.

```
private void setDirty() {
    if (!dirty) {
        dirty = true;
        this.getRootPane().putClientProperty(
            "windowModified",
            Boolean.TRUE );
    }
}

private void cleanDirty() {
    if (dirty) {
        this.getRootPane().putClientProperty( "windowModified",
            Boolean.FALSE );

        dirty = false;
    }
}
}
```

Bei MacOS X werden Änderungen an einem Dokument nicht mit beispielsweise einem Sternchen (»*) vor dem Fenster-Titel gekennzeichnet, sondern mit einem schwarzen Punkt im Schließfeld des Fensters. Sehen Sie sich noch einmal Abbildung 3.8 am Anfang dieses Abschnitts an – in den Fenstern, in denen Text eingegeben wurde, finden Sie links oben das veränderte Schließfeld. Das Setzen und Löschen dieses Punktes ist ohne großen Aufwand möglich: Auf der Wurzel-Ebene (»Root Pane«) eines JFrame setzen Sie mit `putClientProperty()` einfach die Property `windowModified` – entweder auf `TRUE` oder auf `FALSE` (beides sind konstante Objekte der Klasse `java.lang.Boolean`).

```
//CD/examples/ch03/SwingTest/ProgrammInfo.java
//...
private ProgrammInfo() {
    // ...
    String name = SwingTest.getResBundle().getString("appName")
        + " " + SwingTest.getResBundle().getString("appVersion");
    JLabel text = new JLabel( "<html><center><big>" + name
        + "</big><br>&nbsp;<br><small>" +
        SwingTest.getResBundle().getString("dialogAboutCopyright")
        + "</small></center></html>" );
```

```

        this.getContentPane().add( text );
        // ...
    }

```

Nach dem `BasisFenster` und dem `DokumentFenster` bietet die Klasse `ProgrammInfo` kaum noch Neues (im Wesentlichen kennen Sie sie auch schon vom AWT-Beispiel). Falls Swing für Sie neu ist, können Sie in diesem Konstruktor aber die Mächtigkeit der Komponenten erahnen: Dem `JLabel`-Beschriftungsobjekt wird nicht nur ein einfacher Text mitgegeben, sondern HTML-Code!⁵



Abbildung 3.9 Menüzeilen im Fenster passen nicht zu Aqua.

Wenn Sie das Beispielprogramm nun starten, funktioniert eigentlich alles – aber die Menüzeile befindet sich im Fenster und nicht oben am Bildschirmrand (siehe Abbildung 3.9). Während das AWT die Menüzeile automatisch aus dem Fenster herausnimmt, hat der Programmierer bei Swing mehr Kontrolle – schließlich soll es mit Swing möglich sein, dass eine Oberfläche auf verschiedenen Systemen exakt gleich aussieht.

Dennoch sollten Sie Ihre Applikation nach Möglichkeit auch hier an Mac OS X anpassen und die Menüzeile oben am Bildschirm darstellen lassen. Dazu müssen Sie einfach die System-Property `apple.laf.useScreenMenuBar` (bei Java 1.4) bzw. `com.apple.macos.useScreenMenuBar` (bei Java 1.3) setzen – es schadet auch nicht, beide Properties anzugeben. In Xcode geben Sie die Properties in den Target-Einstellungen im Bereich »Pure Java Specific« ein (siehe Abbildung 3.10).

⁵ Auf der Seite <http://java.sun.com/docs/books/tutorial/uiswing/components/html.html> erfahren Sie mehr über HTML-Code in Swing-Komponenten.

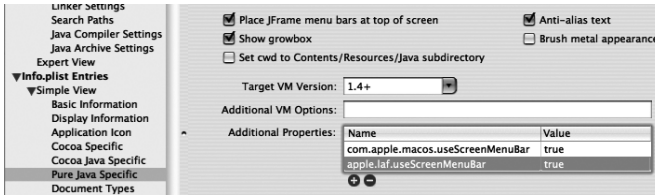


Abbildung 3.10 Properties für die Aqua-Menüzeile in Xcode setzen

Wenn Sie Probleme damit haben, die Menüzeile außerhalb eines Fensters darzustellen, prüfen Sie, ob die drei Voraussetzungen erfüllt sind:

- ▶ Das Aqua-Look & Feel muss verwendet werden – alle anderen PLAFs stellen die Menüzeile im Fenster dar.
- ▶ Innerhalb des `JFrame` müssen Sie die Menüzeile mit `setJMenuBar()` angemeldet haben.
- ▶ Die System-Property `apple.laf.useScreenMenuBar` bzw. `com.apple.macos.useScreenMenuBar` muss gesetzt sein (oder am besten beide).⁶

Wenn Sie einen `javax.swing.JDialog` mit eigener Menüzeile verwenden, werden Sie feststellen, dass dieses Menü immer innerhalb des Dialogs angezeigt wird. Apple rät davon ab, in modalen Dialogen Menüzeilen zu verwenden, denn dies passt nicht zu den Gestaltungsrichtlinien. Ein modaler Dialog soll dem Anwender eine knappe, präzise Auswahl anbieten und ihn nicht mit zu vielen Möglichkeiten überfordern – alles andere ist viel besser in einem nichtmodalen Fenster aufgehoben.

3.2.3 Portables Einbinden der Java 1.4-Apple-Erweiterungen

Wie schon bei den alten MRJ-Erweiterungen stellt Apple die Schnittstellen der neuen Erweiterungen als Stub-Klassen zur Verfügung, die Sie auf der Seite <http://developer.apple.com/samplecode/AppleJavaExtensions/AppleJavaExtensions.html> herunterladen können. Allerdings weist Apple nun explizit darauf hin, dass diese Klassen nicht für einen Einsatz zur Laufzeit gedacht sind – Sie können damit also nur an MacOS X angepassten Java-Code auf fremden Systemen kompilieren.

Eine vernünftige Lösung zum portablen Einbinden der neuen Apple-Erweiterungen bekommen Sie also wieder nur durch den Einsatz von Java Reflection. Auch für Apples Java 1.4-Erweiterungen finden Sie im Paket `com.muchsoft.util.mac` eine Hilfsklasse, die Ihnen die meiste Arbeit

⁶ Yes, there is a step 3! ;-)

abnimmt. Dazu muss Ihre Applikation anstelle des `com.apple.eawt.ApplicationListener-Interfaces` nun die Schnittstelle `com.muchsoft.util.mac.Javal4Handler` implementieren:

```
//CD/examples/ch03/SwingPortabel/SwingPortabel.java
import java.awt.*;
import java.util.*;
import javax.swing.*;
import com.muchsoft.util.Sys;
import com.muchsoft.util.mac.*;

public class SwingPortabel extends BasisFenster
    implements Javal4Handler {
    private static SwingPortabel app;
    private static ResourceBundle resbundle;

    public SwingPortabel() {
        // ...
        Javal4Adapter.registerJaval4Handler( this );
        Javal4Adapter.setEnabledPrefs( false );
        // ...
    }

    public void handleAbout(EventObject event) {
        ProgrammInfo.zeigeInfo();
        Javal4Adapter.setHandled( event, true );
    }

    public void handleQuit(EventObject event) {
        Javal4Adapter.setHandled( event, false );
        // Methoden zum Sichern der Daten aufrufen
        System.exit(0);
    }

    public void handlePrefs(EventObject event) { }
    public void handleOpenApplication(EventObject event) { }
    public void handleReOpenApplication(EventObject event) { }
    public void handleOpenFile(EventObject event, String filename) { }
    public void handlePrintFile(EventObject event, String filename) { }
    // ...
}
```


Das Anmelden beim System erfolgt im Konstruktor mit der statischen Methode `Javal4Adapter.registerJaval4Handler()`. Die Handler-Methoden sehen nun geringfügig anders aus: Anstelle des Apple-spezifischen `ApplicationEvent`-Objekts wird diesen Methoden ein allgemein gültiges `java.util.EventObject` übergeben. Der Aufruf von `setEnabledPrefs()` ist hier eigentlich unnötig, zeigt Ihnen aber, wie Sie den »Einstellungen...«-Menüpunkt aktivieren können.

Die Klasse `Javal4Adapter` und weitere interne Klassen sind hier nicht mehr abgedruckt, da sie sehr ähnlich den entsprechenden Klassen für die Java 1.3-Erweiterungen aufgebaut sind, die Sie bereits beim AWT kennen gelernt haben. Auf der Buch-CD finden Sie aber im Verzeichnis `/utility/sys/source/` alle Quelltexte.

Eine ähnliche Lösung bietet Apple mit dem **OSXAdapter** an, den Sie von <http://developer.apple.com/samplecode/OSXAdapter/OSXAdapter.html> herunterladen können. Dieser Code ist aber enger mit dem eigentlichen Applikationscode verwoben und muss für jede Anwendung neu programmiert werden.

3.2.4 Hinweise zu diversen Swing-Komponenten

Mnemonics, einzelne unterstrichene Buchstaben zur Tastaturbedienung von Menüs und Menüeinträgen, passen nicht zu MacOS X und Aqua – die mit `setAccelerator()` gesetzten Tastaturkürzel rechts neben den Einträgen gelten als vollkommen ausreichend. Entsprechend werden die Mnemonics auch gar nicht angezeigt, wenn die Menüzeile oben am Bildschirmrand dargestellt wird, sondern nur innerhalb eines Fensters (siehe Abbildung 3.11). Verzichten Sie unter MacOS X also am besten komplett auf Mnemonics, beispielsweise indem Sie die Methode `setMnemonic()` – wenn überhaupt – nur auf anderen Systemen aufrufen.



Abbildung 3.11 Menü-Mnemonics passen nicht zu Aqua.



Abbildung 3.12 JFileChooser passt nicht zum System.

Auch wenn Swing einen eigenen Dateiauswahl-Dialog in der Klasse `javax.swing.JFileChooser` mitbringt, sollten Sie nach Möglichkeit auch bei Swing-Anwendungen den AWT-Dateiauswahl-Dialog `java.awt.FileDialog` verwenden. Weil der AWT-Dialog vom System dargestellt wird, muss sich der Anwender nicht an den umständlich zu bedienenden Swing-Dialog gewöhnen – und dies gilt nicht nur für Mac OS X, sondern für viele Systeme. Weil der Dateiauswahl-Dialog modal ist, gibt es an dieser Stelle auch keine Probleme beim Vermischen von AWT- und Swing-Komponenten.

Wenn Sie dennoch mit `JFileChooser` arbeiten wollen oder müssen, können Sie zumindest konfigurieren, ob der Dateiauswahl-Dialog Installations- und Programmpakete (Bundles) als Verzeichnisse oder als eine Datei anzeigt. Dafür stehen Ihnen zwei Properties zur Verfügung:

- ▶ `JFileChooser.packageIsTraversable`
Setzen Sie diese Property auf `never`, wenn Installationspakete (`.pkg`) und Programmpakete (`.app`) als eine Datei angezeigt werden sollen, oder auf `always`, wenn beide als Verzeichnisse erscheinen sollen.
- ▶ `JFileChooser.appBundleIsTraversable`
Setzen Sie alternativ diese Property auf `never`, wenn Programmpakete als Datei, Installationspakete aber als Verzeichnis erscheinen sollen.

Normalerweise sollten Sie `JFileChooser.packageIsTraversable` auf `never` setzen. Das Setzen der Properties nehmen Sie entweder pro Dateiauswahl-Dialog mit der Methode `putClientProperty()` oder für alle Dateiauswahl-Dialoge mit `UIManager.put()` vor.

Beachten Sie, dass diese Properties nur dann beachtet werden, wenn der Dateiauswahl-Dialog mit dem Aqua-Look & Feel angezeigt wird.



Abbildung 3.13 Aqua stellt JTabbedPane anders dar.

Tabs (Karteireiter) werden beim Aqua-Look & Feel niemals gestapelt, wie dies andere PLAFs tun, wenn der Platz knapp wird (siehe Abbildung 3.13 rechts). Stattdessen stellt Aqua automatisch ein Menü dar, in dem die fehlenden Tabs anwählbar sind. Dies kann unter Umständen Unterschiede bei dem Platz ausmachen, den die übrigen Komponenten einnehmen können. Ein weiterer Punkt ist die Tatsache, dass Aqua-Tabs eine bestimmte Standardgröße besitzen – wenn Sie ein Bild in das Tab einfügen, wird beim Aqua-Aussehen das Bild auf die Größe des Tabs skaliert und nicht umgekehrt.

Seit langem stellt Apple ausführliche Richtlinien für die Gestaltung von Benutzungsoberflächen zur Verfügung, die auch die Größe von Komponenten (innerhalb eines gewissen Rahmens) festlegen. So darf eine »Combo Box« eine bestimmte Höhe nicht überschreiten – und dafür ignoriert das Aqua-Look & Feel zur Not sogar die Vorgabe eines Layout-Managers, der die Komponente eigentlich auf die maximal mögliche Höhe ziehen will (und dies bei anderen PLAFs auch tut, siehe Abbildung 3.14). Auch wenn die hier benutzte `JComboBox` ein Extremfall ist, gibt Ihnen dieses Beispiel doch eine Idee davon, wie häufig das Aqua-Look & Feel einen Kompromiss zwischen dem Java-Layout und den Systemrichtlinien eingehen muss (wenn auch oft in viel kleinerem Maße als hier gezeigt) – testen Sie Ihre Oberflächen daher gut, wenn Sie sichergehen wollen, dass sie mit Aqua bedienbar sind. Als Faustregel können Sie sich merken, dass eine Oberfläche, die Sie mit Aqua entworfen haben, in der Regel auch mit anderen PLAFs problemlos genutzt werden kann – umgekehrt erfordert dies auf jeden Fall Tests und gegebenenfalls Anpassungen.

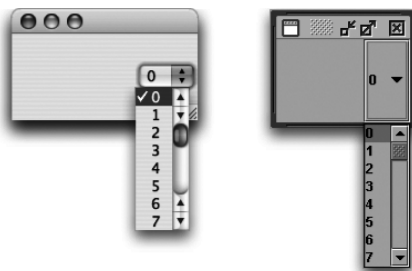


Abbildung 3.14 Aqua beachtet nicht immer den Layout-Manager.

Weitere kleine Details von Apples Swing- und AWT-Implementation sind im Dokument <http://developer.apple.com/documentation/Java/Conceptual/Java141Development/> beschrieben, aber beim normalen Programmieren kommen diese eher selten zum Tragen.

3.3 Fenster mit »Brushed Metal«-Aussehen

Apple verwendet bei einigen Anwendungen als Fensterhintergrund anstelle der Aqua-Querstreifen eine Textur, die wie gebürstetes Metall (»Brushed Metal«) aussieht. Die Idee dahinter ist, dass von solchen Fenstern bzw. Programmen Geräte repräsentiert werden, die der Anwender aus seinem täglichen Leben kennt – beispielsweise einen DVD-Player (/Programme/DVD Player), einen CD-/MP3-Player (/Programme/iTunes) oder einen Taschenrechner (/Programme/Rechner). Dementsprechend sollte eine Applikation, die das Brushed-Metal-Aussehen verwendet, aus nur einem Hauptfenster bestehen, das die »Geräteoberfläche« darstellt. Leider wird diese Idee immer mehr verwässert, auch von Apple selbst – denken Sie nur an die Ränder der Finder-Fenster unter Mac OS X 10.3.

Ab Java 1.4 können Sie das Brushed-Metal-Aussehen auch bei AWT-Applikationen nutzen – allerdings klappt dies erst seit Mac OS X 10.3 richtig, mit Version 10.2 gab es noch zahlreiche Probleme. Beachten Sie, dass »Brushed Metal« nur eine Textur ist und kein komplettes Look & Feel (PLAF), daher wird die Verwendung zusammen mit Swing von Apple nicht unterstützt. Auf der Buch-CD finden Sie im Verzeichnis /examples/ch03/BrushMetalTest/ eine kleine Beispielanwendung (siehe Abbildung 3.15).



Abbildung 3.15 AWT-Frame mit »Brushed Metal«-Aussehen

Zum Aktivieren von »Brushed Metal« müssen Sie einfach nur die System-Property `apple.awt.brushMetalLook` auf »true« setzen, dann wird automatisch der Standardhintergrund eines `java.awt.Frame`-Fensters als gebürstetes Metall gezeichnet. Im Gegensatz zu den meisten anderen Properties funktioniert das dynamische Setzen zur Laufzeit mit `System.setProperty()` aller-

dings nicht! Entweder geben Sie die Property daher mit der Kommandozeilenoption `-D` an, oder Sie setzen sie in Xcode bei der Programmpaket-Konfiguration (siehe Abbildung 3.16). Beide Möglichkeiten werden im folgenden Kapitel 4, *Ausführbare Programme*, genauer besprochen.

Bei der Gelegenheit sollten Sie dann auch die Anzeige des Größenveränderungsfeldes deaktivieren, indem Sie die Property `apple.awt.showGrowBox` auf »false« setzen. Selbst wenn das Fenster in der Größe verändert werden kann, würde das derzeitige Aussehen der »GrowBox« unten rechts im Fenster eher stören – hier müsste Apple nachbessern und das Aussehen z.B. dem iTunes-Größensymbol angleichen.



Abbildung 3.16 Xcode-Einstellungen für »Brushed Metal«

Mit ein paar Tricks kann das Brushed-Metal-Aussehen dennoch mit Swing genutzt werden. Wenn Sie einen `javax.swing.JFrame` verwenden und diesem einen Hintergrund mit voller Transparenz (Alpha-Wert 0) geben, wird der Hintergrund zwar als gebürstetes Metall angezeigt, es gibt dann aber sehr viele Fehler beim Neuzeichnen des Fensters. Nehmen Sie daher statt des Swing-Fensters einen `java.awt.Frame` mit weißem Hintergrund und fügen Sie in diesen Frame ein `java.awt.Panel` mit transparentem Hintergrund ein. Ihre Swing-Komponenten können Sie nun in dieses Panel einfügen. Die Vermischung von Swing- und AWT-Komponenten ist zwar nicht ideal, aber immerhin funktioniert das Neuzeichnen damit relativ problemlos. Denken Sie aber immer daran, dass dieses Vorgehen von Apple offiziell nicht unterstützt wird und daher mit einer neuen Java-Version hinfällig werden kann.

3.4 Drag & Drop

Drag & Drop (abgekürzt »DnD«) mit den Klassen aus den Paketen `java.awt.dnd` und `java.awt.datatransfer` funktioniert mit MacOS X weitestgehend problemlos – allerdings sollten Sie mindestens das Java 1.4.2 Update 1 installiert haben, das viele Fehler in diesem Bereich beseitigt hat (davor mussten Objekte beispielsweise zwingend das Interface `java.io.Serializable` implementieren, um innerhalb der Applikation übertragen werden zu können). Nach wie vor scheint der DnD-Mechanismus aber

zuverlässiger zu funktionieren, wenn Ihr `java.awt.datatransfer.Transferable`-Objekt immer zusätzlich auch den Datentyp `java.awt.datatransfer.DataFlavor.stringFlavor` unterstützt.

Ein wichtiger Unterschied zwischen MacOS X und anderen Systemen besteht bei den Umschalttasten, welche die verschiedenen DnD-Aktionen ansteuern. Folgende Tabelle gibt Ihnen einen Überblick:












	Kopieren (Copy)	Bewegen (Move)	Verknüpfen (Link)	Verhalten
Java			 + 	Innerhalb und zwischen Java-Anwendungen; von nativen zu Java-Anwendungen
Mac OS X			 + 	Innerhalb und zwischen Java-Anwendungen; zwischen nativen und Java-Anwendungen
DnDConstants-Konstante:	ACTION_COPY	ACTION_MOVE	ACTION_LINK	

Tabelle 3.1 Auswirkungen der Umschalttasten bei Drag & Drop

Das Problem dabei ist, dass MacOS X Mausklick-Aktionen bei gedrückter -Taste immer als Aufruf eines Kontext-Popup-Menüs erkennt – eine DnD-Aktion kann bei MacOS X also niemals mit gedrückter -Taste gestartet werden. Gleiches betrifft Rechtsklick-DnD-Aktionen, die nicht unterstützt werden, weil sie intern auf +Klick abgebildet werden.

Als Lösung sollte Ihre Java-Anwendung zumindest unter MacOS X die Mac-typischen Umschalttasten für DnD-Operationen verwenden. Entweder fragen Sie dazu den Tastenstatus direkt ab (mit speziellem Code für MacOS X), oder Sie fragen einfach den Wert von `java.awt.dnd.DragSourceDragEvent.getUserAction()` ab und testen ihn auf die entsprechende Konstante aus der Klasse `java.awt.dnd.DnDConstants`.

3.5 »Headless«-Applikationen ohne Benutzeroberfläche

Gerade bei Server-Anwendungen ist es oft wünschenswert, dass das Programm im Hintergrund läuft, ohne dass der Anwender etwas davon mitbekommt. MacOS X und auch MacOS X Server zeigen aber zumindest ein Programm-

symbol im Dock an, mit dem der Anwender das Programm auch beenden kann.

Für solche Fälle hat Sun die AWT-Bibliothek ab **Java 1.4** um den so genannten »Headless«-Modus erweitert (siehe <http://java.sun.com/j2se/1.4.2/docs/guide/awt/AWTChanges.html#headless>). Ist dieser Modus aktiviert, werden grafische Elemente nicht nur unterdrückt, sie können vom Programm auch gar nicht mehr erzeugt werden. Versucht eine Headless-Anwendung dies dennoch, wird eine `java.awt.HeadlessException` geworfen. Programme, die sowohl auf Desktop-Rechnern mit Maus, Tastatur und Bildschirm als auch auf Server-Rechnern ohne diese Geräte eingesetzt werden sollen, müssen nun anstelle von

```
class MeinMenue {
    static Choice c = new Choice();
}
```

korrekterweise wie folgt programmiert werden:

```
class MeinMenue {
    static Choice c;
    static {
        try {
            c = new Choice();
        }
        catch (HeadlessException e) {
            // ...
        }
    }
}
```

Aktiviert wird der Headless-Modus durch das Setzen der System-Property `java.awt.headless`, beispielsweise beim Aufruf aus der Kommandozeile:

```
java -Djava.awt.headless=true MeinProgramm
```

Für **Java 1.3** hatte Apple davor schon eine Mac OS X-spezifische Lösung bereitgestellt: Mit `-Dcom.apple.backgroundOnly=true` können Sie die Anzeige des Dock-Symbols und der Menüzeile unterdrücken. Wenn Ihre Java-Anwendung in einem Programmpaket verpackt ist, erreichen Sie dies alternativ auch durch das Setzen des `Info.plist`-Schlüssels `LSBackgroundOnly` auf den Wert 1 (siehe Kapitel 4, *Ausführbare Programme*). Eine `HeadlessException` wird in beiden Fällen nicht geworfen.

Sowohl die Java 1.3- als auch die Java 1.4-Lösung verhindern allerdings nicht, dass die Anwendung beendet wird, wenn sich der Benutzer abmeldet.

3.6 Literatur & Links

- ▶ Loy/Eckstein/Wood/Elliott/Cole, »Java Swing« 2nd ed., O'Reilly 2002
Dieses ziemlich umfangreiche Buch behandelt neben den Grundlagen die einzelnen Swing-Bereiche ausführlich mit vielen Beispielen. Am Ende werden auch wichtige Interna diskutiert.
- ▶ David Flanagan, »Java Foundation Classes in a Nutshell«, O'Reilly 1999
Ein etwas älteres Werk, das aber die Grundlagen vom AWT und von Swing kurz und knapp gut erklärt.
- ▶ <http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHI-Guidelines/>
Die »Apple Human Interface Guidelines« (HIG) legen sehr detailliert fest, wie die Oberflächen von MacOS X-konformen Anwendungen auszusehen haben.
- ▶ <http://www.javadesktop.org/>
Die Portalseite rund um GUI-Entwicklung mit Java
- ▶ <https://forms.dev.java.net/>
Das JGoodies-»Forms«-Framework erleichtert das Anordnen von Swing-Komponenten. Die kostenlosen und kommerziellen JGoodies-Produkte (<http://www.jgoodies.com/>) machen regen Gebrauch davon.
- ▶ <http://www.randelshofer.ch/quaqua/download.html>
Der »Quaqua«-Look & Feel ist eine Ergänzung und Korrektur von Apples Aqua-Look & Feel, der viele kleine Unschönheiten beseitigt.

4 Ausführbare Programme

4.1	Shell-Skripte im Terminal	176
4.2	Doppelklickbare JAR-Archive	182
4.3	Mac OS X-Applikationen	187
4.4	Installationswerkzeuge	219
4.5	Java Web Start	234
4.6	Applets	250
4.7	Literatur & Links	282

- 1 **Grundlagen**
- 2 **Entwicklungsumgebungen**
- 3 **Grafische Benutzungsoberflächen (GUI)**
- 4 **Ausführbare Programme**
- 5 **Portable Programmierung**
- 6 **Mac OS X-spezielle Programmierung**
- 7 **Grafik und Multimedia**
- 8 **Werkzeuge**
- 9 **Datenbanken und JDBC**
- 10 **Servlets und JavaServer Pages (JSP)**
- 11 **J2EE und Enterprise JavaBeans (EJB)**
- 12 **J2ME und MIDP**
- A **Kurzeinführung in die Programmiersprache Java**
- B **Java auf Mac OS 8/9/Classic**
- C **Java 1.5 »Tiger«**
- D **System-Properties**
- E **VM-Optionen**
- F **Xcode- und Project Builder-Einstellungen**
- G **Mac OS X- und Java-Versionen**
- H **Glossar**
- I **Die Buch-CD**

4 Ausführbare Programme

»Vestis virum reddit.« (Quintilian)

»Kleider machen Leute.« (Gottfried Keller)

4

Natürlich müssen Programme ausführbar sein – die Frage ist nur, wie? Wie benutzerfreundlich kann der Anwender das Programm starten, und wie präsentiert sich ihm die Applikation – als Sammlung von hunderten von Klassen- und Bilddateien oder als eine einzige Datei? Kommt die Anwendung mit einem langweiligen Standardsymbol daher, oder besitzt sie ein eigenes, schönes Programmsymbol? Neben der eigentlichen Programmierung und der Anpassung der grafischen Benutzungsoberfläche wird es also entscheidend von der »Verpackung« abhängen, ob Ihre Anwendung sofort als Java-Programm entlarvt wird oder ob Sie auf die Anwender wie eine echte Mac-Applikation wirkt.

Es gibt zwei grundsätzliche Möglichkeiten, wie Sie Ihre Programme an die Anwender ausliefern können:

1. Als »normale« lokale Programme, darunter fallen Shell-Skripte, doppelklickbare JAR-Archive und »richtige« Mac OS X-Applikationen.
2. Bei Bedarf von einem Web-Server an den Benutzer übertragene Software, hier werden Sie die Java Web Start-Technologie und Applets kennen lernen.

Das Ziel ist, dass der Anwender nur eine einzige Datei (oder eine einzige Webadresse) kennen muss, um die Software nutzen zu können.

Und obwohl bei lokalen Programmen alle diese Forderungen und Wünsche nur vom Programmtyp »Mac OS X-Applikation« erfüllt werden, bekommen Sie vorher auch noch ausführliche Informationen zu den anderen Programmtypen. Zum einen basieren Mac OS X-Applikationen teilweise auf diesen Techniken (JAR-Archive), zum anderen stellen Shell-Skripte und JAR-Archive eine einfache Möglichkeit dar, ohne großen Aufwand auf fremden Systemen kleine Anpassungen für den Mac vornehmen zu können.

Bei den vom Web-Server gelieferten Programmen werden Sie den in Mac OS X eingebauten Apache-Web-Server kennen lernen, außerdem bekommen Sie einen schnellen Überblick über die typischen Web-Browser, die auf Mac OS X eingesetzt werden. Als Abschluss dieses Kapitels wird auch die Java-JavaScript-Kommunikation kurz besprochen, mit der Sie innerhalb eines HTML-Dokuments auf die Methoden eines Applets zugreifen können (und umgekehrt).

4.1 Shell-Skripte im Terminal

Bereits im ersten Kapitel haben Sie gesehen, wie Sie mit dem Kommandozeilen-Tool `java` einzelne Klassen ausführen lassen können, die eine `main`-Methode enthalten. Während die Verwendung der Kommandozeile nicht besonders Mac-typisch ist, hat dieses Vorgehen einen enormen Vorteil: `java` steht nahezu überall dort zur Verfügung, wo ein Java Runtime Environment (JRE) installiert ist! Bevor wir uns den stärker grafisch orientierten Programmtypen zuwenden, lernen Sie daher zunächst noch Shell-Skripte kennen, mit denen sich Kommandozeilenbefehle zusammenfassen und automatisieren lassen.

Wenn Sie Apples Entwicklerwerkzeuge installiert haben, finden Sie im Verzeichnis `/Developer/Examples/Java/` zahlreiche Beispielprogramme, so dass Sie folgende zwei Befehle im Terminal eingeben (und damit sofort ausführen) können:

```
set PFAD="/Developer/Examples/Java/JFC/Java2D/Java2D.jar"  
java -classpath $PFAD java2d.Java2Demo &
```

Es wird das Java2D-Beispiel von Sun gestartet, das die verschiedenen Aspekte von Javas 2D-Grafik demonstriert. Das Kaufmannsund (&) am Ende des zweiten Befehls dient dazu, das Java-Programm als neuen, separaten Prozess zu starten, damit Sie im Terminal sofort weiterarbeiten können. Ansonsten wird die Terminal-Shell nämlich so lange blockiert, bis das Java-Programm beendet wurde – entweder regulär mit dem Menüpunkt »Exit« oder »Quit java2d.Java2Demo« oder aber etwas unsanft, indem Sie `Ctrl`+`C` drücken, während das Terminal-Fenster aktiv ist. Als Name der zu startenden Klasse wird hier `java2d.Java2Demo` übergeben, also der voll qualifizierte Klassenname inklusive Paketname.

Zuvor wird im ersten Befehl die Umgebungsvariable `PFAD` auf das entsprechende JAR-Archiv gesetzt, das alle Klassen dieses Java-Programms enthält. Beim `java`-Aufruf wird bei der `classpath`-Option mit `$PFAD` lesend darauf zugegriffen. In diesem Abschnitt wird die Umgebungsvariable als Abkürzung für die komplette Pfadangabe verwendet, aber Sie können überall dort, wo auf `$PFAD` zugegriffen wird, natürlich auch immer den gesamten Pfad hinschreiben.

Denken Sie daran, dass Sie Pfade leicht in die Shell eingeben können, indem Sie die entsprechenden Dateien aus einem Finder-Fenster auf das Terminal-Fenster ziehen!

Shell-Skripte sind nun im Wesentlichen nichts anderes als Textdateien, in denen Listen solcher Kommandozeilenbefehle gespeichert sind. Unter Windows werden sie Stapelverarbeitungs-Dateien (Batch-Dateien) genannt und besitzen die Dateinamenserweiterung `.BAT`. Unter UNIX benötigen die Skripte keine spezielle Endung, häufig wird aber `.sh` verwendet. Geben Sie nun folgendes Skript mit einem Text-Editor ein (z.B. mit `pico` im Terminal) und speichern Sie es unter dem Namen `j2d.sh` in Ihrem Benutzerverzeichnis (Home):

```
#!/bin/tcsh
# CD/examples/ch04/j2d.sh
set PFAD="/Developer/Examples/Java/JFC/Java2D/Java2D.jar"
java -classpath $PFAD java2d.Java2Demo
```

Listing 4.1 Shell-Skript `j2d.sh`

In der ersten Zeile des Skripts wird die Shell ausgewählt, mit der das Skript ausgeführt werden soll. Häufig finden Sie dort `#!/bin/sh` für die Bourne-Shell, in diesem Skript wird die `tcsh`-Shell verwendet. Danach folgt eine Kommentarzeile und schließlich die beiden Befehle, die Sie schon kennen.

Passen Sie auf, wenn Sie BBEdit oder ähnliche Editoren verwenden! Mac-Programme speichern Zeilenumbrüche normalerweise im Mac-Format, und damit kommen die Shells nicht zurecht – die Skripte werden dann nicht als solche erkannt und können nicht ausgeführt werden. Wandeln Sie daher die Zeilenenden vor dem Speichern explizit in das UNIX-Format um, was beispielsweise BBEdit komfortabel über ein Icon in der Toolbar anbietet. Wenn die Skriptdatei dagegen schon mit den falschen Zeilenenden auf der Festplatte liegt, können Sie die Datei mit dem Kommandozeilen-Tool `tr` umwandeln:

```
tr \r \n < mac_datei_name > unix_datei_name
```

Dieser Befehl liest die Mac-Datei ein und erzeugt daraus eine neue Datei mit den korrekten UNIX-Zeilenden (mit dem Kleiner- und Größerzeichen wird in UNIX eine Umlenkung der Ein- und Ausgabe von Tastatur und Bildschirm auf Dateien erreicht). Wer die Umwandlung lieber mit einem grafischen Tool durchführen möchte, findet unter der Adresse <http://www.maclester.edu/~jaas/linebreak.html> das kostenlose Programm »LineBreak«.

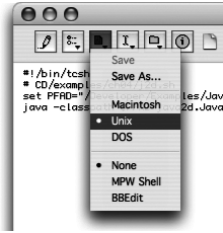


Abbildung 4.1 Umwandlung von Zeilenenden mit BBEdit Lite

Damit das Skript ausführbar ist, müssen Sie nun noch die Zugriffsrechte mit einem Shell-Kommando passend setzen.

```
[straylight:~] much% ls -l j2d.sh
-rw-r--r--  1 much  staff  135 19 Dec 19:14 j2d.sh
```

Ganz links sehen Sie `rw-`, als Benutzer dürfen Sie die Datei also lesen und schreiben – aber nicht ausführen. Mit dem Kommando

```
chmod u+x j2d.sh
```

fügen Sie den Benutzerrechten (das »u« steht für **user**) das Ausführungsbit »x« (**e**xecution) hinzu, so dass die Rechte in der Shell danach wie folgt aussehen:

```
-rwxr--r--  1 much  staff  135 19 Dec 19:14 j2d.sh
```


Das `rwx` bei den Benutzerrechten zeigt an, dass Sie die Datei nun lesen, schreiben und ausführen dürfen! Gestartet wird ein Shell-Skript mit seinem Zugriffspfad und Dateinamen, in diesem Fall also durch Eingabe von `./j2d.sh`. Wenn Sie die Pfadangabe (hier `./`, also das aktuelle Verzeichnis) vergessen, findet die Shell das Skript nicht. Alternativ können Sie das Skript von einem neuen Shell-Prozess ausführen lassen, indem Sie `tcsh j2d.sh` eingeben.

Das Programm wird vom Skript nicht parallel gestartet, da das Kaufmannsund hinter dem `java`-Aufruf fehlt. Dort könnte es zwar stehen, aber schreiben Sie es bei Bedarf besser hinter den Skript-Aufruf (hier also `./j2d.sh &`). Dadurch hat der Aufrufer des Skripts die Kontrolle darüber, ob die Ausführung parallel erfolgen soll oder nicht.

Wenn Sie einem Skript statt der Endung `.sh` die Endung `.command` geben, können Sie es unter Mac OS X mit einem Doppelklick starten. Ausgeführt wird das Skript dann in einem neuen Terminal-Fenster. Um den Dateinamen zu ändern, selektieren Sie entweder das Dateisymbol im Finder und drücken `⌘+I`, oder Sie klicken das Icon mit gedrückter Control-Taste an und wählen im erscheinenden Kontext-Popup den Menüeintrag »Information« aus. Dort können Sie dann unter »Name & Suffix« die Dateinamenserweiterung abändern.



Abbildung 4.2 Datei-Information

Sobald Sie den Informationsdialog schließen oder mit  den Eingabefokus weiterschalten, übernimmt Mac OS X den neuen Namen nach einer Nachfrage. Diese Nachfrage erscheint immer dann, wenn die Endung – und damit der Dateityp – geändert wird, weil davon auch die Zuordnung unter »Öffnen mit« betroffen ist. Wie Sie in Abbildung 4.2 sehen, werden `.command`-Dateien standardmäßig mit dem Terminal assoziiert.¹

Sie können den Dateinamen auch direkt im Finder ändern, indem Sie den Text unter dem Icon anklicken. Falls dort die Dateinamenserweiterung nicht angezeigt wird, müssen Sie im Datei-Informationsdialog im Bereich »Name & Suffix« die Option »Suffix ausblenden« ausschalten. Eine Umbenennung der Datei ist natürlich auch mit dem Shell-Befehl `mv` möglich.

Mit einem Shell-Skript, wie Sie es gerade gesehen haben, können Sie Java-Programme problemlos starten. Es gibt aber ein paar Unschönheiten, die eine so gestartete Applikation nicht wie eine »richtige« Mac-Anwendung aussehen lassen. Zum einen wird als Titel des Programm-Menüs oben links auf dem Bildschirm der voll qualifizierte Klassenname angezeigt, wo eigentlich ein mög-

¹ Es gibt einige Hilfsprogramme, die andere Mechanismen als den Dateinamen verwenden, um Shell-Skripte per Maus im Finder starten zu können. Eines davon ist das Programm DropScript, das Sie unter <http://www.advogato.org/proj/DropScript/> herunterladen können. Solche »externen« Lösungen haben aber oft Nachteile – DropScript beispielsweise verlangt (passend zu seinem Namen), dass Sie Dateien auf das Skriptsymbol ziehen, um es zu starten.

lichst kurzer, prägnanter Name stehen sollte. Zum anderen wird unten im Dock nur ein generisches Symbol für Java-Applikationen angezeigt, während andere Anwendungen hier eigene, »schönere« Symbole verwenden.



Abbildung 4.3 Das Programmsymbol ist zunächst zu allgemein, der Name zu lang.

Apple bietet zur einfachen Anpassung von Java-Applikationen die Nicht-Standard-Option `-Xdock` beim Start der Java-Laufzeitumgebung aus der Kommandozeile heraus an.² Mit dem folgenden Aufruf legen Sie den Programmnamen sowohl für das Dock-Symbol als auch für das Programm-Menü in der Menüzeile fest. Beachten Sie, dass der Programmname nicht länger als 16 Zeichen sein sollte.

```
java -cp $PFAD -Xdock:name="Java2D Test" java2d.Java2Demo
```

Die Option `-classpath` ist hier mit `-cp` abgekürzt. Beide Optionen sind gleichbedeutend, der kürzere Name wurde vor einiger Zeit aus Gründen der Bequemlichkeit eingeführt. Der exakte Optionsname zum Setzen des Programmnamens lautet `-Xdock:name`, der eigentliche Name wird dann nach dem Gleichheitszeichen übergeben. Die Anführungszeichen sind hier wegen des Leerzeichens im Namen zwingend notwendig, es schadet aber nicht, sie immer zu verwenden.

Mit der entsprechenden Option `-Xdock:icon` können Sie das Programmsymbol festlegen. Als Wert wird hierbei der Pfad und Name einer `icns`-Datei angegeben. Die `icns`-Dateinamenserweiterung bezeichnet Icon-Ressourcendateien, in denen Mac OS X die Grafikdaten von Symbolen erwartet.

```
java -cp $PFAD -Xdock:icon="duke.icns" java2d.Java2Demo
```

Bei diesem Beispiel muss die Datei `duke.icns` im aktuellen Arbeitsverzeichnis liegen. Damit Sie den Aufruf testen können, kopieren Sie sich bitte einfach die Datei aus dem Verzeichnis `/examples/ch04/` von der Buch-CD. Wie Sie solche Dateien selbst erzeugen, ist dann in Abschnitt 4.3 beschrieben.

² Sie können sich alle verfügbaren Nicht-Standard-Optionen mit `java -X` anzeigen lassen. Obwohl es sich um Nicht-Standard-Optionen handelt, stehen viele von ihnen auch unter anderen Betriebssystemen zur Verfügung.

Die beiden Optionen für den Programmnamen und das Symbol können natürlich kombiniert werden, so dass sich das Java-Programm mit folgendem Aufruf schon deutlich besser in das System integriert, wie Abbildung 4.4 zeigt:

```
java -cp $PFAD -Xdock:name="Java2D Test":icon="duke.icns"
    java2d.Java2Demo
```



Abbildung 4.4 Spezialisiertes Programmsymbol und besserer Name

Problematisch ist die Verwendung von `-Xdock` insofern, als es sich um eine Nichtstandardoption handelt, die sich theoretisch jederzeit ändern (oder ganz verschwinden) könnte. Andererseits haben Sie bei Shell-Skripten gar keine andere Wahl ...

In älteren Dokumentationen findet Sie häufig noch die Umgebungsvariable `com.apple.mrj.application.apple.menu.about.name` zum Setzen des Programmnamens mit der Standardoption `-D`:

```
java -cp $PFAD -Dcom.apple.mrj.application.apple.menu.
    about.name="Java2D Test" java2d.Java2Demo
```

Da die Verwendung dieser System-Property aber zu unerwünschten Seiteneffekten führen kann, ist sie nicht mehr von Apple dokumentiert und sollte auch nicht mehr verwendet werden. Mit `-Xdock:name` haben Sie für Shell-Skripte eine Alternative, und bei einer Mac OS X-Applikationen werden Sie später einen ganz anderen Mechanismus einsetzen.

Mit der Standardoption `-D` können Sie festlegen, dass die Menüleiste immer oben am Bildschirm angezeigt wird und nicht im Fenster, wie das bei Swing-Oberflächen sonst üblich ist. Das ist beim Java2D-Test aber gar nicht mehr nötig, denn die Sun-Beispiele enthalten schon entsprechende Quellcode-Anpassungen, wie sie in Kapitel 3, *Grafische Benutzungsoberflächen*, vorgestellt wurden! Ansonsten würden Sie ohne Quelltext-Änderung folgenden Aufruf verwenden:

```
java -cp $PFAD -Dapple.laf.useScreenMenuBar="true"
    java2d.Java2Demo
```

Letztendlich ist diese Konfiguration über die Kommandozeile aber recht mühselig und nur dann wirklich zu empfehlen, wenn Sie Ihre Java-Anwendung mit

extrem geringem Aufwand oder aber unter einem anderen Betriebssystem an MacOS X anpassen wollen. Wenn Sie dagegen unter MacOS X entwickeln, erfahren Sie in Abschnitt 4.3, wie Sie mit speziellen Mac-Programmierwerkzeugen ein an MacOS X angepasstes Java-Programm erstellen, das aus einer einzigen Mac-spezifischen Programmdatei besteht und per Doppelklick gestartet werden kann. Doch zuvor sehen Sie im folgenden Abschnitt noch eine Möglichkeit, um zumindest den Programmstart per Doppelklick portabel zu realisieren.

4.2 Doppelklickbare JAR-Archive

JAR-Archive fassen Klassen und sonstige Programmressourcen (beispielsweise Bilder) zusammen, die sonst als einzelne Dateien in Verzeichnishierarchien auf der Festplatte gespeichert sind. Viele einzelne Dateien sind nicht nur unschön für den Anwender, der eher eine einzige Programmdatei erwartet, auch die Distribution Ihrer Applikation wird dadurch unnötig erschwert. Ein einziges JAR-Archiv lässt sich dagegen einfach kopieren und wird vom Anwender leichter als »das Programm« wahrgenommen.

Sie haben JAR-Archive schon als Bibliotheken kennen gelernt, die Sie beim Aufruf des Compilers oder der Laufzeitumgebung als Klassenpfad übergeben können. In diesem Abschnitt geht es nun darum, den Archiven die zusätzliche Eigenschaft zu geben, dass der Anwender sie per Doppelklick aufrufen und damit das enthaltene Java-Programm ausführen kann, ohne dass dazu Befehle in der Shell eingegeben werden müssen. Dazu generieren Sie nun zunächst eine eigene Archiv-Datei.

JAR-Archive können mit dem Kommandozeilen-Tool `jar` erzeugt werden. Als einfaches Beispiel wird im Folgenden die Klasse `HalloWelt` aus Kapitel 1, *Grundlagen*, als Archiv verpackt. Die Datei `HalloWelt.class` muss sich dazu im aktuellen Arbeitsverzeichnis befinden, wo dann auch das JAR-Archiv gespeichert wird.

```
[straylight:~] much% jar cvf MeinArchiv.jar HalloWelt.class
Manifest wurde hinzugefügt.
Hinzufügen von: HalloWelt.class(ein = 423)(aus= 287)
(komprimiert 32%)
```

Nach dem Befehl `jar` werden die Optionen `cvf` übergeben. Das `c` steht für »create«, es soll ein neues Archiv erzeugt werden. `f` lässt Sie den Namen des Archivs, hier `MeinArchiv.jar`, festlegen. Am Ende werden alle Klassen und sonstigen Dateien aufgeführt, die in das Archiv gepackt werden sollen. Hier ist nur eine einzelne Klasse aufgeführt, aber Sie können genauso gut mehrere

Dateien (durch Leerzeichen getrennt) oder beispielsweise `*.class` übergeben. Es gibt zahlreiche weitere Optionen, unter anderem zum Hinzufügen ganzer Verzeichnisse – alle Möglichkeiten bekommen Sie mit `jar` oder wie üblich mit `man jar` aufgelistet.

Nachdem das Archiv erzeugt wurde, können Sie sich den Inhalt wieder mit `jar`, diesmal aber mit der Option `t`, anzeigen lassen.

```
[straylight:~] much% jar tf MeinArchiv.jar
META-INF/
META-INF/MANIFEST.MF
HalloWelt.class
```

JAR-Archive sind im Wesentlichen nichts anderes als die Ihnen sicherlich bekannten ZIP-Archive, in denen nicht nur die eigentlichen Dateien gespeichert werden, sondern diese auch in Verzeichnisse (und Unterverzeichnisse) aufgeteilt sein können. Bei diesem Beispiel ergibt sich also folgende Verzeichnisstruktur:

```
MeinArchiv.jar
  META-INF
    MANIFEST.MF
  HalloWelt.class
```

Das Verzeichnis `META-INF` wird vom `jar`-Tool automatisch erzeugt, ebenso die darin enthaltene Datei `MANIFEST.MF`, das so genannte **Manifest**. Ändern Sie diese Struktur oder Schreibweise niemals ab! Die Dateien in diesem Verzeichnis beschreiben die weiteren Inhalte des JAR-Archivs. Sie gehören nicht zu den eigentlichen Programmdateien und werden im Allgemeinen maschinell erzeugt. Auch wenn Sie hier üblicherweise nur das Manifest finden, können beispielsweise auch `Index`-Dateien auftauchen. Mit dem Inhalt des Manifests beschäftigen wir uns gleich noch ausführlicher.

Im Wurzelverzeichnis taucht die Klasse `HalloWelt` auf. Diese Klasse ist keinem speziellen Paket zugeordnet und gehört somit zum anonymen Paket. Ansonsten würden an dieser Stelle die Pakethierarchien mit den entsprechenden Unterverzeichnissen stehen. Für andere Ressourcen (Bilder, Sounds) dürfen Sie das Wurzelverzeichnis oder beliebige Unterverzeichnisse wählen.³

Nun haben Sie also ein selbst gemachtes JAR-Archiv, das Sie wie die bisherigen `jar`-Dateien auch schon mit `java -cp MeinArchiv.jar HalloWelt` ausführen können. Ziel war aber der Start des Archivs per Doppelklick im Finder. Wenn

³ Wenn Sie solche Ressourcen in Ihrem Programm laden wollen, sehen Sie sich die Methoden `getResource` und `getResourceAsStream` in der Java-API-Dokumentation an.

Sie dies nun ausprobieren, passiert je nach Systemversion und installiertem JDK entweder leider gar nichts oder es wird ein Fehlerdialog angezeigt. In der Konsole (nicht im Terminal!) finden Sie aber eine Fehlermeldung, die in etwa wie folgt aussieht:

```
Failed to load Main-Class manifest attribute from
/Users/much/MeinArchiv.jar
```

Damit ein JAR-Archiv direkt gestartet werden kann, müssen Sie im Manifest angeben, welche Klasse die `main`-Methode enthält. Erzeugen Sie dazu eine Textdatei mit dem Namen `MeinManifest`, die folgende Zeile enthält:

```
Main-Class: HalloWelt
```

In einem Manifest kann durchaus mehr als eine Zeile stehen, aber meistens ist dieser `Main-Class`-Eintrag (man spricht auch von `Main-Class-Attribut`) vollkommen ausreichend.

Achten Sie darauf, dass sich am Ende mindestens ein Zeilenumbruch befindet und dass die Zeilenenden im DOS- oder UNIX-Format gespeichert sind! Die JAR-Spezifikation sieht zwar eigentlich auch Mac-Zeilenenden im Manifest vor, aber in der Praxis kann das zu Problemen führen ... Sie können sich Ihr Manifest mit `more MeinManifest` ausgeben lassen. Wenn Ihnen dann

```
Main-Class: HalloWelt^M^M
```

angezeigt wird, müssen Sie die Zeilenenden umwandeln (^M ist das Steuerzeichen für Mac-Zeilenenden).

Jetzt können Sie dem JAR-Archiv Ihr neues Manifest hinzufügen. Dazu verwenden Sie die `jar`-Optionen `u` (»update«) und `m` und übergeben vor dem Archivnamen die Manifest-Datei:

```
jar uvmf MeinManifest MeinArchiv.jar
```

Das so geänderte Manifest können Sie mit

```
jar xf MeinArchiv.jar META-INF/MANIFEST.MF
```

auslesen und erhalten dann eine Textdatei mit folgendem Inhalt:

```
Manifest-Version: 1.0
Created-By: 1.4.2_03 (Apple Computer, Inc.)
Main-Class: HalloWelt
```

Zusätzlich zu Ihrer Manifest-Zeile wurden also noch optionale Versions- und Herstellerinformationen eingefügt. Wenn Sie das Archiv nun doppelt ankli-

cken, wird die `main`-Methode in der Klasse `HalloWelt` ausgeführt! Die Ausgabe erfolgt wie im Fehlerfall nicht im Terminal, sondern in der Konsole. Das im vorherigen Abschnitt verwendete Archiv `Java2D.jar` lässt sich übrigens auch direkt per Doppelklick starten, da in seinem Manifest die Zeile `Main-Class: java2d.Java2Demo` eingetragen ist.

Ein JAR-Archiv mit einem solchen Manifest können Sie in der Shell verkürzt aufrufen, indem Sie bei `java` die Option `-jar` anstelle von `-classpath` einsetzen. Sie sparen sich in diesem Fall also die Angabe der aufzurufenden Klasse:

```
java -jar MeinArchiv.jar
```

Weitere Informationen zum Erzeugen und Verwenden von JAR-Archiven finden Sie bei Sun auf der Seite <http://java.sun.com/docs/books/tutorial/jar/>.

Wenn Sie sich ein JAR-Archiv im Datei-Informationsdialog ansehen, stellen Sie fest, dass es automatisch dem Programm `Jar Launcher` zugeordnet wurde (siehe Abbildung 4.5). Dieses Programm ist die Systemkomponente von Mac OS X, die für das Ausführen von JAR-Archiven zuständig ist. Sie finden das Programm im Verzeichnis `/System/Library/CoreServices/`.



Abbildung 4.5 JAR-Archive sind automatisch dem Jar Launcher zugeordnet.

`Jar Launcher` verwendet immer das neueste installierte JDK, also zum Beispiel Version 1.4.2. Wenn Ihre Java-Applikation eine bestimmte (ältere) Java-Version benötigt, müssen Sie eine Mac OS X-Applikation generieren (siehe folgender Abschnitt), dort ist das dann konfigurierbar.

Seit Mac OS X 10.3 kann der `Jar Launcher` auch einzelne `class`-Dateien ausführen, wenn diese eine `main`-Methode enthalten (probieren Sie das einfach mal aus, indem Sie `HalloWelt.class` doppelt anklicken).

Richtig funktionieren wird dies aber nur dann, wenn Ihr Programm nur aus dieser einen Klasse besteht oder aber nur auf Bibliotheken in den Standard-Erweiterungsverzeichnissen zugreift. Weitere Klassen im Arbeitsverzeichnis werden dagegen nicht gefunden, da dieses Verzeichnis nicht dem Klassenpfad hinzugefügt wird – insofern sind die Einsatzmöglichkeiten dieses Features relativ beschränkt.

Unter neueren Windows-Versionen sind JAR-Archive nach der Java-Installation automatisch dem Programm `javaw.exe` zugeordnet, das Starten per Doppelklick funktioniert damit also auch dort. Allerdings wurde `javaw` speziell dafür geschrieben, um das Fenster der DOS-Eingabeaufforderung zu unterdrücken, Textausgaben werden dann also nirgends angezeigt! Falls Ihre Anwendung dem Benutzer Konsolenausgaben anzeigen muss, sollten Sie speziell für Windows eine Stapelverarbeitungsdatei als Start-Skript beilegen, die nicht auf `javaw` zurückgreift, sondern ganz normal `java` aufruft.

In Kapitel 2, *Entwicklungsumgebungen*, haben Sie bereits gesehen, wie Sie mit Apples Entwicklungsumgebung Xcode Java-Projekte erstellen und verwalten. Standardmäßig erzeugt Xcode bei allen Java-Projekttypen JAR-Archive als »Product«. Sollten Sie stattdessen einzelne Klassen benötigen, können Sie unter »Java Archive Settings« den »Product type« auf »Class Hierarchy« umstellen (siehe Abbildung 4.6). Belassen Sie es im Zweifel aber besser bei der Voreinstellung. Und wenn Sie beim Neuanlegen eines Projekts den Typ »Java Tool« wählen, bekommen Sie nicht nur ein JAR-Archiv erzeugt, Xcode generiert auch gleich ein passendes Manifest, in dem das `Main-Class`-Attribut eingetragen ist!

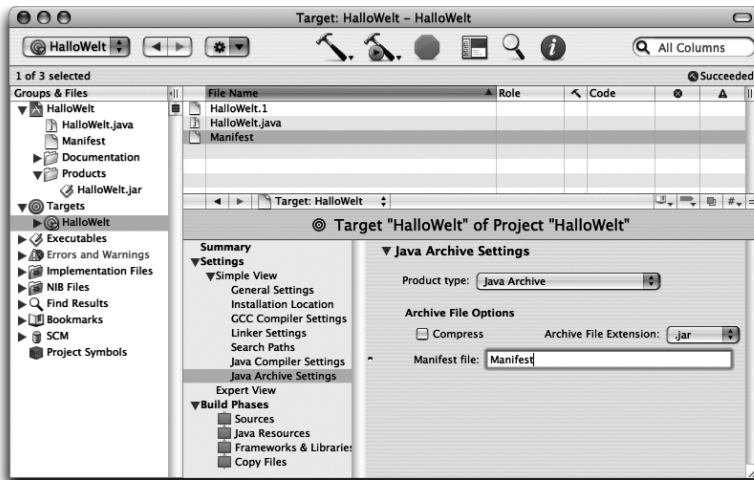


Abbildung 4.6 Xcode erzeugt doppelklickbare JAR-Archive.

Doppelklickbare JAR-Archive funktionieren soweit gut, haben aber auch gewisse Nachteile. Unter anderem können Sie kein spezielles Programmsymbol im Archiv ablegen, und als Programmname wird immer der Name der aufgerufenen Klasse verwendet. Damit Sie dies nicht wieder mit einem Shell-Kommando konfigurieren müssen, erstellen Sie aus dem JAR-Archiv nun eine richtige Mac OS X-Applikation.

4.3 Mac OS X-Applikationen

Mac OS X-Applikationen tragen die Dateinamenserweiterung `.app` (auch wenn diese normalerweise ausgeblendet ist) und erfüllen endlich alle Wünsche, welche die bisherigen Lösungen nur zum Teil realisiert haben:

- ▶ Die Applikation ist durch einen Doppelklick direkt aus dem Finder heraus startbar.
- ▶ Die Anwendung hat ein passendes Programmsymbol.
- ▶ Der Benutzer sieht nur eine einzige Programmdatei, egal aus wieviel JAR-Archiven, JNI-Bibliotheken und sonstigen Ressourcen die Anwendung besteht. Dadurch kann das Programm bequem auf die Festplatte kopiert und gegebenenfalls wieder gelöscht werden.

Apropos löschen: Beim Mac gibt es kein Systemprogramm zum Deinstallieren von Anwendungen. Stattdessen löscht der Benutzer einfach nur die Programmdatei und eventuell noch die Konfigurationsdaten in `~/Library/Preferences/`.

- ▶ Das Programm ist optimal an das System angepasst, ohne dass irgendwelche Kommandozeilenparameter gesetzt werden müssen.

Zusätzlich können Mac OS X-Applikationen besser in den Finder integriert werden als reine Java-Anwendungen. Wenn Sie die Applikation mit bestimmten Dokumenttypen assoziieren, kann der Benutzer das Programm einfach durch Doppelklick auf ein passendes Dokument starten – das dann auch gleich geladen wird. Ebenso taucht die Anwendung dann im »Öffnen mit«-Menü auf, das Sie im Kontext-Popup-Menü jedes Dokuments finden.

Allerdings sind die Applikationen nicht als eine einzige Binärdatei gespeichert, sondern sie bestehen aus einer **Verzeichnisstruktur** mit vorgegebenem Aufbau. Das oberste Verzeichnis, das wie die Anwendung heißt und die Namenserverweiterung `.app` besitzt, bekommt das so genannte Paket-Bit gesetzt, wodurch die Ordnerstruktur im Finder wie eine einzige Datei aussieht. Mac OS X-Applikationen werden daher auch **Programmpakete** oder *Bundles* (genauer »Mac OS X Application Bundles«) genannt. Verwechseln Sie Programmpakete nicht mit Java-Paketen (Packages), das sind zwei komplett verschiedene Konzepte!

Der Vorteil der Verzeichnisstruktur ist, dass sie zwischen verschiedenen Betriebssystemen hin- und herkopiert werden kann (zum Beispiel wenn Sie sich eine Mac-Anwendung von einem Linux-Web-Server herunterladen), ohne dass Daten verloren gehen. Falls das für Sie selbstverständlich klingt: Beim alten Mac OS bestanden alle Dateien – und damit auch Programme – aus einem *Data Fork* (Benutzerdaten, z.B. Bilder, Sounds, Texte) und einem *Resource Fork* (Programmcode, Zeichenketten, Icons).⁴ Wenn man Dateien vor dem Transport über das Internet oder mit DOS-Disketten nicht in ein spezielles Format (.bin oder .hqx) gebracht hatte, ging der Resource Fork verloren, und Programme konnten dann nicht mehr ausgeführt werden.

4.3.1 Programmsymbole

Als Erstes benötigen Sie ein geeignetes Programmsymbol (Icon) für Ihre Applikation. Mac OS X erwartet diese Bildchen in speziellen Icon-Ressource-Dateien mit der Dateinamenserweiterung `.icns`. Darin wird das jeweilige Symbol in verschiedenen Auflösungen gespeichert, und das System sucht sich dann automatisch die passende Größe heraus – je nachdem, wie groß das Symbol gezeichnet werden soll (beispielsweise im Finder oder im Dock).

Zum Erzeugen von `icns`-Dateien stellt Apple das Programm `Icon Composer` zur Verfügung, das Sie im Verzeichnis `/Developer/Applications/Utilities/` finden. Als Ausgangsbild suchen Sie sich irgendeine Bilddatei in einem gängigen Format (GIF, PNG, TIFF, JPEG, PDF, PICT etc.), zum Beispiel Suns Java-Maskottchen »Duke«: <http://java.sun.com/images/duke.wave.shadow.gif>.⁵ Bringen Sie das Bild mit einer Bildbearbeitungssoftware (GraphicConverter⁶ oder Photoshop) durch Anfügen von leerem Rand auf quadratische Größe, und setzen Sie dann die Hintergrundfarbe auf transparent. Auf der CD finden Sie im Verzeichnis `/examples/ch04/` die Datei `duke.gif`, die bereits passend bearbeitet wurde.

Starten Sie nun den `Icon Composer` und ziehen Sie Ihre Bilddatei nacheinander auf alle vier Felder in der Spalte »Image RGB/Alpha« (siehe Abbildung 4.7). Wenn das Programm Sie fragt, ob das Bild passend skaliert werden soll, bestätigen Sie dies. Lassen Sie ebenso eine passende 1-Bit-Maske automatisch extrahieren. Die generierte 1-Bit-Maske wird dann in der Spalte »Hit Mask« dargestellt. Mac OS X ermittelt anhand dieser Maske, welche Teile des Bildes angeklickt werden können (schwarz) und welche Teile transparent dargestellt

⁴ <http://developer.apple.com/documentation/mac/MoreToolbox/MoreToolbox-11.html>

⁵ Oder suchen Sie sich auf http://java.sun.com/features/1999/05/duke_gallery.html ein anderes Duke-Bild aus!

⁶ Download von <http://www.lemkesoft.de/de/graphcon.htm>

werden (weiß). Wenn Sie in Ihrem Bild Transparenzinformationen gespeichert haben (im GIF wie oben beschrieben oder als Alpha-Kanal in einer PNG-Datei), kennt der `Icon Composer` bereits eine perfekte Hit Mask, nämlich genau alle nicht transparenten Stellen! Fehlt die Transparenzinformation, versucht das Programm, in den Bilddaten einen geeigneten Umriss zu erkennen, aber meistens wird dann eine Nachbearbeitung nötig sein.

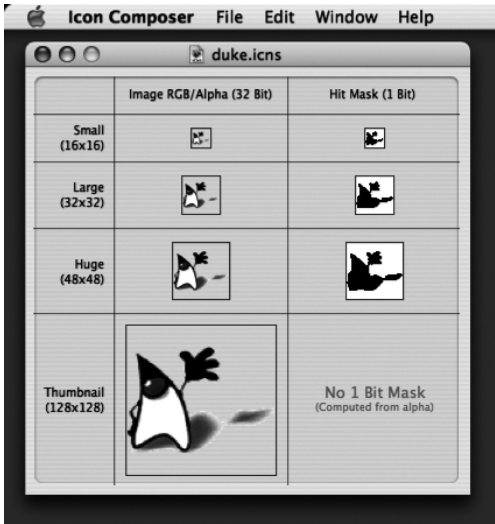


Abbildung 4.7 Icon Composer

Schließlich speichern Sie Ihre Symboldatei als `duke.icns` ab – fertig! Die Icons können nun zusammen mit einem JAR-Archiv zu einer Mac OS X-Applikation verarbeitet werden.

Für Icon-Dateien bringt Mac OS X einige weitere Hilfsprogramme mit:

- ▶ `/Developer/Applications/Utilities/icns Browser`
Der `icns Browser` zeigt alle möglichen Größen an, in denen Bild- und Maskendaten in einer `icns`-Datei gespeichert sein können. Wenn Sie Ihre gerade gespeicherte Datei auf den Browser ziehen, werden Sie feststellen, dass bei weitem nicht alle Felder belegt sind. Das ist absolut normal, denn alle übrigen Größen und Farbtiefen werden automatisch berechnet.
- ▶ `/Developer/Applications/Graphics Tools/Pixie`
Pixie ist eigentlich eine Bildschirmlupe, mit der man fertige Icons pixelgenau überprüfen kann. Sie können damit aber auch prima vergrößerte Screenshots herstellen!

► /Programme/Vorschau

Wenn Sie eine `icns`-Datei doppelt anklicken, wird normalerweise die Vorschau gestartet, mit der Sie auch viele Bildformate und PDF-Dokumente betrachten können. Bei Icon-Dateien wird Ihnen das Symbol in allen gespeicherten Größen angezeigt.

Wenn Sie auf einem anderen Betriebssystem ein MacOS X-Icon erzeugen möchten, finden Sie dafür derzeit leider noch keine passenden Werkzeuge. Anstelle einer `icns`-Datei verwenden Sie dann am einfachsten eine ganz normale Bilddatei in einem der gängigen Formate – MacOS X kann diese auch für die Symboldarstellung verwenden. Wirklich gute Resultate erzielen Sie aber nur mit dem speziellen `icns`-Format.

4.3.2 Programmpakete

Damit Sie das Erzeugen eigener Programmpakete besser verstehen, sehen Sie sich zunächst ein fertiges Paket an. Nehmen Sie dazu das Programm Java 1.4.2 Plugin Einstellungen (oder eine neuere Version) im Ordner `/Programme/Dienstprogramme/Java/`. Klicken Sie das Programmsymbol mit `[Ctrl]+Klick` oder mit der rechten Maustaste an und wählen Sie im erscheinenden Kontext-Popup-Menü »Paketinhalt zeigen« aus.



Abbildung 4.8 Paketinhalt zeigen

Je nachdem, ob Sie im »Darstellung«-Menü des Finders »Als Symbole« oder »Als Liste« gewählt haben, sehen Sie nun einen einzigen Ordner `Contents`, denn Sie dann bitte öffnen. Oder Sie sehen gleich eine Verzeichnishierarchie wie in Abbildung 4.9 dargestellt.

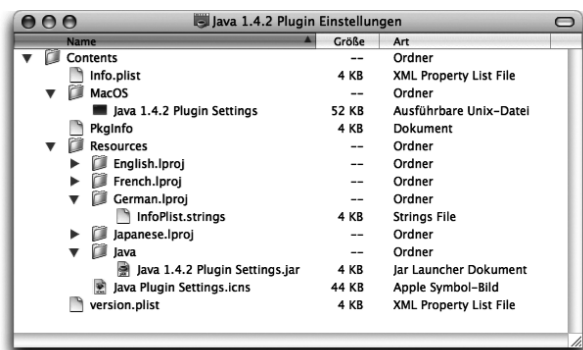


Abbildung 4.9 Inhalt des Programmpakets

Aus folgenden wesentlichen Komponenten setzt sich das Programmpaket zusammen:

► Programmcode

Das Verzeichnis `Contents/Resources/Java/` enthält den Java-Code der Anwendung in Form von `class`-Dateien oder – vorzugsweise – JAR-Archiven. JNI-Bibliotheken können hier ebenfalls platziert werden.

► Icon-Datei

Eine `icns`-Datei, wie sie im vorhergehenden Abschnitt vorgestellt wurde, liegt im Verzeichnis `Contents/Resources/`.

► Ausführbare Programmdatei

Wenn der Benutzer ein Programmpaket doppelt anklickt, wird nicht mehr der Jar Launcher aufgerufen, um das JAR-Archiv auszuführen, sondern es wird ein kleines natives Programm innerhalb des Pakets gestartet. Dieses Programm, auch *Stub* oder *Java Application Stub* genannt, ruft dann Ihren Java-Code auf. Apple stellt diesen Stub, den Sie im Bild als »Ausführbare UNIX-Datei« sehen, fertig kompiliert zur Verfügung. Abgelegt wird die Programmdatei im Verzeichnis `Contents/MacOS/` (ohne Leerzeichen, obwohl sich das »Mac OS« mit Leerzeichen schreibt).

► Info.plist

In der Datei `Contents/Info.plist` werden generell Versionsangaben und Informationstexte zum Programmpaket gespeichert. Bei Java-Applikationen kommen hier noch Konfigurationsdaten für die Java-Laufzeitumgebung hinzu. Der genaue Aufbau dieser Datei wird im Abschnitt »Programmpakete von Hand erzeugen« beschrieben.

► PkgInfo

`Contents/PkgInfo` speichert ebenfalls Informationen über das Paket, und zwar den so genannten *Type*- und *Creator*-Code. Die Datei ist exakt acht Zei-

chen lang und enthält bei Java-Applikationen oft die Zeichenkette `APPL????`. Eigentlich ist `PkgInfo` optional, da sie redundante Informationen aus `Info.plist` enthält, aber das Weglassen kann zu Problemen führen. Daher: Verwenden Sie diese Datei immer, und achten Sie darauf, dass die zweimal vier Zeichen mit den Einträgen `CFBundlePackageType` und `CFBundleSignature` aus `Info.plist` übereinstimmen! Mehr Informationen zum Type und Creator stehen im folgenden Abschnitt »Jar Bundler«.

► Lokalisierungen

Optional kann ein Programmpaket beliebige Lokalisierungen (Übersetzungen) der Zeichenketten enthalten. Dazu fügen Sie zusätzlich zum Ordner `English.lproj`, der immer vorhanden sein muss, weitere Ordner mit den jeweiligen (englischen) Sprachnamen hinzu. In den Ordnern kann dann eine Datei `InfoPlist.strings` liegen, in der die Zeichenketten aus `Info.plist` übersetzt vorliegen. Beachten Sie, dass es sich um Unicode-Texte im UTF-16-Format handelt! Sie müssen zur Bearbeitung also einen geeigneten Editor, beispielsweise `/Programme/TextEdit`, verwenden.

Wichtig: Selbst wenn Sie die Lokalisierung des Java-Programms über `JavaResourceBundles`⁷ lösen, müssen Sie im Programmpaket zumindest einen passenden leeren Ordner – z.B. `German.lproj` – anlegen, damit Mac OS X die gewünschte Umgebung (Tastaturbelegung, Standardtexte im Apfel- und Programm-Menü) korrekt einstellt.

Je nach Applikation können noch weitere Dateien im Programmpaket auftauchen. Hier sehen Sie beispielsweise die Textdatei `version.plist`, die von `/Developer/Applications/Utilities/PackageMaker` ausgewertet wird, um Versionierungsinformationen für Installationspakete zu erstellen. Solche Pakete können dann vom Anwender bequem mit dem Installationsprogramm aus `/Programme/Dienstprogramme/` installiert werden. Bei Ihren Anwendungen ist so eine Datei zunächst nicht erforderlich.

Wenn Sie sich das Programmpaket einer älteren Mac OS X-Java-Applikation ansehen, taucht dort häufig noch eine Datei `MRJApp.properties` auf. Bei Mac OS X 10.0 wurden die Paket-Versionsinformationen und die Java-Konfigurationsdaten noch getrennt in den Dateien `Info.plist` und `MRJApp.properties` gespeichert. Obwohl diese Konfigurationsmethode zum Teil auch heute noch funktioniert, empfiehlt Apple seit Mac OS X 10.1, alle Daten in `Info.plist` abzulegen (siehe Abschnitt 4.3.5) und auf `MRJApp.properties` komplett zu verzichten.

⁷ `java.util.ResourceBundle` – Lokalisierungen werden bei Java unter dem Stichwort »Internationalisierung« oder »I18N« behandelt.

Sie haben nun verschiedene Möglichkeiten, eigene Programmpakete zu erzeugen: Halbautomatisch mit dem `Jar Bundler`, vollautomatisch mit einer geeigneten Entwicklungsumgebung oder komplett von Hand.

4.3.3 Jar Bundler

Im Verzeichnis `/Developer/Applications/Java Tools/` wird von Apples Entwicklerwerkzeugen das Programm `Jar Bundler` installiert, mit dem Sie bestehende JAR-Archive als `Mac OS X`-Applikation, also als Programmpaket, verpacken können.

Falls es dieses Programm bei Ihnen nicht gibt und Sie stattdessen im Verzeichnis `/Developer/Applications/` die Applikation `MRJAppBuilder` finden, haben Sie noch die ursprünglichen `Mac OS X 10.2`-Entwicklerwerkzeuge installiert. `MRJAppBuilder`, der Vorgänger vom `Jar Bundler`, sollte nicht mehr zur Programmgenerierung eingesetzt werden. Bitte bringen Sie Ihr Java auf eine aktuelle 1.4.x-Version und installieren Sie die dazugehörigen Entwickler-Tools.

Als Beispiel sehen Sie im Folgenden, wie aus `Suns Java2Demo`, das Sie schon am Anfang des Kapitels verwendet haben, ein Programmpaket erzeugt wird. Starten Sie dazu den `Jar Bundler`. Es geht ein Fenster auf, in dem die Dialogseite »Build Information« aktiv ist (siehe Abbildung 4.10).

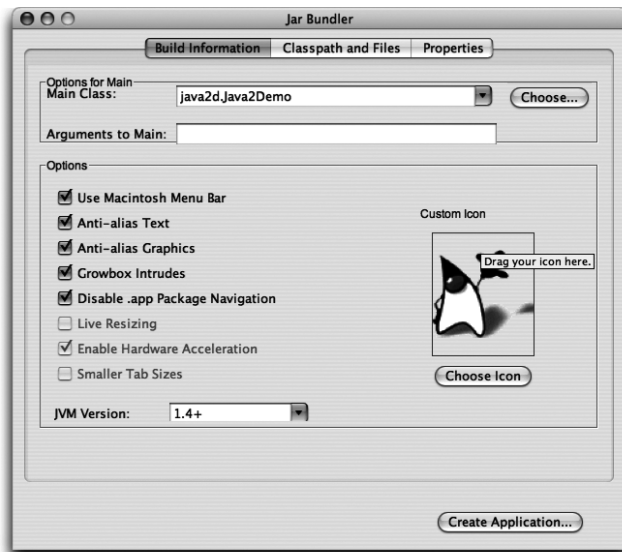


Abbildung 4.10 Jar Bundler »Build Information«

Geben Sie bei »Main Class« die Klasse mit der `main`-Methode ein, hier `java2d.Java2Demo`. Wenn Sie den vollqualifizierten Namen nicht kennen, sehen Sie in der Programmdokumentation nach oder extrahieren Sie das Manifest aus dem JAR-Archiv. Wenn Ihr Programm Argumente erwartet, können Sie diese bei »Arguments to Main« eintragen. Das Format entspricht dem von Kommandozeilenparametern (Zeichenketten, die durch Leerzeichen getrennt sind).

Ziehen Sie dann die Icon-Datei `duke.icns` auf die Bildmulde unterhalb von »Custom Icon«. `duke.icns` finden Sie auf der CD im Verzeichnis `/examples/ch04/`, falls Sie sich die Datei nicht selbst erzeugt haben.

Als Nächstes legen Sie fest, mit welcher Java-Version das Programm ausgeführt werden soll. Dazu wählen Sie bei »JVM Version« eine geeignete Version aus oder tragen von Hand eine andere passende Zeichenkette ein. Die Versionsangaben bedeuten Folgendes, sofern Sie Java 1.4.2 als aktuellste Version installiert haben:

Zeichenkette	Verwendete Java-Version	Anmerkungen
1.3.1	1.3.1	Verwendet exakt Version 1.3.1
1.3*	1.3.1 oder eine neuere 1.3.x	Verwendet die aktuellste 1.3-Version
1.3+	1.4.2 oder irgendeine neuere	Verwendet die aktuellste Version, mindestens aber 1.3
1.4.1	1.4.2	Da Mac OS X derzeit nur jeweils ein JDK in den Hauptversionen (1.3 und 1.4) verwalten kann, wird das JDK 1.4.1 bei der Installation vom JDK 1.4.2 gelöscht! Aus Kompatibilitätsgründen wird daher bei der Angabe von Version 1.4.1 das aktuellere 1.4.2 verwendet.
1.4.2	1.4.2	Verwendet exakt Version 1.4.2
1.4*	1.4.2 oder eine neuere 1.4.x	Verwendet die aktuellste 1.4-Version
1.4+	1.4.2 oder irgendeine neuere	Verwendet die aktuellste Version, mindestens aber 1.4

Tabelle 4.1 Festlegen der benötigten Java-Version

Verwenden Sie am besten keine speziellen Versionen (z.B. »1.4.1«), da es sein kann, dass diese Version gar nicht (mehr) vorhanden ist. Mit den flexiblen Angaben (z.B. »1.3+«) sind Sie dagegen auf der sicheren Seite.

Die flexiblen Angaben mit + und * können Sie nur auf Hauptversionen verwenden, also beispielsweise »1.3+« oder »1.3*«. Angaben wie »1.3.1+« oder »1.4.1+« sind **falsch!** Leider gab es in der Vergangenheit auch Programmierwerkzeuge, die solche falschen Zeichenketten automatisch erzeugt haben.

Wenn Sie keine oder eine fehlerhafte Versionsangabe machen, wird Ihre Applikation mit Java 1.3.1 gestartet. Verlassen Sie sich aber nicht darauf, denn eine zukünftige Mac OS X-Version könnte hierfür eine andere Version wählen.

Nun können Sie noch einige Optionen auswählen, von denen folgende sowohl für Java 1.3 als auch für Java 1.4 angeboten werden:

► **Use Macintosh Menu Bar**

Legt fest, ob `JMenuBar`s von `JFrames` die Mac-typische Menüzeile oben am Bildschirm verwenden oder ob sie Swing-typisch im jeweiligen Fenster angezeigt werden. Sie sollten diese Option nach Möglichkeit immer einschalten.

► **Anti-alias Text, Anti-alias Graphics**

Legt fest, ob Texte und/oder Grafikausgaben (Linien, Kreise usw.) mit Anti-Aliasing, also mit weichgezeichneten Rändern, ausgegeben werden. Beide Optionen sollten aktiv sein.

► **Growbox Intrudes**

Mac OS X-Fenster stellen unten rechts normalerweise einen »Greifer« dar, mit dem die Fenstergröße geändert werden kann. Java-Fenster können zwar meistens auch in der Größe geändert werden, zeigen das im Aqua-Aussehen aber nicht an. Mit dieser Option blenden Sie auch in Aqua-Java-Fenstern den »Greifer« ein. Testen Sie dann aber Ihre Applikation, ob dadurch irgendwelche Dialogkomponenten gestört (übermalt) werden.

► **Disable .app Package Navigation**

Bestimmt, ob die Java-Dateiauswahl-Dialoge Mac OS X-Applikationen als eine Datei oder als Verzeichnisstruktur zeigen. Wenn Sie nicht gerade ein Programmierwerkzeug schreiben, mit dem man Applikationen »von innen« sehen können muss, sollten Sie diese Option einschalten.

Folgende Optionen stehen nur für Java 1.3 zur Verfügung:

► **Live Resizing**

Legt fest, ob während einer Größenänderung des Fensters der Inhalt sofort neu gezeichnet wird oder erst nach Abschluss der Änderung. Schalten Sie diese Option nur ein, wenn Ihre Dialoge nicht zu komplex sind, damit die Größenänderung nicht zu ruckelig wirkt. Mit Java 1.4 wird immer eine Echtzeit-Größenänderung durchgeführt.

► **Enable Hardware Acceleration**

Sorgt dafür, dass Swing-Oberflächen und Java2D-Grafiken direkt von der Grafikkarte dargestellt werden und nicht erst vom Hauptprozessor berechnet werden müssen. Wenn Sie Mac OS X 10.2 oder neuer voraussetzen können, sollten Sie diese Option einschalten. Bei Java 1.4 wird die Grafikbeschleunigung von Apples »Quartz Extreme«-Grafikkomponente durchgeführt, ohne dass Sie sich als Entwickler darum kümmern müssen.

► **Smaller Tab Sizes**

Tabs werden normalerweise in der Mac-typischen Größe angezeigt (wie Sie das beispielsweise oben im Jar Bundler-Dialog sehen können). Wenn Sie diese Option einschalten, werden Tabs etwas kleiner dargestellt, um besser zum Swing-Metal-Aussehen zu passen.

Wechseln Sie dann zur Dialogseite »Classpath and Files« (siehe Abbildung 4.11). Hier legen Sie fest, aus welchen Code-Ressourcen sich Ihre Applikation zusammensetzt, welche Archive und Bibliotheken also in das Programmpaket kopiert und welche Komponenten außerhalb des Pakets auf den Klassenpfad gesetzt werden sollen.

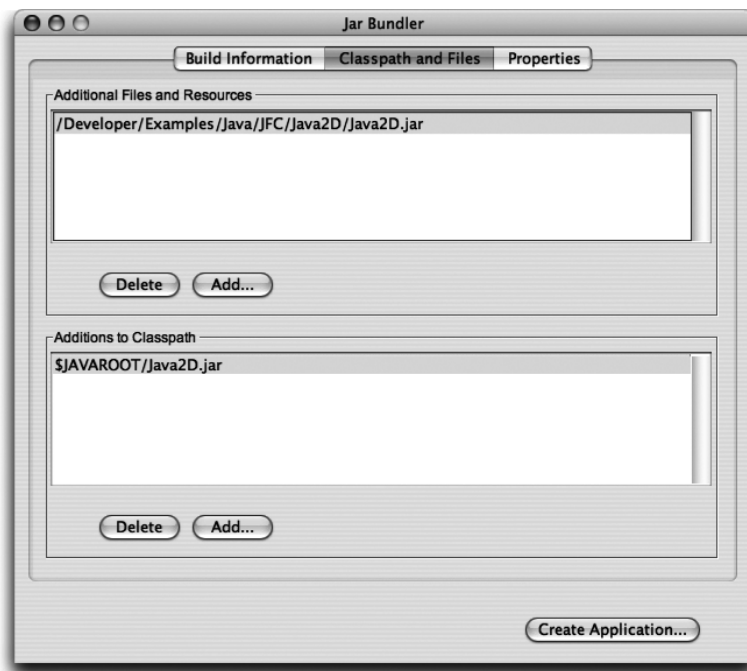


Abbildung 4.11 Jar Bundler »Classpath and Files«

Ziehen Sie dazu das Archiv `Java2D.jar` aus dem Verzeichnis `/Developer/Examples/Java/JFC/Java2D/` in den Bereich »Additional Files and Resources«. Die Datei wird dort eingetragen und damit später in das Programmpaket kopiert. Zusätzlich wird ein Eintrag in den Bereich »Additions to Classpath« eingefügt, damit Ihre Applikation auch wirklich auf die Klassen des Archivs zugreifen kann.

Sie sehen, dass der `Jar Bundler` beim Klassenpfad die Variable `$JAVAROOT` verwendet. Wenn Sie Ihr Programmpaket als `Xyz.app` speichern, ist diese Variable eine bequeme Abkürzung für das Verzeichnis `Xyz.app/Contents/Resources/Java/`. Wie Sie weiter oben schon gesehen haben, liegen JAR-Archive innerhalb eines Programmpakets genau in diesem Verzeichnis! Es gibt noch eine weitere Variable, die Sie in den Klassenpfaden als Abkürzung verwenden können: `$APP_PACKAGE`. Diese Variable bezeichnet das Paketverzeichnis selbst, bei diesem Beispiel also `Xyz.app/` (allerdings ohne den Schrägstrich). Beide Variablen sind nur innerhalb des Programmpakets (und dort sogar nur in einem bestimmten Abschnitt der Datei `Info.plist`) gültig – verwenden Sie sie also niemals im Java-Code oder anderswo in Konfigurationsdateien!

Wenn Sie also ein JAR-Archiv in Ihrer Applikation nutzen wollen, das nicht in das Programmpaket kopiert werden soll, fügen Sie die Pfadangabe des Archivs nur bei »Additions to Classpath« ein. Handelt es sich dabei nicht um eine absolute, sondern um eine relative Pfadangabe, beispielsweise weil das Archiv immer im selben Verzeichnis wie das Programmpaket liegt, müssen Sie den Pfad noch von Hand ändern. Nach einem Doppelklick auf die Textzeile kann diese bearbeitet werden. Um auf `MeinTollesArchiv.jar` im Verzeichnis der Applikation zuzugreifen, müssten Sie dort `$APP_PACKAGE/./MeinTollesArchiv.jar` eintragen.

Die Dialogseite »Properties« schließlich legt unter anderem Versionsnummern und den Programmnamen fest (siehe Abbildung 4.12). Die bunt verstreuten Optionen lassen sich grob in drei Bereiche einteilen: Versions- und Typangaben, Parameter für die Java-Laufzeitumgebung sowie weitere Konfigurationsdaten für das Programmpaket.

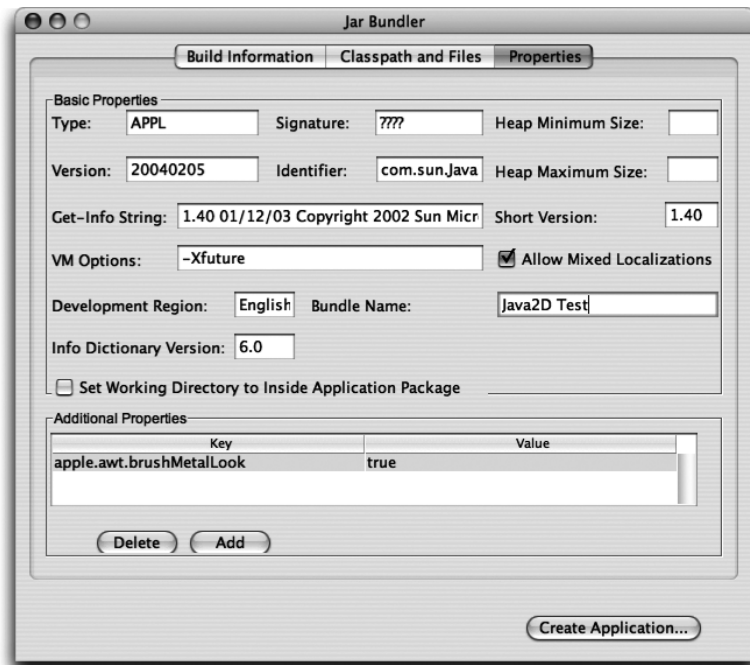


Abbildung 4.12 Jar Bundler »Properties«

Name, Version, Typ

► Bundle Name

Hiermit legen Sie den Namen für das Programm-Menü und den »Über...«-Menüeintrag fest. Denken Sie daran, dass diese Zeichenkette nicht länger als 16 Zeichen sein sollte.

► Get-Info String

Der Datei-Informationsdialog zeigt in der »Version«-Zeile einen Versions- und Copyright-Eintrag an, den Sie hier eintragen sollten. Im Beispiel wurde der Copyright-Hinweis aus den Sun-Quelltexten verwendet.

► Short Version

Diese Zeichenkette enthält nur die Versionsnummer ohne weitere Texte. Dieses Feld wird häufiger frei gelassen, aber es schadet auch nicht, wenn Sie es setzen.

► Version

»Version« gibt die Build-Nummer der Software an (im Beispiel wurde einfach das Tagesdatum verwendet). Häufig wird hier auch einfach nur die Versionsnummer eingetragen, zusätzlich zu oder als Ersatz von »Short Version«.

► Type, Signature

Diese beiden Felder haben Sie im vorangegangenen Abschnitt schon als *Type* und *Creator* kennen gelernt. »Type« legt fest, um welche Art von Datei es sich handelt: `TEXT` für Texte, `MP3` für MP3-Dateien. Bei ausführbaren Applikationen ist dies immer `APPL`.

In vielen Dokumentationen finden Sie den Typ für Applikationen mit `AAPL` angegeben. Das ist leider ein klassischer **Tippfehler**, denn dabei handelt es sich um Apples Börsensymbol an der NASDAQ – der Applikationstyp dagegen war und ist `APPL`.

Mit »Signature« legen Sie fest, welches Programm die Datei erzeugt hat (daher der alte Name »Creator«), oder besser, mit welchem Programm die Datei vorzugsweise geöffnet werden soll. Bei Applikationen macht das Öffnen mithilfe anderer Programme natürlich nicht allzu viel Sinn (der Anwender will sie einfach nur starten), aber bei Dokumenten kommt dies häufiger vor. Zum Beispiel möchte ein Benutzer alle Word-Dokumente auf seinem Rechner mit `TextEdit` ansehen. Dann bleibt der Dokumenttyp auf `Word` stehen, der `Creator` wird aber auf `TextEdit` geändert.

Wenn Ihre Java-Anwendung Dokumentdateien erzeugt, können Sie bei Apple auf der Seite <http://developer.apple.com/dev/cftype/> einen `Creator-Code` beantragen, sofern der gewünschte Code nicht schon belegt ist (`Creator-Codes`, die nur aus Kleinbuchstaben bestehen, sind übrigens für Apple reserviert). Allerdings müssen Sie einen solchen Code nicht zwingend registrieren, stattdessen geben Sie dann einfach `????` als Signatur an und setzen »Identifizier« (s. u.) auf einen passenden Wert.

Zum Ändern der Signatur können Sie im Datei-Informations-Dialog eines Dokuments im Bereich »Öffnen mit« die bevorzugte Applikation ändern. Den Typ können Sie in diesem Dialog nur bedingt durch eine Änderung der Dateinamenserweiterung beeinflussen – eine weiter gehende Kontrolle ist aber eigentlich auch gar nicht nötig. Falls Sie dennoch ein bisschen mit dem `Type` und `Creator` herumexperimentieren wollen, finden Sie auf <http://www.frederikseiffert.de/filetype/> die Anwendung »FileType«, die eine einfache Oberfläche dafür anbietet. Apple selbst liefert nur Kommandozeilenwerkzeuge im Verzeichnis `/Developer/Tools/` mit. Sehen Sie sich bei Bedarf mit `man` die Informationen zu den dortigen Programmen an, insbesondere zu `GetFileInfo` und `SetFile`. Und wenn Sie sowieso am liebsten mit der Shell arbeiten, sollten Sie sich den Befehl `lsmac` aus den »osxutils« ansehen (Download von <http://sourceforge.net/projects/osxutils>).

Am häufigsten hatten Sie mit dem Type und Creator bisher vermutlich bei Dateien zu tun, die Sie aus dem Internet heruntergeladen haben. Manchmal werden diese einem falschen oder zumindest unerwünschten Programm zugeordnet, was man meistens am Programmsymbol erkennt. Das System hat dann zu einer Dateinamenserweiterung den falschen Type und/oder Creator »geraten«. Auf <http://www.clauss-net.de/misfox/misfox.html> finden Sie das Programm »MisFox«, mit dem Sie solche Zuordnungen systemweit festlegen oder korrigieren können.

► **Identifier**

Der »Identifier« ist der neuere Weg, eine Applikation ohne Creator-Code (Signatur) eindeutig für das System zu kennzeichnen. Geben Sie dazu eine eindeutige Zeichenkette im Java-Package-Stil an, hier beispielsweise `com.sun.Java2Demo`. Für eigene Anwendungen verwenden Sie natürlich Ihre Domain und beliebige Untergruppierungen im Format `de.meinedomain.projektname.MeinProgrammXyz`. Sie sollten immer entweder »Signature« auf einen Wert ungleich `???` setzen oder aber einen passenden »Identifier« angeben!

JVM-Parameter

► **VM-Options**

In dieses Feld tragen Sie – optional – Parameter für den Aufruf der Java Virtual Machine ein. Oft sind dies die erweiterten `-X`-Optionen, aber Sie können alle zur Verfügung stehenden Optionen verwenden, beispielsweise `-server`. Eine Übersicht über alle VM-Optionen finden Sie im Anhang.

► **Heap Minimum Size, Heap Maximum Size**

Mit diesen beiden Feldern können Sie den anfänglichen und den maximalen Speicherverbrauch der JVM festlegen. Hier eingetragene Werte werden mit den VM-Optionen `-Xms` und `-Xmx` gesetzt.

► **Set Working Directory to Inside Application Package**

Normalerweise ist das Arbeitsverzeichnis von Java-Applikationen das Verzeichnis, in dem das Programmpaket liegt. Wenn Sie diese Option einschalten, wird das Arbeitsverzeichnis dagegen auf `$APP_PACKAGE/Contents/Resources/Java` gesetzt, also auf das Verzeichnis, wo die JAR-Archive der Anwendung liegen.

► **Additional Properties**

Hier können Sie beliebig viele System-Properties als Schlüssel/Wert-Paare definieren und damit das Laufzeitverhalten Ihrer Anwendung konfigurieren. Eine Übersicht über alle Apple-spezifischen System-Properties finden Sie im Anhang.

Programmpaket-Einstellungen

► Allow Mixed Localizations

Schalten Sie diese Option generell ein. Wichtig ist sie dann, wenn Sie in Ihrem Programmpaket Übersetzungen der `Info.plist`-Zeichenketten anbieten, wie Sie das in Abschnitt 4.3.2 gesehen haben.

► Development Region

Gibt an, wo die Anwendung entwickelt wurde. Auch wenn Sie hier »German« eintragen möchten, scheint es besser zu sein, immer »English« zu verwenden.

► Info Dictionary Version

Gibt an, mit welcher Versionsnummer der `Jar Bundler` die `Info.plist`-Konfigurationsdatei erzeugt. Lassen Sie ganz einfach die Vorgabe – hier »6.0« – stehen.

Nun haben Sie den `Jar Bundler` fertig konfiguriert! Klicken Sie auf »**Create Application...**« und suchen Sie im erscheinenden Dateiauswahl-Dialog ein geeignetes Verzeichnis (z. B. den Desktop) und einen Programmnamen aus. Die Endung `.app` wird automatisch angehängt. Wenn das Programm nicht erzeugt wird oder es zu irgendwelchen Problemen kommt, können Sie sich in der Konsole ein kurzes Protokoll vom `Jar Bundler` ansehen.

Starten Sie das Programm, und Sie werden sehen, dass sich das Java-Programmpaket nun wie eine ganz normale Mac OS X-Applikation anfühlt – mit korrektem Programmnamen und einem schönen Symbol! Wenn Sie den »Über...«-Dialog im Programm-Menü öffnen, tauchen dort die beiden Einträge aus den Feldern »Version« und »Short Version« auf (falls Sie nicht schon einen eigenen Über-Dialog programmiert haben, wie er in Kapitel 3, *Grafische Benutzungsoberflächen*, vorgestellt wurde). Und im Datei-Informationsdialog wird die Version inklusive Copyright angezeigt.

In diesem Beispiel wurde »Java2D Test« als Bundle-Name und »Java2DTest« (ohne Leerzeichen) für den Programmnamen verwendet. Den Programmnamen finden Sie im Datei-Informations-Dialog unter dem Programmsymbol im Finder und über dem Dock-Icon. Der Bundle-Name wird dagegen im Programm-Menü angezeigt. Während dies hier zur Verdeutlichung dient, ist es bei eigenen Applikationen besser, wenn Sie in beiden Fällen die gleiche Zeichenkette verwenden.

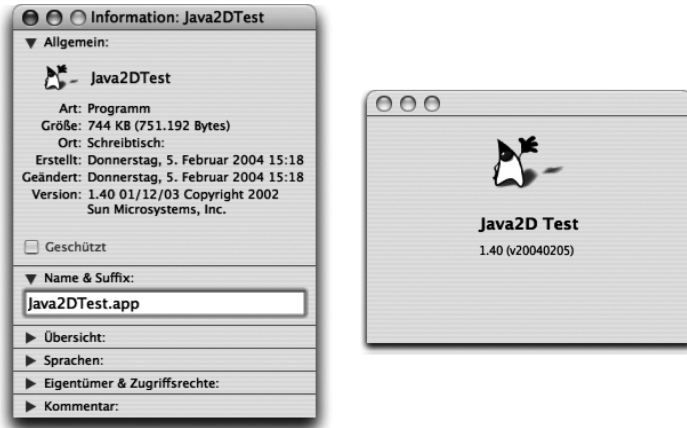


Abbildung 4.13 Datei-Informations- und »Über...«-Dialog der neuen Applikation

Wenn der Finder nach der Programmgenerierung nicht das Icon anzeigt, das Sie in der `icons`-Datei angegeben haben, führen Sie folgende Schritte in der angegebenen Reihenfolge aus, bis einer davon zum Erfolg führt:

1. Starten Sie den Finder neu, indem Sie `⌘` + `⌘` + `Esc` drücken, im »Programme sofort beenden«-Dialog den Finder auswählen und auf »Neu starten« klicken.
2. Melden Sie sich über das Apfel-Menü ab und wieder an.
3. Löschen Sie die Datei `~/Library/Caches/com.apple.LaunchServices.UserCache.csstore` und melden Sie sich danach ab und wieder an.
4. Löschen Sie die Datei `/Library/Caches/com.apple.LaunchServices.LocalCache.csstore` und starten Sie danach den Rechner neu.

Weitere Informationen und Beispiele zum `Jar Bundler` finden Sie im Verzeichnis `/Developer/Documentation/Java/Conceptual/Jar_Bundler/` auf Ihrer Festplatte.

Obwohl der `Jar Bundler` gut funktioniert, hat er einen großen Nachteil: Er ist schlecht automatisierbar. Jedes Mal, wenn Sie Änderungen an einem Programmpaket vornehmen wollen, müssen Sie alle Einstellungen im `Jar Bundler` neu eingeben oder die erzeugten Dateien innerhalb des Pakets (vor allem `Info.plist`) von Hand ändern. Deshalb kann es Sinn machen, ein Programmpaket gleich von Anfang an komplett von Hand zu erzeugen – was dann nämlich mit einem Shell-Skript oder Ant-Task automatisierbar ist. Oder Sie verwenden eine geeignete Entwicklungsumgebung, die sich alle Einstellungen zu einem Paket merken kann.

4.3.4 Project Builder, Xcode und Eclipse

Derzeit sind vor allem Apples Entwicklungsumgebungen so gut an Mac OS X angepasst, dass sie nach dem Übersetzen eines Software-Projekts am Ende automatisch eine Mac OS X-Applikation erzeugen können. Die Einstellungen für das Programmpaket werden dabei in der jeweiligen Projektdatei gespeichert. Der `Jar Bundler` wird dann nicht mehr benötigt.

Der **Project Builder** und **Xcode** können verschiedene Java-Projekttypen erzeugen: *Tool* und *Application*. Während der Tool-Typ JAR-Archive erzeugt, bekommen Sie beim Application-Typ ein fertiges Programmpaket. Wenn Sie also ein neues Projekt als »Java AWT Application« oder »Java Swing Application« erzeugen, sehen Sie bei den Target-Informationen in etwa Abbildung 4.14. Für die Erzeugung des Programmpakets ist der Abschnitt »Info.plist Entries« interessant. Hier können Sie in den Bereichen »Basic Information«, »Display Information«, »Application Icon«, »Cocoa Specific« und »Pure Java Specific« all die Einstellungen vornehmen, die auch der `Jar Bundler` bietet. Die einzelnen Einträge wurden bereits im vorangegangenen Abschnitt beschrieben.

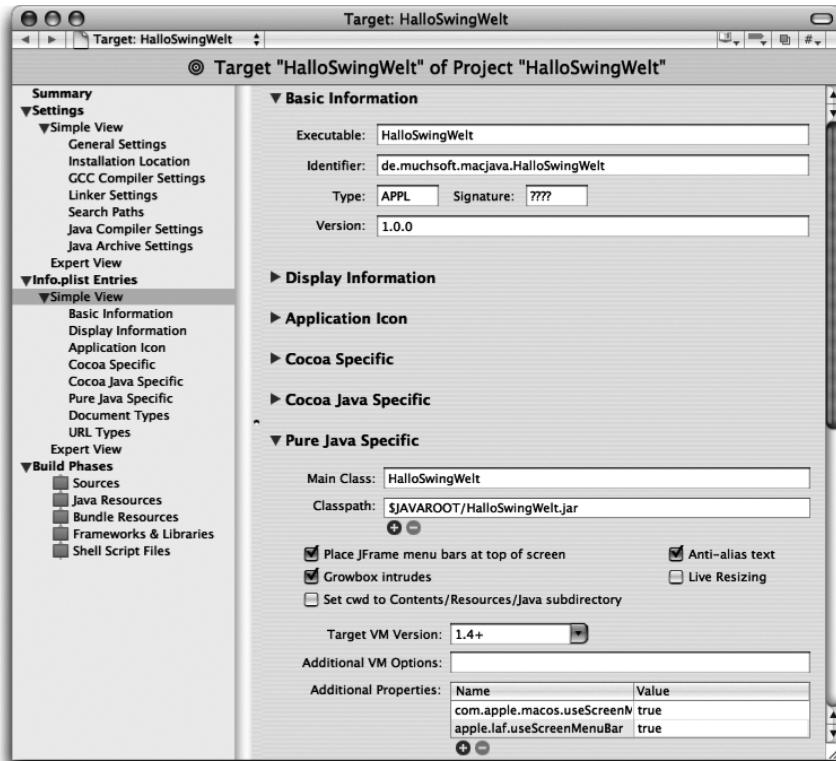


Abbildung 4.14 Programmpakete in Xcode konfigurieren

Darüber hinaus erlauben der Project Builder und Xcode die Konfiguration bestimmter Dokumententypen. Im Bereich »Document Types« können Sie eintragen, welche Dokumente von Ihrer Applikation geöffnet bzw. bearbeitet werden können (siehe Abbildung 4.15). Im Beispiel soll die Applikation ein *Viewer* für HTML-Dokumente sein, diese also nur anzeigen können. Dies wird mit der »Role« festgelegt, die alternativ auch »Editor« (zum Anzeigen und Bearbeiten von Dokumenten) oder »None« (die Applikation verarbeitet den Typ nicht, es werden nur Icon-Informationen festgelegt) sein kann. Folgende weitere Felder finden Sie hier:

► **Extensions**

Legt alle erlaubten Dateinamenserweiterungen für einen Dokumenttyp fest, die üblicherweise kleingeschrieben und durch Leerzeichen getrennt werden. Das Sternchen * bezeichnet alle Erweiterungen, die Anführungszeichen werden von der Entwicklungsumgebung automatisch eingefügt.

► **File Package**

Schalten Sie diese Option nur ein, wenn es sich bei den Dokumentdateien um Dateipakete handelt, die ähnlich wie Programmpakete aufgebaut sind. Die typischen von Java-Applikationen verarbeiteten Dokumente sind *keine* Dateipakete.

► **MIME Types**

Ein MIME-Typ ist quasi ein Internet-Dokumenttyp, der unter anderem beim Herunterladen von Dateien eine Rolle spielt, beispielsweise `text/plain` (ASCII-Text) oder `text/html`. Wenn er Ihnen bekannt ist, geben Sie hier an, ansonsten lassen Sie das Feld leer. Entspricht der Dokumenttyp mehreren MIME-Typen, werden sie wieder durch Leerzeichen getrennt angegeben.

► **OS Types**

Dies ist der vier Zeichen lange Type-Code, den Sie weiter vorne schon bei *Type* und *Creator* kennen gelernt haben. Tragen Sie hier den oder die Codes ein (oder lassen Sie das Feld einfach leer). Wenn ein Code aus Sonderzeichen (das Leerzeichen gehört dazu) besteht, müssen Sie ihn in Anführungszeichen setzen. Wenn Ihre Applikation Dokumente mit beliebigem OS-Typ öffnen kann, tragen Sie als Sonderfall `****` ein.

► **Icon File**

Hiermit können Sie jedem Dokumenttyp optional ein eigenes Dokument-symbol verpassen.

► **Document Class**

Wird nur bei der Cocoa-Programmierung benötigt und ist für Java nicht relevant.

Wie eine Java-Applikation letztendlich darauf reagiert, wenn der Benutzer eine Dokumentdatei auf das Programmsymbol zieht und das Dokument geöffnet werden soll, haben Sie bereits in Kapitel 3, *Grafische Benutzungsoberflächen*, mit der Methode `handleOpenFile()` aus den Interfaces `com.apple.mrj.MRJ-OpenDocumentHandler` und `com.apple.eawt.ApplicationListener` kennen gelernt.

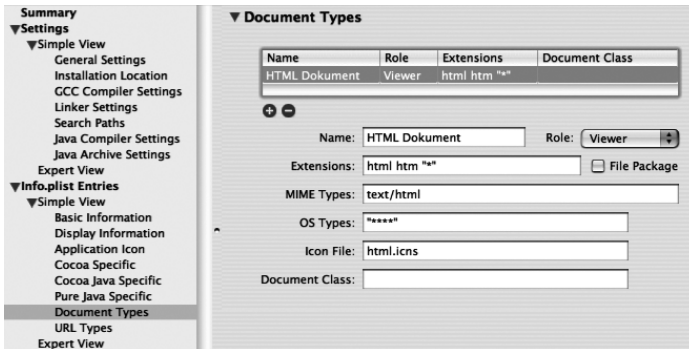


Abbildung 4.15 Dokumenttypen festlegen

Das direkte Erzeugen von Programmpaketen wird aber auch von neueren **Eclipse**-Versionen unterstützt! Sie benötigen dafür die Version 3.0M7 oder neuer. Wenn Sie damit ein Projekt fertig übersetzt haben, rufen Sie den Menüpunkt **File · Export** auf und wählen dort die Option »Mac OS X application bundle« aus (siehe Abbildung 4.16). Es folgen dann drei Dialogseiten, die sehr ähnlich zum `Jar Bundler` aufgebaut sind.

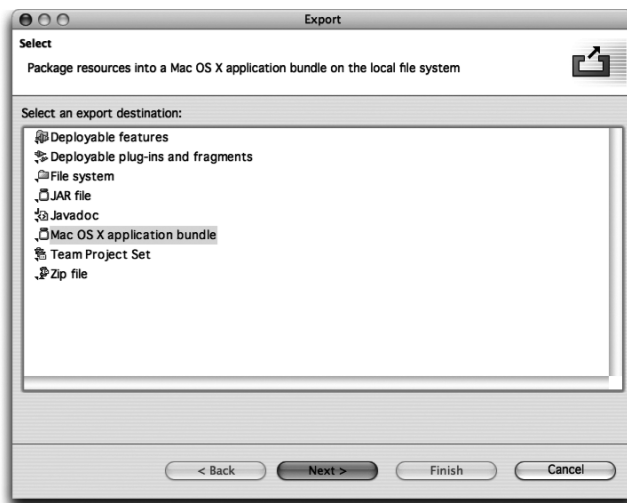


Abbildung 4.16 Eclipse-Export-Wizard

Die Export-Möglichkeit als Programmpaket steht derzeit nur bei der Mac OS X-Version von Eclipse zur Verfügung. Auf anderen Systemen oder mit älteren Eclipse-Versionen müssen Sie die MacOS X-Applikation manuell oder mit einem Skript erzeugen.

Für NetBeans ist geplant, dass es ab Version 4.0 ebenfalls möglich sein soll, Mac OS X-Applikationen direkt zu erzeugen.

4.3.5 Programmpakete von Hand erzeugen

Wenn Sie das Erzeugen von Mac OS X-Applikationen komplett automatisieren wollen, aber eine Entwicklungsumgebung einsetzen, die Programmpakete nicht direkt erzeugen kann, können Sie ein Programmpaket mit seiner Verzeichnisstruktur und seinen notwendigen Dateien auch von Hand zusammensetzen oder die einzelnen Schritte von einem Shell-Skript ausführen lassen. In diesem Abschnitt erfahren Sie den genauen Aufbau der Konfigurationsdateien innerhalb eines Pakets, damit Sie sich ein solches Skript schreiben können (ein fertiges Ant-Skript wird in Kapitel 8, *Werkzeuge*, vorgestellt). Sie können die Informationen aber natürlich auch einfach nur dazu verwenden, um ein fertig erzeugtes Programmpaket nachträglich von Hand zu optimieren.

In sechs Schritten gelangen Sie zu einer fertigen Mac OS X-Applikation. Gehen Sie einfach die folgenden Anweisungen durch, oder verwenden Sie `/examples/ch04/Minimal.app` von der Buch-CD als Grundlage, die Sie anpassen.

1. Erzeugen Sie die nötige Verzeichnisstruktur. Das oberste Verzeichnis bekommt den Namen Ihrer Applikation plus die Erweiterung `.app`. Darin legen Sie weitere Ordner an, so dass sich folgende Hierarchie ergibt:

```
IhreApplikationXyz.app
  Contents
    MacOS
    Resources
    Java
```

2. Kopieren Sie die Datei `JavaApplicationStub` aus dem Verzeichnis `/System/Library/Frameworks/JavaVM.framework/Versions/Current/Resources/MacOS/` in das MacOS-Verzeichnis des Programmpakets. Von diesem UNIX-Programm wird Ihre Java-Anwendung letztendlich gestartet. Apple liefert mit jeder Systemversion ein neues `JavaApplicationStub` mit, und bei der Erzeugung eines Programmpakets sollten Sie nach Möglichkeit die aktuellste Version verwenden. Wenn Sie nicht unter Mac OS X entwickeln, können Sie einfach den Stub einer bestehenden Mac OS X-Applikation verwenden, zum Beispiel `Minimal.app/Contents/MacOS/JavaApplicationStub`.

3. Erzeugen Sie eine Textdatei `Info.plist` im `Contents`-Verzeichnis, oder kopieren Sie eine bestehende Datei dort hinein und passen Sie sie an. `Minimal.app/Contents/Info.plist` ist ein guter Ausgangspunkt – alle Stellen, die Sie ändern müssen, enthalten die Zeichenkette »TODO«. Der genaue Aufbau dieser Datei ist weiter unten beschrieben.
4. Erzeugen Sie eine Textdatei `PkgInfo` im `Contents`-Verzeichnis, die exakt aus den acht Zeichen `APPL????` besteht. Anstelle der vier Fragezeichen können Sie auch einen Creator-Code eintragen, sofern Sie für Ihre Applikation einen reserviert haben.
5. Kopieren Sie eine passende Icon-Datei in das `Resources`-Verzeichnis. Wenn Sie keine eigene `icns`-Datei erzeugt haben, können Sie die Standardsymbole `/Developer/Applications/JavaTools/JarBundler.app/Contents/Resources/GenericJavaApp.icns` oder `Minimal.app/Contents/Resources/GenericJavaApp.icns` verwenden.
6. Legen Sie Ihr(e) JAR-Archiv(e) in das `Java`-Verzeichnis des Pakets.

In einigen Dokumentationen empfiehlt Apple noch, beim Paketverzeichnis das so genannte `Bundle-Bit` zu setzen, was Sie mit dem Kommandozeilenaufruf

```
/Developer/Tools/SetFile -a B IhreApplikationXyz.app
```

erreichen. Das scheint aber bei MacOS X 10.2 und 10.3 nicht mehr nötig zu sein, und Apples Entwicklerwerkzeuge setzen dieses Bit auch gar nicht. Wichtig ist dagegen, dass bei dem obersten Verzeichnis die `x`-Bits gesetzt sind. Das sollte bei Verzeichnissen aber sowieso der Fall sein, ansonsten könnten Sie nämlich gar nicht in das Verzeichnis wechseln. Falls die Zugriffsrechte doch einmal verstellt sind, können Sie sie mit `chmod 755 IhreApplikationXyz.app` wieder korrekt setzen.

Nun müssen Sie noch die Datei `Info.plist` modifizieren, damit sie an Ihre Java-Klassen und an den Programmnamen angepasst ist. Wenn Sie dies vergessen oder dabei einen Fehler machen, startet Ihre Applikation nicht!

Falls sich ein Programmpaket nicht starten lässt, können Sie sich im Terminal ausführliche Debug-Ausgaben während des Startvorgangs anzeigen lassen. Dazu müssen Sie eine Umgebungsvariable setzen und das Programm dann von Hand starten:

```
cd IhreApplikationXyz.app
setenv JAVA_LAUNCHER_VERBOSE 1
Contents/MacOS/JavaApplicationStub
```

Information Property List

MacOS X verwendet für viele Konfigurationsdateien das XML-Format, das auch bei `plist`- oder *Property-List*-Dateien eingesetzt wird. `Info.plist` ist eine solche Datei und dient speziell dazu, die Eigenschaften (*Properties*) von MacOS X-Applikationen zu konfigurieren. Da es sich um ein ganz normales XML-Dokument handelt, können Sie dieses mit einem beliebigen Editor bearbeiten. Im Folgenden sehen Sie die `Info.plist`-Datei, die der Jar Bundler für die `Java2DTest`-Applikation aus Abschnitt 4.3.3 erzeugt hat:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleAllowMixedLocalizations</key>
    <string>>true</string>
    <key>CFBundleDevelopmentRegion</key>
    <string>English</string>
    <key>CFBundleExecutable</key>
    <string>JavaApplicationStub</string>
    <key>CFBundleGetInfoString</key>
    <string>1.40 01/12/03 Copyright 2002 Sun ... </string>
    <key>CFBundleIconFile</key>
    <string>duke.icns</string>
    <key>CFBundleIdentifier</key>
    <string>com.sun.Java2Demo</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleName</key>
    <string>Java2D Test</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>1.40</string>
    <key>CFBundleSignature</key>
    <string>????</string>
    <key>CFBundleVersion</key>
    <string>20040205</string>
    <key>Java</key>
</dict>
    <key>ClassPath</key>
```

```

<string>$JAVAROOT/Java2D.jar</string>
<key>JVMVersion</key>
<string>1.4</string>
<key>MainClass</key>
<string>java2d.Java2Demo</string>
<key>Properties</key>
<dict>
  <key>apple.awt.brushMetalLook</key>
  <string>true</string>
  <key>apple.awt.showGrowBox</key>
  <string>true</string>
  <key>apple.laf.useScreenMenuBar</key>
  <string>true</string>
</dict>
<key>VMOptions</key>
<string>-Xfuture</string>
</dict>
</dict>
</plist>

```

Listing 4.2 Eine typische Info.plist-Datei

Wie bei jeder XML-Datei finden Sie einen Prolog `<?xml ... ?>` und die Definition des Dokumenttyps am Anfang, danach folgt das Wurzelement `<plist>`. Im Prolog sehen Sie auch, dass als Zeichensatz normalerweise »UTF-8«, also eine Unicode-Kodierung, verwendet wird, aber Sie können hier auch andere Kodierungen eintragen.

Property-List-Dateien bestehen aus so genannten *Dictionaries* (Wörterbüchern), in denen einem Schlüssel (*Key*) ein Wert, eine Liste von Werten (*Array*) oder ein untergeordnetes Dictionary zugewiesen wird. Im obigen Beispiel ist also dem Schlüssel `CFBundleIconFile` der Wert `duke.icns` zugeordnet. Ein Dictionary wird mit dem *Start-Tag* `<dict>` eingeleitet und mit dem *Ende-Tag* `</dict>`, wie es das XML-Format vorschreibt, beendet. Die Schlüsselnamen darin sind von `<key>` und `</key>` umschlossen, die jeweils folgenden Werte von `<string>` und `</string>` (es gibt außer »string« noch andere Werttypen, die Sie im weiteren Verlauf bei Bedarf kennen lernen).

Das oberste Dictionary wird vom Betriebssystem benötigt, um die Anwendung korrekt starten zu können. Die Schlüsselnamen hierin beginnen normalerweise mit `CFBundle`. Nach den `CFBundle`-Schlüsseln finden Sie am Ende den `Java`-Schlüssel mit einem untergeordneten Dictionary – hier wird die Java-Laufzeitumgebung konfiguriert.

Die folgende Liste beschreibt die für eine Java-Applikation relevanten `CFBundle`-Schlüssel, und fast alle dieser Schlüssel sollten auch in Ihren `Info.plist`-Dateien enthalten sein:

- ▶ `CFBundleAllowMixedLocalizations`
Wenn dieser Schlüssel den Wert »true« hat, werden Lokalisierungen im `Resources`-Ordner beachtet. Obwohl das Setzen bei neueren Mac OS X-Versionen nicht mehr notwendig ist, schadet es nicht, diesen Schlüssel zu verwenden.
- ▶ `CFBundleDevelopmentRegion`
Dieser Schlüssel gibt das Land (oder eher die Sprache) an, in dem die Anwendung entwickelt wurde, beispielsweise `English` oder `German`. Wenn eine Zeichenkette nicht übersetzt gefunden werden kann, wird der Standardwert aus der hier angegebenen Lokalisierung verwendet. Die praktische Erfahrung hat allerdings gezeigt, dass es offenbar am besten ist, wenn Sie immer auch eine englische Lokalisierung anbieten und hier dann `English` eintragen.
- ▶ `CFBundleDisplayName`
Wenn Sie den Namen Ihrer Applikation mit `CFBundleName` lokalisieren, sollten Sie `CFBundleDisplayName` sowohl in `Info.plist` als auch in den lokalisierten `InfoPlist.strings` (s. u.) mit dem passend übersetzten Namen eintragen. Falls Ihr Programmname nicht übersetzt werden muss, lassen Sie diesen Schlüssel weg, damit der Finder nicht unnötig nach den Übersetzungen sucht.
- ▶ `CFBundleExecutable`
Legt den Namen der ausführbaren Datei fest, also des Programms, das letztendlich Ihre Java-Applikation startet. Dieser Name muss mit der ausführbaren Datei in `Contents/MacOS/` übereinstimmen. Bei Java-Anwendungen ist dies normalerweise `JavaApplicationStub`, aber Sie können beides – die Datei und diesen Schlüsselwert – auch umbenennen, damit der Name beispielsweise mit dem Paketnamen übereinstimmt. Dieser Schlüssel muss angegeben werden, sonst startet Ihre Applikation nicht!
- ▶ `CFBundleGetInfoHTML`
Entspricht `CFBundleGetInfoString`, aber bei diesem Schlüssel können Sie eine HTML-formatierte Zeichenkette angeben. Wenn sowohl dieser als auch der folgende Schlüssel gesetzt sind, wird nur `CFBundleGetInfoHTML` beachtet.
- ▶ `CFBundleGetInfoString`
Hiermit bestimmen Sie, welchen Text der Datei-Informations-Dialog zu Ihrem Programm anzeigt. Die Zeichenkette sollte aus einer Versionsnummer

gefolgt von dem Copyright bestehen, also z.B. »1.1.0, Copyright 2004 muchsoft.com. Alle Rechte vorbehalten«. Dieser Schlüssel wird ignoriert, wenn der Schlüssel `CFBundleGetInfoHTML` angegeben ist!

- ▶ `CFBundleIconFile`
Wenn Sie für Ihr Programmpaket ein eigenes Programmsymbol entworfen haben, müssen Sie hier den Namen der Icon-Datei eintragen. Die Datei wird in `Contents/Resources/` gesucht, und wenn Sie keine Dateinamenserweiterung angeben, wird automatisch `.icns` angenommen. Wenn Sie sich hier vertippen, wird Ihr Programmpaket mit einem Standardsymbol dargestellt.
- ▶ `CFBundleIdentifier`
Dies ist eine von Ihnen frei wählbare Zeichenkette, um Ihr Programm eindeutig im System zu identifizieren. Damit die Zeichenkette wirklich möglichst eindeutig ist, sollte die Zeichenkette nach den Java-Package-Konventionen aufgebaut sein, beispielsweise `com.sun.Java2Demo`.
- ▶ `CFBundleInfoDictionaryVersion`
Um mit künftigen Formaten der Property-List-Dateien kompatibel zu sein, sollten Sie immer die Version angeben, in der Ihre `Info.plist`-Datei gespeichert wurde. Derzeit ist »6.0« aktuell.
- ▶ `CFBundleName`
Gibt den kurzen (maximal 16 Zeichen langen) Namen des Programmpakets an, der auch für das Programm-Menü und den »Über...«-Dialog verwendet wird. Wenn Sie diesen Schlüssel lokalisieren, sollten Sie zusätzlich auch den Schlüssel `CFBundleDisplayName` angeben.
- ▶ `CFBundlePackageType`
Dies ist der vier Zeichen lange Dateityp. Bei Mac OS X-Applikationen steht hier immer `APPL`. Der Wert muss mit den ersten vier Zeichen aus der Datei `Contents/PkgInfo` exakt übereinstimmen.
- ▶ `CFBundleShortVersionString`
Hiermit legen Sie die »öffentliche« Versionsnummer Ihres Programms fest. Die Zeichenkette ist üblicherweise im Format *n.m.o* aufgebaut, wobei *n* die Hauptversion (major version) angibt und *m* und *o* Unterversionsnummern (minor revision) angeben. Während `CFBundleVersion` eine sich ständig ändernde Build-Nummer angibt, bleibt `CFBundleShortVersionString` üblicherweise über mehrere Builds konstant.
- ▶ `CFBundleSignature`
Dies ist die Signatur bzw. der Creator-Code des Programms. Wenn Sie keinen eigenen Code bei Apple registriert haben, tragen Sie hier einfach vier Fragezeichen (????) ein. Dieser Wert muss mit den letzten vier Zeichen aus der Datei `Contents/PkgInfo` exakt übereinstimmen.

► `CFBundleVersion`

Dieser Schlüssel gibt die exakte Build-Nummer des Programms an. Üblicherweise wird für die Zeichenkette das Format *nn.n.nxxxx* verwendet. Nach der Hauptversionsnummer folgen zwei einstellige Unterversionsnummern, danach eines der Zeichen *abdf* und schließlich die genaue Build-Nummer. Unterversionsnummern und die Build-Nummer können bei Bedarf weggelassen werden. Gültige Werte für diesen Schlüssel sind beispielsweise *1.0.1*, *1.2.1b10*, *1.2d200*, *d125*, *101* und *1.0*.

Viele dieser Schlüssel haben Sie schon beim `Jar Bundler` gesehen, wo Sie die Werte in ähnlich benannte Felder eingeben konnten.

In der folgenden Liste sind nun alle erlaubten Schlüssel für das `Java-Dictionary` beschrieben. Der Schlüssel `MainClass` ist zwingend erforderlich, `JVMVersion` ist dringend empfohlen, und auch `ClassPath` werden Sie normalerweise setzen:

► `MainClass`

Gibt den vollqualifizierten Namen der Klasse mit der `main`-Methode an, in diesem Beispiel `java2d.Java2Demo`. Dieser Schlüssel muss angegeben werden.

► `JVMVersion`

Legt die zur Ausführung benötigte Java-Version fest. Tragen Sie hier *1.3+* ein, wenn Ihre Applikation mindestens Java 1.3 benötigt, oder *1.4+*, wenn Java 1.4 (oder neuer) Voraussetzung ist. Die weiteren erlaubten Werte wurden schon in der Tabelle in Abschnitt 4.3.3 beschrieben. Wenn Sie diesen Schlüssel nicht angeben, wird Ihre Java-Applikation automatisch mit Java 1.3.1 gestartet.

► `ClassPath`

Dieser Schlüssel legt den Klassenpfad fest und wird normalerweise auf ein JAR-Archiv im Verzeichnis `Contents/Resources/Java/` gesetzt – hier im Beispiel auf `§JAVAROOT/Java2D.jar`. Sie können aber auch einfach nur `§JAVAROOT/` angeben, dann werden einzelne Klassen in diesem Verzeichnis gefunden (bzw. Klassen in einer Java-Package-Verzeichnisstruktur). Wenn Sie mehrere Pfade setzen müssen, verwenden Sie statt des Werttyps »string« den Typ »array«:

```
<key>ClassPath</key>
<array>
  <string>§JAVAROOT/</string>
  <string>§JAVAROOT/Java2D.jar</string>
</array>
```


Wenn dieser Schlüssel nicht angegeben ist, wird als Klassenpfad das Verzeichnis der Applikation verwendet, es werden dann also einzelne Klassen außerhalb des Programmpakets gesucht – und genau das will man mit Programmpaketen ja eigentlich vermeiden.

► `WorkingDirectory`

Hiermit wird das anfängliche Arbeitsverzeichnis der Applikation festgelegt. Normalerweise wird dieser Schlüssel weggelassen, dann entspricht das Arbeitsverzeichnis dem der Applikation. Sie können es aber auch mit `$APP_PACKAGE/Contents/Resources/Java` (oder mit `$JAVAROOT`) auf das Verzeichnis der JAR-Archive bzw. Java-Klassen setzen.

► `VMOptions`

Wenn Sie die Java Virtual Machine mit bestimmten Parametern konfigurieren wollen, beispielsweise um den anfänglichen und maximalen Speicherverbrauch festzulegen (siehe Kapitel 17, *VM-Optionen*), tragen Sie hier eine Zeichenkette mit den durch Leerzeichen getrennten Optionen ein:

```
<key>VMOptions</key>
<string>-Xms=512m -Xmx=1024m</string>
```

Oder Sie verwenden – wie bei `ClassPath` – ein Array von Zeichenketten für die einzelnen Optionen:

```
<key>VMOptions</key>
<array>
  <string>-Xms=512m</string>
  <string>-Xmx=1024m</string>
</array>
```

► `Arguments`

Falls Ihre `main`-Methode Kommandozeilenparameter erwartet, können Sie hier eine Zeichenkette eintragen, in der die einzelnen Argumente durch Leerzeichen getrennt aufgeführt sind:

```
<key>Arguments</key>
<string>Ganz viele Argumente durch Leerzeichen
getrennt</string>
```

Alternativ verwenden Sie wieder ein Array, um die Argumente anzugeben. Das hat zudem den Vorteil, dass die Argumente dann auch Leerzeichen enthalten können!

```
<key>Arguments</key>
<array>
  <string>Ein einziges Argument mit Leerzeichen!</string>
</array>
```

► Properties

Dieser optionale Schlüssel beschreibt ein weiteres Unter-Dictionary, in dem Sie alle möglichen Systemeigenschaften (*System Properties*) konfigurieren können, die Sie auch beim Aufruf von `java` mit der `-D`-Option angeben können. Während Sie dies in der Kommandozeile mit `-Dapple.laf.useScreenMenuBar="true"` festlegen, wird im Properties-Dictionary das übliche Schlüssel/Wert-Paar verwendet:

```
<key>Properties</key>
<dict>
  <key>apple.laf.useScreenMenuBar</key>
  <string>true</string>
</dict>
```

Eine Auflistung der Apple-spezifischen System-Properties finden Sie im Anhang in Kapitel 16, *System-Properties*.

Die Pfadvariablen `$JAVAROOT` und `$APP_PACKAGE`, die Sie schon beim `Jar Bundler` kennen gelernt haben, dürfen Sie nur innerhalb des Java-Dictionaries verwenden!

Das Schreiben von XML-Dokumenten von Hand ist zwar machbar, aber die Gefahr ist viel zu groß, dass Sie sich bei den Strukturangaben vertippen – das Dokument ist dann oft sofort ungültig, da das XML-Format einen bestimmten Aufbau zwingend vorschreibt. Aus diesem Grund stellt Apple im Verzeichnis `/Developer/Applications/Utilities/` den `Property List Editor` zur Verfügung. Mit diesem Programm können Sie bestehende Property-List-Dateien bearbeiten oder neue erzeugen (siehe Abbildung 4.17).

Ohne XML-Quelltext zu manipulieren, können Sie hier neue Schlüssel anlegen und Werte mit ihren Typen zuweisen. Wenn Sie die Schaltfläche »Dump« anklicken, wird im unteren Bereich des Fensters aus den bisherigen Eingaben gültiger XML-Code generiert. Das ist allerdings nur als Unterstützung gedacht, wenn Sie schnell einen Überblick über die XML-Datei bekommen möchten, denn beim Speichern wird dieser XML-Quelltext sowieso automatisch erzeugt.

Wenn Sie sich eine vom `Jar Bundler` erzeugte `Info.plist`-Datei ansehen, werden Sie feststellen, dass sie mit der Versionsangabe »0.9« und einer »SYSTEM«-DTD gespeichert wurde. Seit längerem verwendet Apple aber eigentlich nur noch die `plist`-Version »1.0« mit einer »PUBLIC«-DTD (was Sie vorher schon im Beispiel gesehen haben). Wenn Sie eine alte `plist`-Datei ins neue Format bringen möchten, öffnen Sie diese einfach mit dem `Property List Editor` und speichern Sie sie dann sofort wieder – die Versionsnummer und die Dokumenttyp-Definition werden dabei aktualisiert.

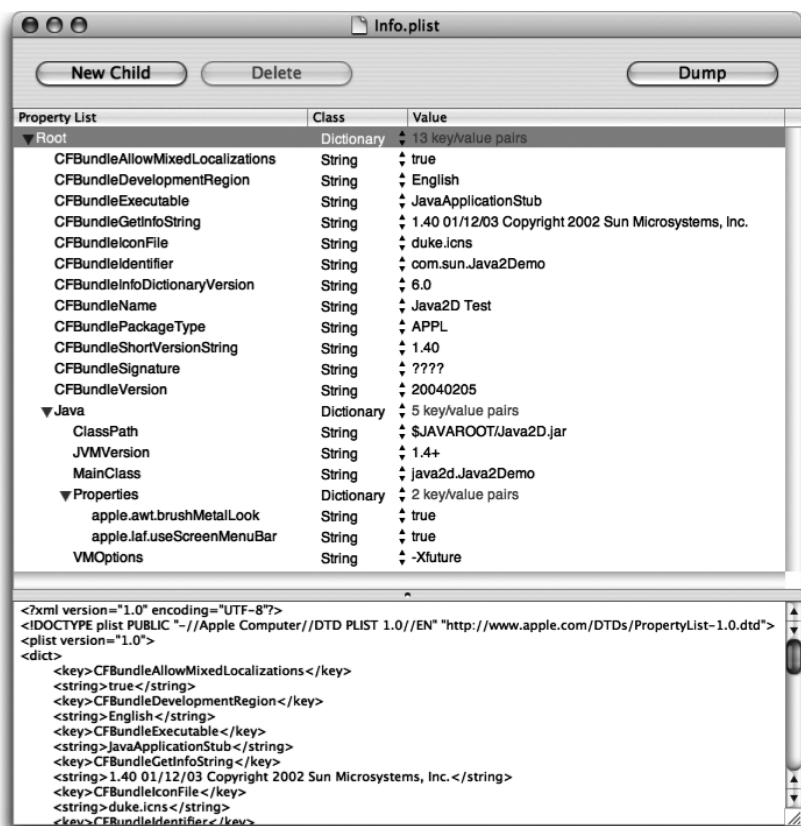


Abbildung 4.17 Property List Editor

Wenn Sie eine Java-Applikation pflegen, die noch für MacOS X 10.0 erzeugt worden war, finden Sie im Programmpaket zusätzlich zu Contents/Info.plist noch eine Datei Contents/Resources/MRJApp.properties. Darin wurden alle Systemeigenschaften gesetzt, die zum Starten und zum Betrieb einer Java-Anwendung nötig waren. Der Aufbau entspricht dem einer ganz normalen Java-Properties-Datei, wo einem Property-Namen ein Wert zugewiesen wird:

```

# nicht mehr so konfigurieren, sondern mit Info.plist!
com.apple.mrj.application.main=java2d.Java2Demo
com.apple.mrj.application.classpath=Contents/Resources/Java/Java
2D.jar
com.apple.macos.useScreenMenuBar=true

```

Listing 4.3 Alte MRJApp.properties-Datei (Ausschnitt)

Auch wenn diese Konfigurationsmethode zum Teil⁸ heute noch funktioniert, sollten Sie solche alten Java-Applikationen bei nächster Gelegenheit auf eine aktuelle `Info.plist`-Datei umstellen, da Apple seit MacOS X 10.1 von der Verwendung von `MRJApp.properties` abrät. Die folgende Tabelle listet auf, welche alte Property durch welchen Schlüssel im Java-Dictionary der `Info.plist`-Datei ersetzt wird:

Alter <code>MRJApp.properties</code> -Eintrag	Neuer <code>Info.plist</code> -Schlüssel
<code>com.apple.mrj.application.main</code>	<code>MainClass</code>
<code>com.apple.mrj.application.classpath</code>	<code>ClassPath</code>
<code>com.apple.mrj.application.parameters</code>	<code>Arguments</code>
<code>com.apple.mrj.application.workingdirectory</code>	<code>WorkingDirectory</code>
<code>com.apple.mrj.application.vm.options</code>	<code>VMOptions</code>
<code>com.apple.mrj.application.JVMVersion</code>	<code>JVMVersion</code>

Alle anderen Einträge aus `MRJApp.properties` listen Sie im `Properties-Dictionary` innerhalb des `Java-Dictionaries` auf; sie werden dort also als ganz normale `System-Properties` definiert.

Bei der Verwendung von `MRJApp.properties` waren außerdem noch zwei Einschränkungen zu beachten:

- ▶ Argumente für `main` wurden immer mit Leerzeichen getrennt als Zeichenkette übergeben – Argumente konnten also niemals Leerzeichen enthalten.
- ▶ Die Pfadvariable `$APP_PACKAGE` war noch unbekannt und wurde dementsprechend nicht ausgewertet.

Dokumenttypen konfigurieren

Wie Sie schon im vorangegangenen Abschnitt bei Xcode gesehen haben, können Sie Ihrer Java-Applikation Dokumenttypen zuordnen. Der Benutzer kann dann durch Doppelklick auf ein Dokument die Applikation starten. Ebenso können dann passende Dokumente zum Öffnen auf das Programmsymbol gezogen werden. Ermöglicht wird dies über einen weiteren Schlüssel im obersten Dictionary einer `Info.plist`-Datei, `CFBundleDocumentTypes`. Dieser Schlüssel bekommt als Wert ein Array von Dictionaries, und jedes dieser Dic-

⁸ Sofern der `JavaApplicationStub` im Programmpaket von einer Java 1.3-Implementation stammt. Wenn Sie ein aktuelles Java 1.4 installiert haben und den zugehörigen Stub verwenden, kennt dieser Stub die Datei `MRJApp.properties` gar nicht mehr.

tionaries definiert jeweils einen einzigen Dokumenttyp. Konfiguriert wird ein Dokumenttyp dann mit folgenden Schlüsseln:

- ▶ `CFBundleTypeExtensions`
Dieser Schlüssel enthält ein Array von Strings. Jede Zeichenkette gibt eine Dateinamenserweiterung (ohne den Punkt am Anfang) an. Um beliebige Erweiterungen öffnen zu können, tragen Sie das Sternchen (*) ein.
- ▶ `CFBundleTypeIconFile`
Gibt eine Zeichenkette mit der Icon-Datei an, die für Dokumente dieses Typs verwendet werden soll. Wenn Sie die Dateinamenserweiterung weglassen, wird automatisch nach einer Datei mit der Endung `.icns` gesucht.
- ▶ `CFBundleTypeMIMETypes`
Enthält ein Array von Strings. Jede Zeichenkette gibt dabei einen MIME-Typ an, der diesem Dokumenttyp zugeordnet werden soll, z.B. `image/jpeg` für JPEG-Bilder.⁹
- ▶ `CFBundleTypeName`
Legt einen eindeutigen, abstrakten Namen für diesen Dokumenttyp fest. Mit diesem Namen kann der Dokumenttyp später referenziert werden. Apple empfiehlt daher, den Namen im Java-Package-Stil zu vergeben. Dieser Schlüssel muss vorhanden sein.
- ▶ `CFBundleTypeOSTypes`
Enthält ein Array von Strings. Jede Zeichenkette legt dabei einen vier Zeichen langen Dateityp fest, wie Sie ihn früher schon bei *Type* und *Creator* gesehen haben. Wenn Ihre Applikation Dokumente mit beliebigem Typ öffnen soll, geben Sie als Sonderfall vier Sternchen (****) an.
- ▶ `CFBundleTypeRole`
Diese Zeichenkette gibt an, welche Rolle die Applikation bezüglich dieses Dokumenttyps spielt. Gültige Werte sind `Editor`, `Viewer`, `Shell` oder `None`. Bei `None` wird einfach nur ein Icon für den Dokumenttyp definiert, übliche Werte sind `Editor` (die Anwendung kann solche Dokumente bearbeiten) und `Viewer` (die Anwendung kann die Dokumente nur anzeigen). Dieser Schlüssel muss angegeben sein.

Damit ein Dokumenttyp gültig ist, muss auch mindestens einer der Schlüssel `CFBundleTypeExtensions`, `CFBundleTypeMIMETypes` und `CFBundleTypeOSTypes` angegeben sein.

⁹ Eine Liste der standardisierten MIME-Typen finden Sie auf <http://www.iana.org/assignments/media-types/>. Alternativ können Sie sich die Liste im Terminal mit `more /etc/httpd/mime.types` anzeigen lassen.

Das folgende Listing ist ein Ausschnitt aus einer `Info.plist`-Datei, mit dem der Applikation ein einziger Dokumenttyp zum Öffnen von HTML-Dateien zugeordnet wird:

```
<key>CFBundleDocumentTypes</key>
<array>
  <dict>
    <key>CFBundleTypeExtensions</key>
    <array>
      <string>html</string>
      <string>htm</string>
      <string>*</string>
    </array>
    <key>CFBundleTypeIconFile</key>
    <string>html.icns</string>
    <key>CFBundleTypeMIMETypes</key>
    <array>
      <string>text/html</string>
    </array>
    <key>CFBundleTypeName</key>
    <string>HTML Dokument</string>
    <key>CFBundleTypeOSTypes</key>
    <array>
      <string>****</string>
    </array>
    <key>CFBundleTypeRole</key>
    <string>Viewer</string>
  </dict>
</array>
```

Listing 4.4 `Info.plist`-Ausschnitt zur Zuordnung von Dokumenttypen

Denken Sie daran, dass das Öffnen der Dokumente nur dann klappt, wenn Sie die Methode `handleOpenFile()` passend implementiert haben!

Zeichenketten lokalisieren

In der Datei `Info.plist` tauchen einige Zeichenketten auf, die bei lokalisierten Anwendungen übersetzt werden müssen, beispielsweise das Copyright oder sogar der Programmname. Dazu legen Sie im Programmpaket im Ordner `Contents/Resources/` für jede gewünschte Sprache ein eigenes Unterverzeichnis an – `English.lproj/` für die englische Übersetzung, `German.lproj/` für die deutsche usw. In diesen Verzeichnissen erzeugen Sie jeweils eine Textdatei

`InfoPlist.strings`, in der Sie den `Info.plist`-Schlüsseln passend übersetzte Zeichenketten zuweisen:

```
/* Localized versions of Info.plist keys */
CFBundleName="Java 1.4.2 Plugin Einstellungen";
CFBundleGetInfoString="2.1.0 ... Alle Rechte vorbehalten.";
```

Listing 4.5 `InfoPlist.strings`

Beachten Sie, dass es sich bei diesen Textdateien um Unicode-Texte im UTF-16-Format handeln sollte. Am besten bearbeiten Sie die Texte also mit `/Programme/TextEdit`.

Von allen Schlüsseln aus `Info.plist` werden üblicherweise nur die folgenden in `InfoPlist.strings` lokalisiert:

- ▶ `CFBundleDisplayName`
- ▶ `CFBundleGetInfoHTML`
- ▶ `CFBundleGetInfoString`
- ▶ `CFBundleName`
- ▶ `CFBundleShortVersionString`

Wenn Sie den Namen Ihrer Applikation mit `CFBundleDisplayName` in diversen `InfoPlist.strings`-Dateien übersetzt haben, sollten Sie noch den Schlüssel `LSHasLocalizedDisplayName` in `Info.plist` eintragen, damit der lokalisierte Name vom Finder schneller gefunden und angezeigt wird:

```
<key>LSHasLocalizedDisplayName</key>
<true/>
```

Weitere Informationen zu Property-List-Dateien finden Sie bei Apple auf der Seite <http://developer.apple.com/documentation/MacOSX/Conceptual/BPRuntime-Config/>. Eine allgemeine Übersicht über »Bundles« und »Application Packaging« enthält das Dokument <http://developer.apple.com/documentation/MacOSX/Conceptual/SystemOverview/>.

4.4 Installationswerkzeuge

Wenn Sie Ihr Java-Programm als Mac OS X-Applikation verpackt haben, ist dies schon ein guter und wichtiger Schritt, damit der Anwender wie gewohnt Mac-typisch mit der Software arbeiten kann. Ein Problem bleibt aber noch bestehen – wie kommt Ihr Programm auf die Festplatte des Benutzers? Die einfachste Möglichkeit ist sicherlich, wenn Sie das Programmpaket als Archiv (`.zip` oder `.tar.gz`) verpacken, allerdings ist dies aus Benutzersicht nicht der beste Weg.

Die ausgepackten Programme bleiben zu häufig dort liegen, wo sie eigentlich nichts zu suchen haben, beispielsweise im Benutzerverzeichnis oder auf dem Schreibtisch.

Apples empfohlener Weg für die Software-Installation ist *Drag & Drop*. Das zu installierende Programm sollte nur aus einer einzigen Datei bestehen, die der Anwender von einem Volume (z.B. von einer CD-ROM oder, wie Sie gleich sehen werden, von einem Disk Image) irgendwohin kopieren kann, vorzugsweise in den `Programme`-Ordner. Der Vorteil für den Anwender ist, dass er genau weiß, wohin er das Programm kopiert hat – es wird nichts »unsichtbar« am System verändert. Und genauso einfach kann der Benutzer die Software dann auch wieder löschen.

Nur, wenn Ihre Anwendung an verschiedenen Stellen im System Komponenten installieren muss (beispielsweise ein systemweit genutztes JAR-Archiv), sollten Sie die Installation automatisiert ablaufen lassen. Dazu können Sie Apples eigene (kostenlose) Werkzeuge verwenden, oder Sie nehmen einen der zahlreichen kommerziellen Installer. Ein weiterer Grund für den Einsatz eines Installationsprogramms kann sein, dass Sie es dem Anwender so bequem wie möglich machen wollen. Zudem sieht eine dialoggesteuerte Installation professionell aus.

4.4.1 Mac OS X-Hausmittel

Mac OS X bringt sowohl für die Drag & Drop- als auch für die automatisierte Installation alle nötigen Werkzeuge mit, für letztere müssen Sie lediglich Apples Entwickler-Tools installiert haben.

Disk Images (DMG)

Disk Images sind Dateien mit der Endung `.dmg`, die Daten für ein virtuelles Laufwerk enthalten. Wenn das Image durch Doppelklick auf die DMG-Datei aktiviert (gemountet) wurde, können Sie damit wie mit einem ganz normalen Mac OS X-Volume arbeiten. Zum Erzeugen von Disk Images verwenden Sie das Festplatten-Dienstprogramm im Ordner `/Programme/Dienstprogramme/`.

Wenn Sie noch mit Mac OS X 10.2 arbeiten, finden Sie die im Folgenden vorgestellten Funktionen nicht im Festplatten-Dienstprogramm, sondern im Programm `Disk Copy` im selben Ordner.

Rufen Sie über das Menü **Images** • **Neu** den Menüpunkt »Leeres Image« auf (siehe Abbildung 4.18). Als Erstes geben Sie dem Image einen Namen, hier

»Test« (das `.dmg` wird automatisch angehängt). Danach müssen Sie die Größe des Volumes festlegen – die Größe ist nachträglich nicht mehr änderbar, Sie können also maximal so viele Daten darauf ablegen. Bei Bedarf können Sie das Image mit einem Passwort verschlüsseln, und als Format ist »Image lesen/schreiben« üblich. Nun können Sie mit »Erstellen« das Disk Image erzeugen lassen, es wird dann auch gleich aktiviert.



Abbildung 4.18 Disk Images mit dem Festplatten-Dienstprogramm erzeugen

Wenn Sie das aktivierte Volume per Doppelklick öffnen, erscheint ein ganz normales Finder-Fenster, in das Sie Ihre Programmdateien hineinziehen (und damit kopieren) können. Nachdem Sie so alle Programmkomponenten zusammengestellt haben, deaktivieren Sie das Volume (mit »Auswerfen« im Finder-Ablage-Menü oder, wie in Abbildung 4.19 gezeigt, über das Kontext-Popup-Menü). Die DMG-Datei speichert die soeben hineinkopierten Dateien weiterhin und kann nun an die Anwender verteilt werden!



Abbildung 4.19 DMG-Datei und aktiviertes Volume

Anstatt ein leeres Image zu erzeugen, können Sie im Festplatten-Dienstprogramm im Menü **Images · Neu** auch die Option »Image von Ordner...« verwenden. Dadurch wird der Ordnerinhalt (beispielsweise Ihr Programmpaket samt Dokumentation) in das neue Image kopiert, dessen Größe dann auch automatisch ermittelt wird. Zusätzlich können Sie das Image komprimieren oder als Nur-Lese-Image anlegen lassen.

Wenn Sie das Erzeugen eines Disk Images automatisieren wollen, können Sie dafür den Kommandozeilenbefehl `hdiutil` einsetzen, der deutlich mehr Optionen als das Festplatten-Dienstprogramm bietet. Es gibt aber auch zahlreiche Free- und Shareware-Programme, die die zusätzlichen `hdiutil`-Optionen zugänglich machen, darunter:

► **buildDMG**

Dieses kostenlose Perl-Skript lässt sich gut in den Build-Prozess einbinden, so dass Sie nach der Projektgenerierung eine fertige DMG-Datei bekommen.

<http://www.objectpark.org/buildDMG.html>

► **DMG Tool**

Dieses Freeware-Programm besitzt eine einfache Oberfläche, mit der man schnell zum Ziel kommt – wenn auch nicht sehr viel mehr Optionen als im Festplatten-Dienstprogramm geboten werden.

<http://www.them.ws/themsw/dmgtool/>

► **DMG Maker**

Der DMG Maker bietet in einer übersichtlichen Oberfläche mehr Kontrolle über die Image-Erzeugung, kostet dafür aber auch zehn US-\$ Shareware-Gebühr.

<http://www.pliris-soft.com/products/dmgmaker/>

Wenn Sie sich dafür entscheiden, zum Erzeugen von DMG-Dateien ein Shareware-Programm zu registrieren, sollten Sie sich vorher auf jeden Fall noch das nur wenig teurere FileStorm (s. u.) ansehen, das eine gut benutzbare Oberfläche besitzt und mit dem Sie dem Disk Image sehr einfach eine schöne Hintergrundgrafik verpassen können.

Mit ein bisschen Handarbeit können Sie einem Disk Image aber auch ohne zusätzliche Software ein Hintergrundbild hinzufügen. Kopieren Sie dazu die gewünschte Bilddatei – beispielsweise `background.jpg` – in das aktivierte Image-Volume. Rufen Sie für dieses Finder-Fenster dann im Finder-Menü **Darstellung · Darstellungsoptionen** einblenden auf (siehe Abbildung 4.20). Schalten Sie dort auf jeden Fall die Option »Nur dieses Fenster« ein und wählen Sie dann unten bei »Hintergrund« die Option »Bild«. Achten Sie darauf, dass Sie

die Bilddatei aussuchen, die Sie gerade auf das Image-Volume kopiert haben! Anschließend können Sie noch die Symbolgröße auf 128x128 stellen.

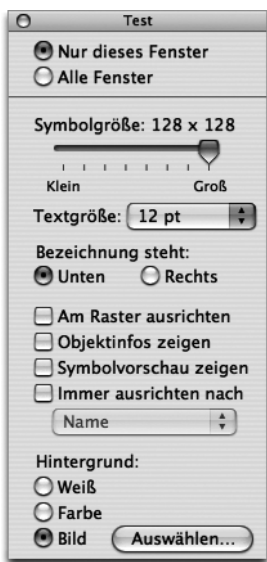


Abbildung 4.20 Darstellungsoptionen für ein Disk-Image-Hintergrundbild

Damit das Icon für die Bilddatei nicht mehr im Volume-Fenster zu sehen ist, müssen Sie diese nun unsichtbar machen. Dazu verwenden Sie das `Terminal` und wechseln in das Verzeichnis des Image-Volumes (im Beispiel wird der Volume-Name »Test« angenommen). Mit dem Befehl `SetFile` setzen Sie dann das Unsichtbar-Attribut:

```
cd /Volumes/Test/
/Developer/Tools/SetFile -a V background.jpg
```

Wenn Sie das Image-Volume auswerfen und neu aktivieren, wird das Hintergrundbild angezeigt, aber die Bilddatei selbst ist nicht mehr sichtbar. Im `Festplatten-Dienstprogramm` können Sie das Disk Image nun noch mit dem Menüpunkt **Images • Konvertieren...** in das Format »Nur lesen« bringen. Dadurch wird die DMG-Datei auch gleich komprimiert, und Nur-Lesen-Images werden vom Finder nach dem Aktivieren automatisch geöffnet.

PackageMaker

Sobald für eine Software-Installation mehrere Dateien an verschiedene Stellen im System zu kopieren sind, ist eine automatische Installation sinnvoll. Apple stellt dafür einen passenden Mechanismus zur Verfügung: Softwareentwickler können ihre Programme zu so genannten Installationspaketen – Dateien mit

der Endung `.pkg` – zusammenschürren, die der Anwender dann durch einen Doppelklick installieren lassen kann. Ausgeführt wird die Installation vom Installationsprogramm, das Sie im Verzeichnis `/Programme/Dienstprogramme/` finden und das standardmäßig den `pkg`-Dateien zugeordnet ist. Vielleicht kennen Sie solche Installationspakete und den Ablauf der dialoggesteuerten Installation schon von Apples Betriebssystem-Updates.

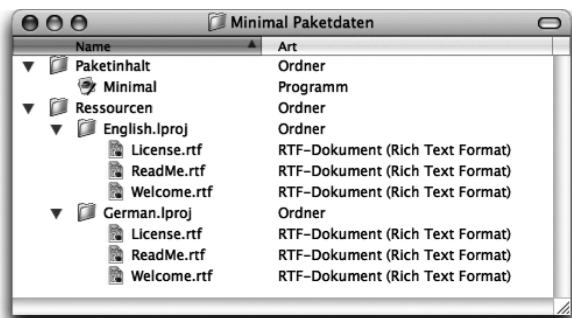


Abbildung 4.21 Verzeichnisstruktur für ein Installationspaket

Damit Sie ein eigenes Installationspaket erzeugen können, benötigen Sie zunächst eine Verzeichnisstruktur wie in Abbildung 4.21 dargestellt. In den Ordner »Paketinhalt« (der Name ist beliebig) legen Sie all die Dateien, die installiert werden sollen – das Programmpaket, die Dokumentation, einen URL-Clip für Ihre Webseite usw. Während in diesem Beispiel nur die Anwendung »Minimal« (bzw. das Programmpaket »Minimal.app«) installiert werden soll, könnten Sie hier auch ein Unterverzeichnis anlegen, das alle Ihre Dateien zusammenfasst.

Die Dateien im Ordner »Ressourcen« – dieser Name ist wieder beliebig, aber die Struktur und die Namen der darin enthaltenen Dateien und Verzeichnisse sind es *nicht* – werden vom Installationsprogramm ausgewertet und dem Anwender während der Installation angezeigt. Sie sehen hier zwei lokalisierte Ordner `English.lproj` und `German.lproj` jeweils mit den Dateien `Welcome`, `ReadMe` und `License`. Wenn die Dateien `Welcome` und `ReadMe` vorhanden sind, werden sie zu Beginn der Installation als Information angezeigt – probieren Sie es ruhig einmal mit dem Paket `Minimal.pkg` aus, das Sie auf der Buch-CD im Verzeichnis `/examples/ch04/packagemaker/` finden. Danach wird, falls vorhanden, die Lizenz eingeblendet, die vom Anwender bestätigt werden muss. Als Format dieser drei Texte können Sie beliebig zwischen ASCII-Texten (`*.txt`), RTF-Texten (`*.rtf`) und HTML-Dateien (`*.html`) wählen.

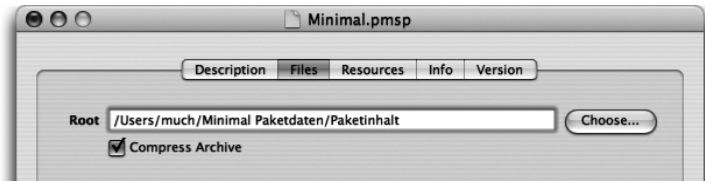


Abbildung 4.22 PackageMaker: Verzeichnis für den Paketinhalt setzen

Starten Sie nun das Programm `/Developer/Applications/Utilities/PackageMaker`, das aus dieser Verzeichnisstruktur ein Installationspaket generiert. Geben Sie dazu auf der Dialogseite »Description« einen Titel ein (hier »Minimal«) und wählen Sie dann auf der Seite »Files« das Verzeichnis »Paketinhalt« als Wurzelverzeichnis (»Root«) der zu installierenden Dateien (siehe Abbildung 4.22). Auf der Dialogseite »Resources« tragen Sie anschließend das Verzeichnis mit den Installationsressourcen ein (siehe Abbildung 4.23).

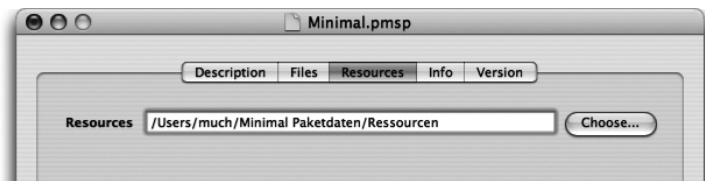


Abbildung 4.23 Verzeichnis für die Installationsressourcen setzen

Nun müssen Sie auf der Dialogseite »Info« noch festlegen, wo Ihre Software installiert werden soll (siehe Abbildung 4.24). Im Beispiel wurde »Applications« ausgewählt, das ist der `Programme-Ordner` (PackageMaker zeigt immer den ursprünglichen Namen an, auch wenn Sie den lokalisierten Namen ausgewählt haben). Sie können bestimmen, ob zur Installation Administrator-Rechte nötig sind und ob nach der Installation ein Neustart erfolgen soll. Beides sollte aber bei Java-Anwendungen, die nicht in das System eingreifen, unnötig sein. Im Beispiel ist noch die Option »Relocatable« markiert. Dadurch kann der Anwender während der Installation einen beliebigen Zielordner auswählen – ansonsten wäre er auf die Vorgabe bei »Default Location« beschränkt.

Nachdem Sie alle wesentlichen Einstellungen vorgenommen haben, rufen Sie den Menüpunkt **File • Create Package** auf, woraufhin der PackageMaker das Installationspaket in einem wählbaren Verzeichnis generiert. Anschließend können Sie noch **File • Save** aufrufen und eine `.psmp`-Datei mit den aktuellen PackageMaker-Einstellungen speichern. Das nächste Mal, wenn Sie dasselbe Installationspaket erzeugen wollen, müssen Sie dann nur noch diese Einstellungsdatei laden und brauchen nicht mehr alle Werte neu eingeben.



Abbildung 4.24 Zielverzeichnis setzen

Das gerade erzeugte Installationspaket `Minimal.pkg` können Sie nun doppelt anklicken. Das Installationsprogramm wird gestartet und führt Sie durch die einzelnen Schritte der Installation (siehe Abbildung 4.25). Typischerweise verteilen Sie die Installationspakete nicht »roh« an die Anwender, sondern Sie speichern das Paket in einem Disk Image, wie Sie es weiter vorne kennen gelernt haben.



Abbildung 4.25 Erzeugtes Installationspaket und Installationsprogramm

Apples Installationsmechanismus sieht auch so genannte Meta Packages (.mpkg) vor. Das sind Installationspakete, die sich aus mehreren Einzel-Installationspaketen zusammensetzen und die automatisch zusammen installiert werden. Wenn ein Paket dabei nicht zwingend nötig ist, kann der Anwender es bei der Installation optional auslassen.

Ebenso ist eine Versionierung vorgesehen, die darüber entscheidet, ob eine Software neu installiert oder ob nur eine Aktualisierung durchgeführt wird. Dafür merkt sich das System alle installierten Pakete im Verzeichnis `/Library/Receipts/` – für das Minimal-Paket taucht dort also eine Datei `/Library/Receipts/Minimal.pkg` auf. Die gespeicherten Pakete enthalten aber nur noch die Ressourcen mit den Versionsinformationen, die Programm-daten, die sonst viel Speicherplatz belegen würden, wurden daraus gelöscht.

Mehr Informationen zu Disk Images und Installationspaketen finden Sie in der PackageMaker-Hilfe, die Sie im Programm über das Hilfe-Menü aufrufen können, sowie auf der Seite <http://developer.apple.com/documentation/Developer-Tools/Conceptual/SoftwareDistribution/>.

4.4.2 Kommerzielle Installer

Obwohl Sie mit Disk Images und dem PackageMaker sehr gute Software-Distributionsmöglichkeiten für MacOS X haben, ist genau dies manchmal der größte Nachteil: Die Images und Installationspakete können nur auf MacOS X erzeugt werden und sind auch nur dort für eine Installation einsetzbar. Wenn Sie aber auf MacOS X Installer für andere Systeme generieren wollen oder auf anderen Systemen solche für den Mac, müssen Sie auf andere – in der Regel kostenpflichtige – Lösungen zurückgreifen.

Die Gemeinsamkeit der hier vorgestellten Programme ist, dass sie neben MacOS X meistens noch für Windows (und teilweise Linux bzw. UNIX-Varianten) verfügbar sind und für alle diese Zielplattformen Installationsprogramme generieren können. Dabei bieten sie umfangreiche Konfigurationsmöglichkeiten, die hier nicht alle vorgestellt werden können. Wenn Ihnen also einer der Installer zusagt, müssen Sie ihn selber noch genauer testen, was auch angesichts des meistens recht hohen Preises dringend angeraten ist. Kostenlose Demoversionen sind für alle diese Programme erhältlich.

Zero G InstallAnywhere

InstallAnywhere ist als Java-Programm für viele Plattformen verfügbar. Die »MacOS X Edition« ist gut an den Mac angepasst – der Installer-Generator selber ist eine als MacOS X-Applikation verpackte Java-Anwendung, ebenso die

erzeugten Installer. Der Generator kennt Mac-spezifische Eigenheiten wie Kodierungen und wichtige Verzeichnisse. Einzig die Benutzeroberfläche des Generators wirkt ein bisschen unaufgeräumt – dafür sehen die erzeugten Installer gut aus und zeigen alle wichtigen Informationen übersichtlich an. Schade ist, dass es für MacOS X nur Version 5.0 gibt, während für andere Systeme schon länger Release 6 verfügbar ist.

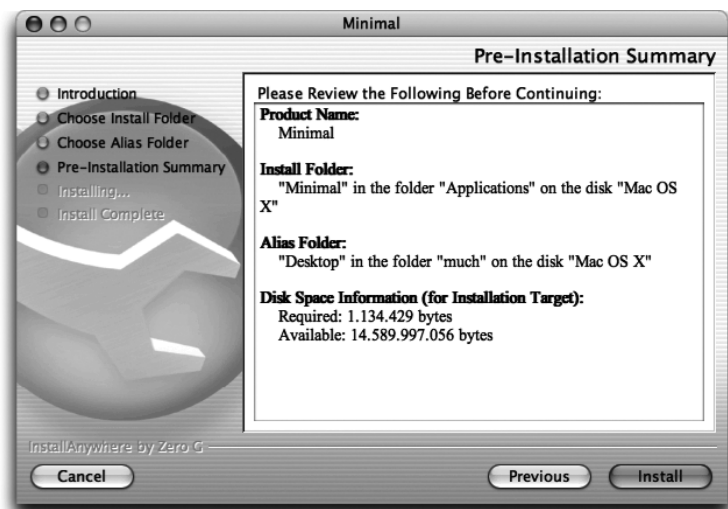


Abbildung 4.26 Ein von InstallAnywhere generierter Installer

- InstallAnywhere 5.0 Mac OS X Edition – ab ca. 1.600 €
<http://www.zerog.com/>

InstallShield X

Seit Version 5 kennt die »MultiPlatform«-Version von InstallShield auch Mac OS X. Auch hierbei handelt es sich um eine Java-Anwendung, die aber im Gegensatz zu InstallAnywhere nicht so schön ins System eingebunden ist – der Generator ist im Wesentlichen ein JAR-Archiv, das über ein (doppelklickbares) Shell-Skript gestartet wird. Zunächst läuft dies im Terminal ab und öffnet erst danach eine grafische Benutzeroberfläche. Die Oberfläche des Generators wirkt dann aber aufgeräumt und wurde bei der aktuellen Version »InstallShield X« nochmals vereinfacht. Mit der neuen Version kann InstallShield die Installationsprogramme nun auch als Mac OS X-Programmpakete erzeugen, so dass diese dann kein Shell-Skript mehr benötigen. Dazu müssen Sie die entsprechende »Mac OS X App Bundle«-Option aber erst in den erweiterten Konfigurationsmöglichkeiten einschalten (siehe Abbildung 4.27). Die erzeugten Installer sehen gut aus, auch wenn sie die Informationen nicht ganz so übersichtlich

wie Apples Installationsprogramm oder InstallAnywhere anzeigen. Der große Vorteil von InstallShield sind die vielen verfügbaren Sprachen (Lokalisierungen) sowie sehr viele Zielsysteme.

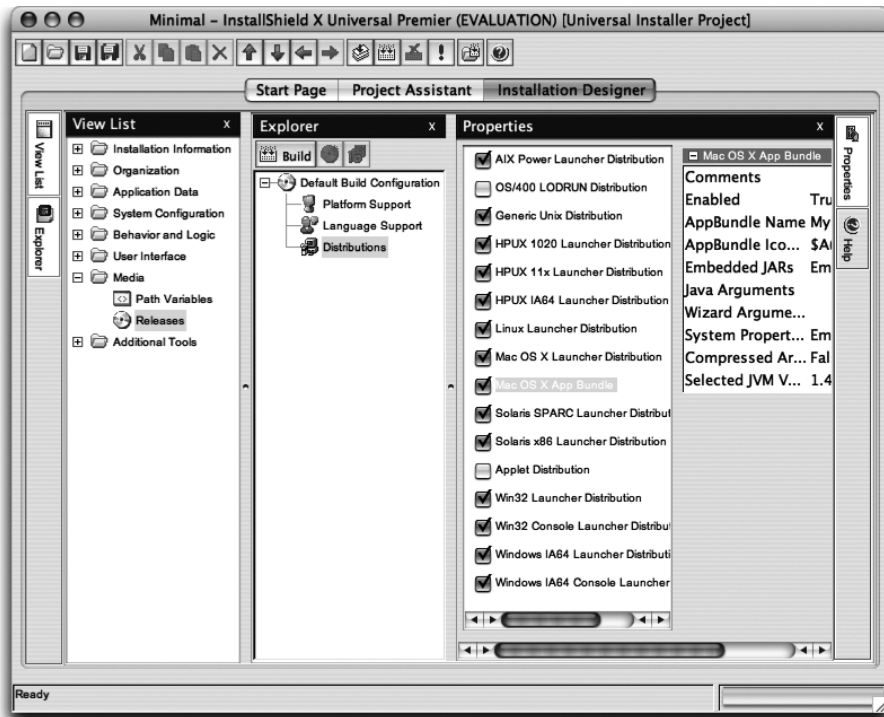


Abbildung 4.27 InstallShield X

- InstallShield X – ab ca. 1.200 US-\$
<http://www.installshield.com/products/x/>

ej-technologies install4j

install4j ist – obwohl es sich auch hier um einen plattformübergreifenden Installer-Generator handelt – hervorragend an Mac OS X angepasst. Die Oberfläche ist übersichtlich und schnell zu bedienen, trotzdem kann man viele Feineinstellungen vornehmen. Sollte man Einstellungen vergessen haben, wird man darauf hingewiesen, wo welche Angaben fehlen. Und auch ein »Test build« fehlt nicht, mit dem Sie prüfen können, ob alle zur Installation vorgesehenen Dateien auch tatsächlich gefunden werden. Die erzeugten Installer sind Mac OS X-Applikationen, die sogar standardmäßig automatisch als Stuffit-.sit-Archiv verpackt werden – so, wie es ein Mac-Anwender erwartet. Werkzeuge wie der Jar Bundler werden überflüssig, denn install4j kann von sich

aus MacOS X-Programmpakete aus Java-Klassen und -Archiven erzeugen! Dabei kann das zu installierende Programm entweder als Verzeichnis erzeugt werden, in dem sich alle Ressourcen und ein »leeres« Programmpaket (zum Starten) befinden – dies ist gut geeignet, wenn Sie Ihre Anwendung für mehrere Systeme in einem Archiv anbieten wollen. Oder es wird speziell für MacOS X ein komplettes Programmpaket inklusive aller Ressourcen generiert.

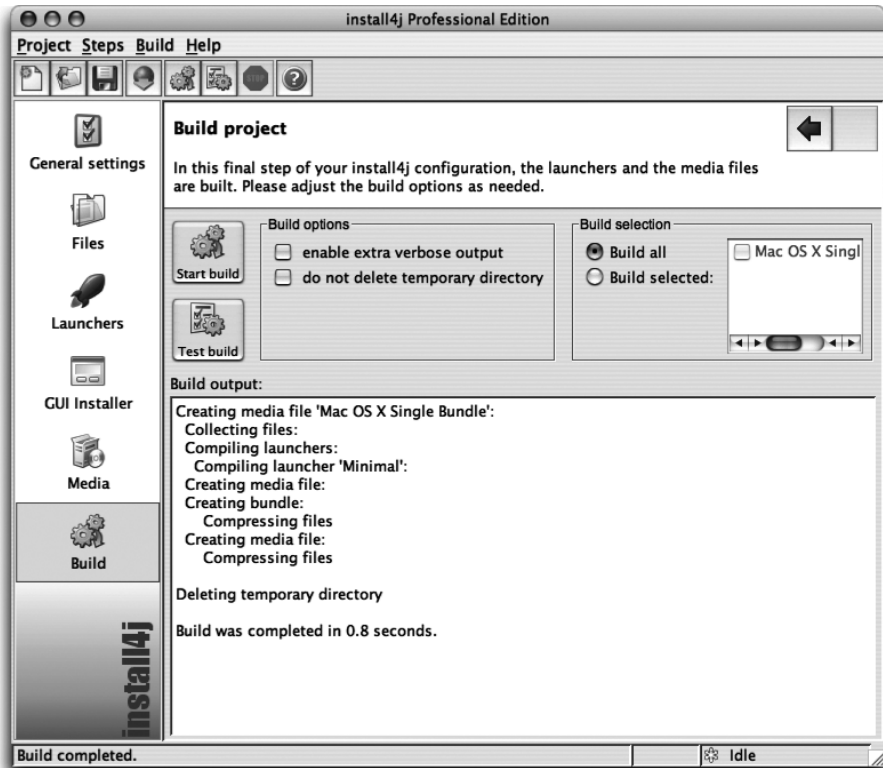


Abbildung 4.28 install4j 2.0.6

► install4j 2.0 – ab ca. 400 €

<http://www.ej-technologies.com/products/install4j/overview.html>

MindVision Installer VISE / VISE X

Installer VISE erzeugt Installationsprogramme nicht nur für MacOS X und Windows, sondern auch für ältere MacOS-Versionen. Diese lange Mac-Verbundenheit merkt man dem Programm an, denn es können sehr viele Mac-spezifische Einstellungen vorgenommen werden – auch wenn diese für Java-Anwendungen in der Regel unnötig sind. VISE X ist eine Neuentwicklung spe-

ziell für Mac OS X, aber auch hier finden Sie nach wie vor viele spezielle Mac-Konfigurationsmöglichkeiten. Die erzeugten Installer sehen typisch aus, in etwa so wie Apples Installationsprogramm.

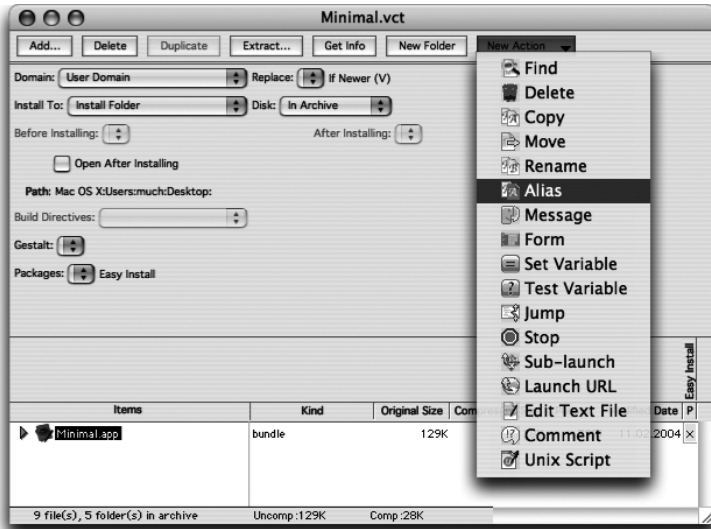


Abbildung 4.29 VISE X

- ▶ Installer VISE 8.3 for Macintosh – ab ca. 650 US-\$ Jahresgebühr
- ▶ VISE X 1.2 for Mac OS X – ab ca. 650 US-\$ Jahresgebühr
<http://www.mindvision.com/>

MindVision FileStorm

FileStorm ist ein Sonderfall der hier vorgestellten Installationsprogramme, denn zum einen ist dieses Programm nur auf Mac OS X verfügbar und kann auch nur dafür Installer erzeugen. Zum anderen kann die normale Version nur Disk Images erzeugen – das Generieren von Installationsprogrammen bleibt der »Pro«-Version vorbehalten. Das Erzeugen von Disk Images aber beherrscht FileStorm sehr gut, es bietet eine übersichtliche Oberfläche und ist recht preiswert. Außerdem können Sie dem Image sehr einfach eine Hintergrundgrafik verpassen, indem Sie geeignete Bilddaten auf das FileStorm-Fenster ziehen. Das Hintergrundbild kann dann beispielsweise Hinweise zur Drag & Drop-Installation anzeigen. Wenn Sie keinen Installer benötigen und damit auf die »Pro«-Version verzichten können, können Sie solche Disk Images aber auch komplett von Hand zusammenstellen, wie Sie es weiter vorne schon gesehen haben.

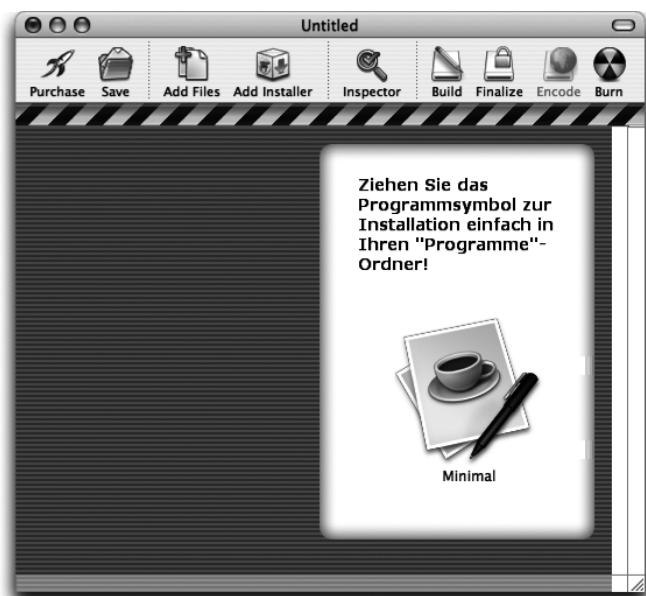


Abbildung 4.30 Disk Images mit FileStorm erzeugen

- ▶ FileStorm 1.7 (nur Disk Images) – ca. 20 US-\$
- ▶ FileStorm Pro 1.7 (auch Installer) – ca. 80 US-\$
<http://www.mindvision.com/filestorm.asp>

Aladdin Stuffit InstallerMaker

InstallerMaker kann für diverse Mac OS-Versionen Installationsprogramme erzeugen, Version 8 ist speziell an Mac OS X angepasst. Die Generator-Oberfläche wirkt nüchtern, ist aber funktionell. Dennoch hat man beim InstallerMaker noch mehr als bei VISE den Eindruck, dass er nicht so recht für Java-Anwendungen geeignet ist – zu sehr dominieren die speziellen Einstellungen für reine Mac-Anwendungen. Größtes Manko aber sind die generierten Installer, die eigentlich nur aus einem Dateiauswahl-Dialog für das Zielverzeichnis bestehen und recht antiquiert wirken.

- ▶ Stuffit InstallerMaker 8.0 – ab ca. 250 US-\$ Jahresgebühr
<http://www.stuffit.com/mac/installermaker/>



Abbildung 4.31 Stuft InstallerMaker

IzPack

IzPack passt eigentlich nicht so recht in diese Liste der kommerziellen Installationsprogramme, denn es ist Open Source und kostenlos! Der Generator-Installer selbst und die mit IzPack erzeugten Installer sind ganz gewöhnliche, doppelklickbare JAR-Archive. Die MacOS X-Integration ist gut, wenn auch noch nicht ganz vollständig – wichtige Systemverzeichnisse sind IzPack bekannt, Finder-Aliase können aber noch nicht erzeugt werden. Die Konfiguration von IzPack geschieht über eine XML-Datei, anhand der der Generator (ein Kommandozeilen-Tool, bei IzPack »Compiler« genannt) das Installations-JAR-Archiv erstellt.



Abbildung 4.32 Der mit IzPack erzeugte IzPack-Installer

- ▶ IzPack 3.6 – kostenlos
<http://www.izforge.com/izpack/>

4.5 Java Web Start

Mit MacOS X-Programmpaketen und Disk Images haben Sie eine gute Möglichkeit, Ihre Software an die Anwender zu verteilen. Die Anwender sind dann für die eigentliche Installation verantwortlich und müssen darauf achten, regelmäßig nach neuen Versionen zu schauen und diese bei Bedarf einzuspielen. Eine ganz andere Art der Software-Verteilung stellt daher Java Web Start (JAWS) dar. Eine Java Web Start-Anwendung sieht der Benutzer zunächst als einen ganz normalen Link im Web-Browser – ein Klick darauf startet die Applikation. Ob die Java-Ressourcen dafür erstmalig geladen werden müssen, als bereits vorhandene lokale Kopie genutzt werden können oder ob die lokale Kopie aktualisiert werden muss, darum kümmert sich Java Web Start automatisch. Wie funktioniert dies genau?

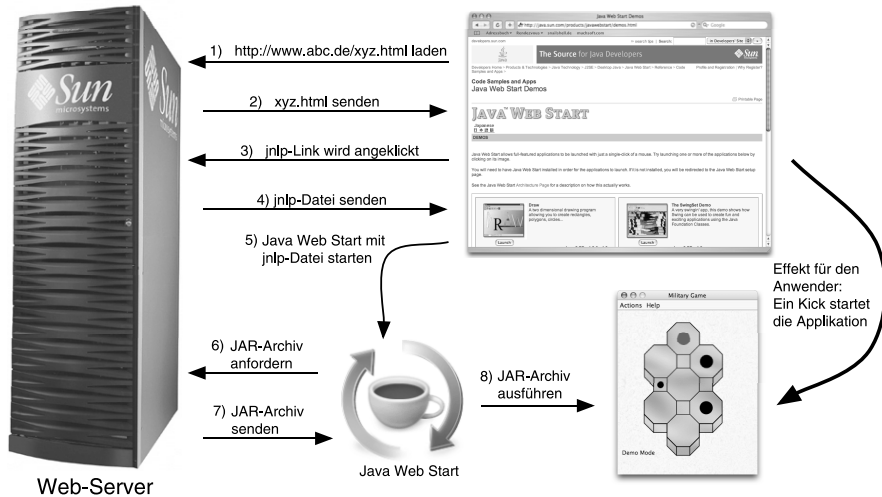


Abbildung 4.33 Funktionsweise von Java Web Start

Zunächst lädt der Anwender im Web-Browser ein HTML-Dokument von einem Web-Server (Schritte 1 und 2). In dem Dokument befindet sich ein Link auf eine so genannte JNLP-Datei mit der Dateinamenserweiterung `.jnlp`. Die Datei ist der zentrale Bestandteil von Java Web Start, sie ist die Konfigurationsdatei für die Java-Anwendung – und enthält unter anderem Angaben zu den benötigten JAR-Archiven, zur gewünschten JRE-Version und zum Programmicon. Ihr Aufbau ist in der Spezifikation des »Java Network Launching Protocol« (JNLP) festgelegt.

Klickt der Anwender den Link an, lädt der Browser die JNLP-Datei vom Web-Server herunter (Schritte 3 und 4). Mit dieser Datei ruft der Browser nun den

Java Web Start-Programm-Manager auf (Schritt 5), der entscheidet, ob die gewünschte Applikation bereits als lokale Kopie vorhanden und aktuell ist oder ob noch JAR-Archive herunter geladen werden müssen. Ist letzteres der Fall, lädt der Programm-Manager (und nicht der Browser!) die nötigen Daten vom Web-Server (Schritte 6 und 7), ansonsten kann das Laden übersprungen und die Applikation sofort gestartet werden (Schritt 8). Für den Benutzer zeigt sich der ganze Vorgang so, als ob der Klick auf den JNLP-Link die Java-Anwendung direkt gestartet hätte.

Während Java Web Start auf anderen Plattformen erst seit Java 1.4.1 fester Bestandteil ist und für ältere Java-Versionen separat heruntergeladen werden muss, gehört JAWS – und damit auch der Programm-Manager – seit Mac OS X 10.1 zum System dazu. Mit dem damaligen Java 1.3.1 wurde die Java Web Start Version 1.0.1 ausgeliefert. Das Java 1.4.1-Update brachte dann JAWS 1.2, und mit dem Java 1.4.2-Update steht das aktuelle JAWS 1.4.2 zur Verfügung.

Hier noch einmal alle wichtigen Vorteile von Java Web Start im Überblick:

- ▶ Der Einsatz (Deployment) und das Verwenden der Anwendungen ist sehr einfach – sie werden einfach von einem Web-Server geladen.
- ▶ Durch einen lokalen Zwischenspeicher (Cache) kann auch offline mit den Programmen gearbeitet werden, außerdem entfällt so das ständige Neuladen bei regelmäßiger Benutzung.
- ▶ JAWS bietet eine Desktop-Integration der Anwendungen. Der Benutzer muss dann die Programme nicht mehr über Browser-Links starten, sondern kann Programmsymbole auf dem Desktop verwenden.
- ▶ Die Anwendungen sind auf allen Systemen mit Java Web Start gut verwendbar und müssen nicht speziell angepasst werden (z.B. als Mac OS X-Applikation).
- ▶ Updates werden automatisch heruntergeladen und installiert. Dies geschieht entweder anhand des Datei-Änderungsdatums auf dem Server oder durch explizite Versionsangaben in der JNLP-Datei.
- ▶ Die Sicherheit des lokalen Rechners ist gewährleistet. JAWS-Applikationen laufen innerhalb einer so genannten »Sandbox« (Sandkasten), aus der sie nur eingeschränkten Zugriff auf die lokale Festplatte und das Netzwerk haben. Weitergehende Zugriffe sind nur durch Signierung des Programmcodes oder durch Verwendung der JNLP-API möglich.
- ▶ Sie können in der JNLP-Datei angeben, welche Java-Version mindestens vorhanden sein muss und welche Version idealerweise verwendet wird.

Java Web Start-Programme aufrufen

Nun wird es aber Zeit für ein Beispiel aus dem Web! Nehmen Sie einen Web-Browser Ihrer Wahl – beispielsweise Apples Safari (diverse Browser sind im nächsten Abschnitt über Applets beschrieben) – und rufen Sie die Seite <http://java.sun.com/products/javawebstart/demos.html> auf. Klicken Sie dort den JNLP-Link für das »Military Game« an. Die Konfigurationsdatei wird heruntergeladen und Java Web Start damit aufgerufen. Aus Sicherheitsgründen starten einige Browser den JAWS-Programm-Manager nicht sofort, sondern legen nur die JNLP-Datei auf dem Desktop (oder im jeweiligen Download-Ordner des Browsers) ab – der Anwender muss die JNLP-Datei dann noch explizit durch einen Doppelklick aufrufen.

Beim ersten Start dieser Java Web Start-Anwendung sehen Sie eine Fortschrittsanzeige für den Programm-Download (siehe Abbildung 4.34), danach wird das Java-Programm ganz normal ausgeführt.

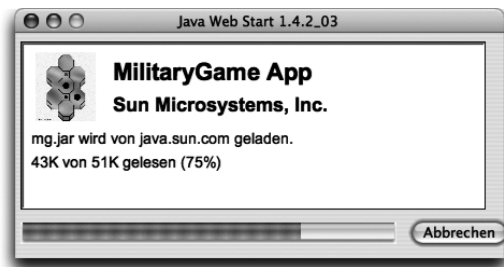


Abbildung 4.34 Eine JAWS-Anwendung wird erstmalig geladen.

Alle bisher verwendeten Java Web Start-Applikationen können Sie bequem im Programm-Manager verwalten, den Sie als Java Web Start im Verzeichnis `/Programme/Dienstprogramme/Java/` finden (siehe Abbildung 4.35). Wenn Sie dort unten links auf das Symbol klicken, können Sie ein Offline-Arbeiten erzwingen: Die Stecker-Symbole werden getrennt, und es können nun nur noch die JAWS-Programme ausgeführt werden, die komplett als lokale Kopie vorliegen und die in ihrer Konfigurationsdatei als offline-fähig markiert wurden. Die automatische Suche nach neuen Versionen funktioniert dann natürlich auch nicht mehr.

Sobald Sie eine Anwendung zweimal gestartet haben, fragt Java Web Start nach, ob für diese Applikation ein Programmkürzel erzeugt werden soll (siehe Abbildung 4.36). Wo dies gespeichert wird, können Sie sich in einem Dateiauswahldialog aussuchen. Erzeugt wird dann ein Mac OS X-Programmpaket, wie Sie es schon kennen gelernt haben. Das passende Programmsymbol wird dabei automatisch aus dem Icon der Java Web Start-Anwendung generiert.



Abbildung 4.35 Java Web Start Programm-Manager



Abbildung 4.36 Desktop-Integration nach mehrmaligem Start

Das Programmpaket speichert allerdings keinen Java-Code – die JAR-Archive werden trotzdem immer noch im Cache von Java Web Start verwaltet. Das hat den Vorteil, dass Sie die Anwendung nun zwar wie eine ganz normale MacOS X-Applikation verwenden können, die automatische Versionsverwaltung aber nach wie vor von JAWS durchgeführt werden kann. Das Programmpaket ist also quasi nur ein Alias für die JNLP-Datei. Realisiert wird dies über einen speziellen JNLP-Schlüssel im Java-Dictionary der Info.plist-Datei.

Das Erzeugen der Programmpakete war mit Java 1.4.1 fehlerhaft – die generierten Applikationen haben immer das JRE 1.3.1 verwendet, unabhängig davon, welche Version ursprünglich in der JNLP-Datei konfiguriert war. Java 1.4.2 generiert wieder korrekte Programmpakete – fehlerhafte Programme müssen Sie aber neu erzeugen lassen, um den Fehler zu beseitigen.

Wann der Programm-Manager ein Programm generieren möchte – nach dem ersten Start, nach dem zweiten, immer oder nie –, können Sie übrigens in den Einstellungen von Java Web Start festlegen.

Sofern Sie in den JAWS-Programm-Manager-Einstellungen nichts anderes konfigurieren, werden die heruntergeladenen Applikationen im Verzeichnis `~/Library/Caches/Java Web Start/` zwischengespeichert. Normalerweise müssen Sie aber niemals direkt auf dieses Verzeichnis zugreifen.

Java Web Start-Anwendungen erzeugen

Es ist relativ einfach, eigene Java Web Start-Applikationen zu erstellen – üblicherweise sind dies normale Java-Programme, die über die `main()`-Methode aufgerufen werden. Es gibt zwar auch Unterstützung für Applets (siehe nächster Abschnitt), um die Migration nach JAWS zu erleichtern, aber neue JAWS-Anwendungen sollten Sie nicht als Applet entwickeln.

Sie müssen Ihr Programm als JAR-Archiv verpacken und sollten diesem ein Manifest mit einem passenden `Main-Class`-Attribut geben.

Zusätzlich – und dies ist der aufwändigste Teil – müssen Sie eine JNLP-Datei schreiben, um Ihre Java Web Start-Applikation zu konfigurieren. Es handelt sich dabei um eine XML-Datei mit folgendem Aufbau:

```
<!-- CD/examples/ch04/javawebstart/Web-Sites/jwsdemo.jnlp -->
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"
  codebase="http://127.0.0.1/~much"
  href="jwsdemo.jnlp">
  <information>
    <title>Java Web Start Demo</title>
    <vendor>Thomas Much</vendor>
    <homepage href="jwsdemo.html"/>
    <description>Ein kleines Beispiel für JWS</description>
    <icon href="duke.gif"/>
    <offline-allowed/>
  </information>
  <resources>
    <j2se version="1.4+"/>
    <j2se version="1.3*"/>
    <jar href="JWSDemo.jar"/>
    <property name="apple.awt.brushMetalLook" value="true"/>
    <property name="apple.awt.showGrowBox" value="false"/>
  </resources>
```

```

    <application-desc
        main-class="com.muchsoft.macjava.jwsdemo.JWSDemo"/>
</jnlp>

```

Listing 4.6 JNLP-Datei jwsdemo.jnlp

Im Prolog der XML-Datei sehen Sie die UTF-8-Kodierung. Zur Zeit ist dies für Java Web Start zwingend nötig – nehmen Sie zur Bearbeitung also am einfachsten `/Programme/TextEdit`. Damit das Beispiel auf Ihrem Rechner funktioniert, müssen Sie beim `codebase`-Attribut in Zeile 4 die Zeichenkette »much« durch Ihren Benutzernamen ersetzen.

Bei den einzelnen XML-Elementen finden Sie folgende Attribute und Unterelemente:

▶ `jnlp`

- ▶ Das `spec`-Attribut gibt die Versionsnummer der JNLP-Spezifikation an und sollte 1.0 oder höher sein. Hier wird der Standardwert »1.0+« verwendet.
- ▶ Das `codebase`-Attribut legt für alle relativen URLs in dieser Datei fest, auf welche Basis sie sich beziehen. Für das Beispiel ist die Standardadresse des lokalen Rechners (»localhost«) eingetragen, `http://127.0.0.1/`, gefolgt von der Tilde und dem Benutzernamen. Der in Mac OS X eingebaute Web-Server bildet diese URL dann auf das Verzeichnis `Web-Sites/` des angegebenen Benutzers ab.
- ▶ Mit dem `href`-Attribut wird ein Rückverweis auf die JNLP-Datei selbst angelegt. Dies ist für den JAWS-Programm-Manager nötig. Hier ist nur die relative URL `jwsdemo.jnlp` angegeben, was mit obigem `codebase`-Attribut auf die Adresse `http://127.0.0.1/~much/jwsdemo.jnlp` abgebildet wird.

▶ `information`

Dieses Element enthält Informationen, die nicht den Java-Code, sondern die System-Integration betreffen. Bei den enthaltenen Textelementen dürfen Sie nur ganz normalen Text in der angegebenen Kodierung verwenden, HTML-Formatierungen sind nicht erlaubt.

- ▶ Das `title`-Element legt den Programmnamen für Ihre JAWS-Anwendung fest. Dieser Name wird nicht nur im Programm-Manager, sondern auch im Programm-Menü und im Dock angezeigt!
- ▶ Das `vendor`-Element gibt den Hersteller der Software an.
- ▶ Mit dem `href`-Attribut des `homepage`-Elements geben Sie die Webseite für Ihre Applikation an, auf der Sie weiterführende Informationen zum Pro-

gramm anbieten sollten. Der Anwender kann dann über einen Link im Programm-Manager die Seite sehr einfach aufrufen. Die hier angegebene relative URL wird zusammen mit dem vorher festgelegten `codebase`-Attribut auf die Adresse `http://127.0.0.1/~much/jwsdemo.html` abgebildet, aber Sie können natürlich auch eine absolute URL eintragen.

- ▶ Das optionale `description`-Element enthält eine kurze Beschreibung der Applikation. Bei diesem Element können Sie mit dem `kind`-Attribut festlegen, wie die Beschreibung verwendet werden soll (z.B. als Einzeiler oder als Absatz). Geben Sie das Attribut nicht an, wird die Beschreibung als Standardwert für alle möglichen Fälle verwendet.
- ▶ Das `icon`-Element enthält eine relative oder absolute HTTP-URL für ein GIF- oder JPEG-Bild. Die Grafik wird beim Start der Applikation, im Programm-Manager sowie als Programmsymbol verwendet und automatisch passend skaliert. Normalerweise werden die Größen 64x64 und 32x32 Pixel verwendet, mit den Attributen `width` und `height` können Sie die tatsächliche Breite und Höhe angeben. Wenn Sie das `icon`-Element ein zweites Mal mit dem Attribut `kind="splash"` verwenden, wird dieses zweite Bild beim Programmstart angezeigt (»Splash-Screen«). Wenn Sie gar kein Bild angeben, werden nur der Programmname und der Hersteller dargestellt.
- ▶ Mit dem Element `offline-allowed` legen Sie fest, ob Ihre Anwendung auch ohne Netzwerkverbindung gestartet werden darf. Die automatische Aktualisierung funktioniert trotzdem noch, sofern der Benutzer zufällig online ist. Wenn Sie dieses Element weglassen, muss zum Starten eine Netzwerkverbindung verfügbar sein – damit kann Java Web Start dann aber auch sicherstellen, dass garantiert immer die neueste Version ausgeführt wird.

▶ `resources`

Das `resources`-Element enthält Angaben zum Java-Code:

- ▶ Das `j2se`-Element legt mit seinem `version`-Attribut fest, welche Java-Version zur Ausführung verwendet werden muss. Das Element kann dabei mehrfach auftauchen und gibt dann die Priorität (in Reihenfolge der Auflistung) an, mit der die Java-Versionen verwendet werden. Hier würde also zunächst nach Java 1.4 oder neuer gesucht werden. Sollte keine passende Version vorhanden sein, würde sich die Applikation mit irgendeiner 1.3-Version zufrieden geben. (Die zwei Angaben wurden hier nur als Beispiel aufgeführt und könnten zu »1.3+« zusammengefasst werden.) Falls Sie vergessen, dieses Element einzutragen, startet Mac OS X die Anwendung mit Java 1.3.1.

- ▶ Das `jar`-Element gibt mit seinem `href`-Attribut an, welches JAR-Archiv die Anwendung benötigt. Baut die Applikation auf mehreren Archiven auf, führen Sie das `jar`-Element einfach mehrfach auf – für jedes JAR-Archiv eines.
- ▶ Mit `property`-Elementen können Sie System-Properties setzen, die Sie bei MacOS X-Applikationen in der `Info.plist`-Datei eintragen oder im Java-Code mit `System.setProperty()` setzen würden.
- ▶ `application-desc`
Nun müssen Sie nur noch angeben, mit welcher Klasse Ihre Applikation gestartet wird, d.h., wo sich die `main()`-Methode befindet. Wenn das erste bei den `jar`-Elementen aufgeführte JAR-Archiv ein Manifest mit einem passenden `Main-Class`-Attribut enthält, reicht das schon aus. Es schadet aber nicht, wenn Sie diese Klasse auch noch im `main-class`-Attribut des `application-desc`-Elements eintragen. Dann können Sie auch mit optionalen `argument`-Unterelementen Parameter an die `main()`-Methode übergeben.
- ▶ `applet-desc`
Das (hier nicht verwendete) `applet-desc`-Element ermöglicht es Ihnen, ein Java Applet als JAWS-Programm auszuführen.

Der Aufbau der JNLP-Datei ist bei Sun auf der Seite <http://java.sun.com/j2se/1.4.2/docs/guide/jws/developersguide/syntax.html> genauer beschrieben, aber alle wesentlichen Elemente haben Sie hier bereits kennen gelernt.

Das Setzen der Properties wie in der obigen JNLP-Datei birgt ein Problem: Änderungen von GUI-Eigenschaften werden teilweise nicht beachtet, da die Oberfläche schon von Java Web Start benutzt wird, bevor die JNLP-Datei ausgewertet wird – Oberflächen-Properties müssen aber gesetzt sein, bevor ein Programm das erste Mal auf GUI-Elemente zugreift. Dies ist kein MacOS X-spezifischer Bug, sondern ein generelles Problem von Java Web Start, das Sun durch eine geeignete Spezifikation lösen muss.

Nun benötigen Sie nur noch ein HTML-Dokument, das einen Link auf Ihre JNLP-Datei enthält:

```
<!-- CD/examples/ch04/javawebstart/Web-Sites/jwsdemo.html -->
<html>
<head>
  <title>Java Web Start Demo</title>
</head>
```

```

<body>
  <h1>Java Web Start Demo</h1>
  <p><a href="jwsdemo.jnlp">JWSDemo starten</a></p>
</body>
</html>

```

Listing 4.7 HTML-Datei jwsdemo.html (mit JNLP-Link)

Der Aufbau der HTML-Datei ist sehr einfach gehalten, aber ein Verweis mit dem `<A>`-Element¹⁰ auf die passende JNLP-Datei reicht wirklich aus, damit der Benutzer den Link im Web-Browser anklicken kann und die Java Web Start-Anwendung gestartet wird.

Wenn man mehr Aufwand betreiben möchte, kann man mit einem JavaScript-Programm innerhalb des HTML-Dokuments testen, ob Java Web Start überhaupt auf dem Rechner des Benutzers verfügbar ist, und nur dann einen Link auf der Webseite anzeigen. Falls Sie sich für diese Möglichkeit interessieren, finden Sie auf der Seite <http://java.sun.com/products/javawebstart/docs/developersguide.html#mime> weitere Informationen.

Java Web Start-Anwendungen einsetzen

Zum Testen der JAWS-Anwendung müssen Sie die benötigten Dateien – JNLP-Datei, HTML-Dokument, JAR-Archiv und Icon-Bilddatei – auf einem Web-Server ablegen. Üblicherweise ist dies ein Rechner im Internet. Weil aber jede Mac OS X-Installation einen kompletten Web-Server mitbringt (Apache 1.3; die genaue Version können Sie im Terminal mit `httpd -version` abfragen), verwenden wir einfach Ihren lokalen Rechner als Server.

Kopieren Sie dazu von der Buch-CD alle Dateien aus dem Verzeichnis `/examples/ch04/javawebstart/Web-Sites/` nach `~/Web-Sites/` in Ihr Benutzerverzeichnis (wenn Sie die Verzeichnisnamen auf Englisch anzeigen lassen, heißt dieser Ordner bei Ihnen ganz einfach nur `Sites`). Denken Sie daran, in der Datei `jwsdemo.jnlp` das `codebase`-Attribut des `jnlp`-Elements an Ihren Benutzernamen anzupassen.

Danach aktivieren Sie in den Systemeinstellungen den Apache-Web-Server. Auf der Dialogseite **Systemeinstellungen • Sharing • Dienste** können Sie das »Personal Web Sharing« starten (siehe Abbildung 4.37) und später auch wieder stoppen.

¹⁰ HTML ignoriert die Groß-/Kleinschreibung bei den Elementnamen, früher war Großschreibung üblich. Bei XHTML und XML ist dagegen die Schreibweise wichtig – meistens werden Kleinbuchstaben verwendet.

Wenn Sie den Web-Server regelmäßig an- und ausschalten, geht das einfacher und schneller mit dem kleinen Tool »SharingMenu«, das Sie von <http://www.mani.de/de/software/macosx/sharingmenu/> kostenlos herunterladen können.

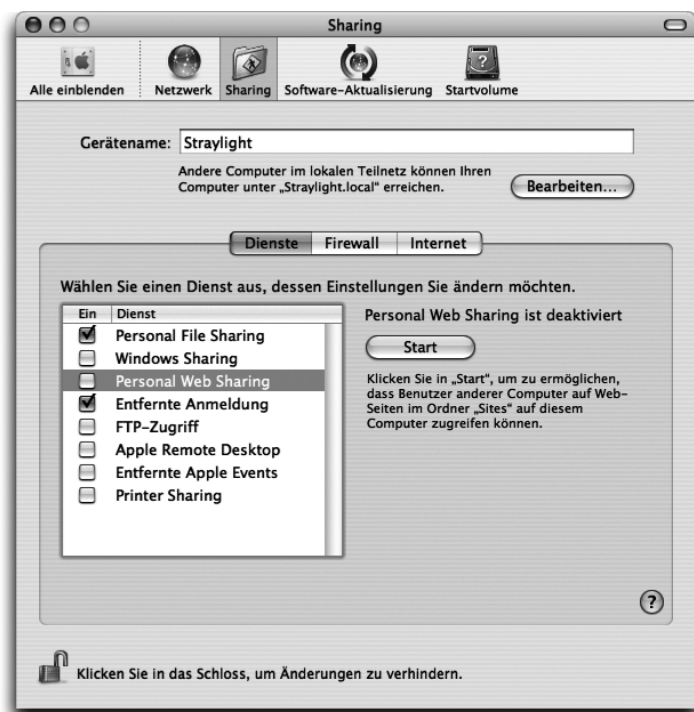


Abbildung 4.37 Web Sharing in den Systemeinstellungen aktivieren

Falls Sie noch Mac OS X 10.2 einsetzen, stellen Sie vorher bitte sicher, dass Apache überhaupt mit JNLP-Dateien umgehen kann. Dazu muss in der Datei `/etc/httpd/mime.types` ein passender MIME-Typ eingetragen sein. Wenn das folgende Kommando keine Rückgabe liefert, d.h. die zweite Zeile nicht angezeigt wird, müssen Sie die Datei selber erweitern:

```
[straylight:~] much% grep jnlp /etc/httpd/mime.types
application/x-java-jnlp-file    jnlp
```

Listing 4.8 Test, ob Apache korrekt für JNLP konfiguriert ist

Rufen Sie dazu am einfachsten im Terminal den Editor `pico` auf. Mit `sudo` bekommen Sie Administratorrechte, die Sie benötigen, um die Datei abzuspeichern zu können:

```
sudo pico /etc/httpd/mime.types
```

Dementsprechend werden Sie als erstes nach Ihrem Benutzerpasswort gefragt. Danach startet der Editor, und Sie können die Zeile

```
application/x-java-jnlp-file      jnlp
```

am Anfang der MIME-Typ-Liste hinzufügen. Drücken Sie dann `Ctrl`+`X` zum Beenden des Editors und bestätigen Sie mit `Y`, dass die Datei gespeichert werden soll.

Wenn Apache (oder allgemeiner, der von Ihnen verwendete Web-Server) nicht für JNLP-Dateien konfiguriert ist, kann es je nach Browser vorkommen, dass beim Klick auf den JNLP-Link nicht die Java Web Start-Applikation gestartet, sondern statt dessen die JNLP-Datei als Text im Browser angezeigt wird. Bei Mac OS X 10.3 sollte alles passend voreingestellt sein.

Sobald der Web-Server erfolgreich gestartet wurde, können Sie sich in einem beliebigen Browser das HTML-Dokument unter der Adresse `http://127.0.0.1/~much/jwsdemo.html` ansehen (statt »much« müssen Sie natürlich Ihren Benutzernamen eingeben). Wenn Sie darin auf den Link »JWSDemo starten« klicken, wird Java Web Start aufgerufen und die Applikation ausgeführt!

Zum richtigen Einsatz werden Sie JAWS-Anwendungen später nicht auf Ihrem lokalen Rechner installieren, sondern auf einem Web-Server im Internet. Da Sie die Dateien der Anwendung aber nicht mit einem Finder-Fenster dorthin kopieren können, benötigen Sie ein so genanntes FTP-Programm (»File Transfer Protocol«). Auf dem Mac wird dafür gerne »Transmit«¹¹ oder das kostenlose »Cyberduck«¹² eingesetzt. Alternativ steht Ihnen auch der Kommandozeilenbefehl `ftp` zur Verfügung.

Zugriff auf Ressourcen

Bei Java Web Start-Applikationen besteht der Klassenpfad nicht aus Pfadangaben für das lokale Dateisystem, sondern wird aus den in der JNLP-Datei aufgeführten JAR-Archiven zusammengesetzt. Alle Ressourcen, die Sie in Ihrer Applikation verwenden – Bilder, Konfigurationsdateien usw. –, müssen sich daher in einem der JAR-Archive befinden, wenn Sie nicht jede Ressource einzeln und zeitaufwändig vom Web-Server nachladen wollen. Weil zum Laden der Archive außerdem ein spezieller Class-Loader eingesetzt wird, können Sie

11 Download von <http://www.panic.com/transmit/>

12 Download von <http://www.cyberduck.ch/>

nicht mit den üblichen Techniken¹³ auf diese Ressourcen zugreifen, sondern müssen immer genau den Class-Loader verwenden, der auch Ihre Applikation geladen hat. Das folgende Beispiel zeigt, wie Sie Bilddateien korrekt aus einem JAR-Archiv laden:

```
//CD/examples/ch04/javawebstart/JWSDemo/JWSDemo.java
ClassLoader cl = this.getClass().getClassLoader();
Icon bild1 = new ImageIcon(cl.getResource("images/duke.gif"));
Icon bild2 = new ImageIcon(cl.getResource("images/galileo_
logo.gif"));
```

Den Class-Loader können Sie alternativ auch mit `Thread.currentThread().getContextClassLoader()` ermitteln. Wichtig ist nur, dass dies im Hauptthread stattfindet, also in dem Thread, der die `main()`-Methode ausführt.

Das Beispiel setzt voraus, dass sich die Bilder innerhalb des JAR-Archivs im Verzeichnis `/images/` befinden – das ist ein typischer Anwendungsfall, damit die Dateien gut strukturiert gespeichert sind und nicht alle im Wurzelverzeichnis herumliegen. In Xcode können Sie das gewünschte Zielverzeichnis recht einfach in den Target-Einstellungen setzen. Wählen Sie dazu bei den »Build Phases« den Schritt »Java Resources« an. In der Tabelle können Sie dann in der Spalte »Destination Subdirectory« zu jeder Datei den gewünschten Pfad eintragen, nachdem Sie doppelt in die entsprechende Zelle geklickt haben (siehe Abbildung 4.38).

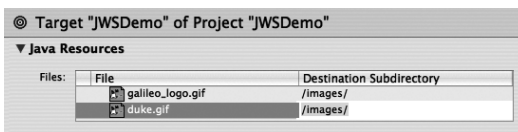


Abbildung 4.38 Pfad für JAR-Ressourcen in Xcode festlegen

Sicherheit

Da Java Web Start-Anwendungen aus dem Internet geladen werden und somit aus einer unsicheren Quelle stammen, werden sie in einer speziellen Umgebung mit eingeschränkten Rechten ausgeführt, in der so genannten »Sandbox«. Folgende Punkte müssen Sie beachten:

- ▶ Die Programme haben keinen Zugriff auf das lokale Dateisystem.
- ▶ Alle JAR-Archive müssen vom selben Server heruntergeladen werden.
- ▶ Netzwerkverbindungen sind nur zu dem Rechner erlaubt, von dem auch die Archive geladen werden.

¹³ Beispielsweise mit `ClassLoader.getSystemClassLoader()` oder mit `Class.forName()`

- ▶ Sie können keinen Security-Manager installieren.
- ▶ JNI-Bibliotheken können nicht verwendet werden.
- ▶ Auf die System-Properties haben Sie nur eingeschränkten Zugriff. Schreib- und Lese-Zugriff besteht auf alle in der JNLP-Datei definierten Properties, Nur-Lese-Zugriff haben Sie auf die Properties `file.separator`, `java.class.version`, `java.vendor`, `java.vendor.url`, `java.version`, `line.separator`, `os.arch`, `os.name` und `path.separator`. Alle anderen Properties können nicht genutzt werden.¹⁴

Erlaubt ist dagegen das Beenden der Anwendung mit `System.exit()`, denn im Gegensatz zu den im nächsten Abschnitt vorgestellten Applets ist dies für JAWS-Applikationen die einzige Möglichkeit, sich korrekt zu beenden.

Falls Ihr Programm dennoch Zugriff auf das Dateisystem benötigt, müssen Sie alle JAR-Archive der Applikation mit einem digitalen Zertifikat **signieren**. Java Web Start kann dann zum einen sicherstellen, dass der Programmcode nicht böswillig manipuliert wurde – der Anwender weiß also, dass er unveränderten Code einsetzt. Zum anderen wird dem Benutzer vor dem Ausführen einer so signierten Applikation ein Dialog angezeigt, von wem das Zertifikat stammt und ob er diesem Zertifikat vertraut. Damit der Zugriff auf das System schließlich funktioniert, müssen Sie noch folgende Zeilen in die JNLP-Datei (als Unter-element von `<jnlp>`) einfügen:

```
<security>
  <all-permissions/>
</security>
```

Listing 4.9 Freischalten des System-Zugriffs für signierte JAR-Archive

Wenn Sie nur diesen Text eintragen, die JAR-Archive aber nicht signieren, erhalten Sie beim Start eine Fehlermeldung, und das Ausführen wird abgebrochen!

Zum Testen können Sie sich im Terminal ein eigenes Zertifikat erzeugen. Dazu verwenden Sie das Programm `keytool`, das für einen Zertifikat-Herausgeber (hier »thmch«) ein Zertifikat generiert und in einer speziellen Keystore-Datei (hier »meinKeystore«) ablegt:

```
keytool -genkey -keystore meinKeystore -alias thmch
```

Sie werden nach Ihrem Namen, Ihrer Organisation, Ihrer Stadt usw. gefragt, was als Beschreibung für das Zertifikat verwendet wird. Außerdem müssen Sie

¹⁴ <http://java.sun.com/docs/books/tutorial/applet/practical/properties.html>

noch ein Passwort vergeben, das Sie später benötigen, um Archive mit Ihrem Zertifikat signieren zu können. Ob das Erzeugen des Zertifikats geklappt hat, können Sie wieder mit `keytool` feststellen:

```
[straylight:~] much% keytool -list -keystore meinKeystore
Geben Sie das Keystore-Passwort ein: *****
Keystore-Typ: jks
Keystore-Provider: SUN
Ihr Keystore enthält 1 Eintrag/-äge:
thmuch, 18.01.2004, keyEntry,
Zertifikatsfingerabdruck (MD5): 3B:EB:47:17:07:6E:FC:8C:AD:D1:40
:95:7C:CE:3D:9F
```

Listing 4.10 Keystore mit einem selbst signierten Zertifikat

Nun können Sie das JAR-Archiv mit dem Programm `jarsigner` signieren:

```
jarsigner -keystore meinKeystore JWSDemo.jar thmuch
```

Fertig! Wenn Sie nun die JNLP-Datei aufrufen, prüft Java Web Start, ob das Archiv seit der Signierung nicht verändert wurde, und startet dann die Anwendung. Vorher wird allerdings der Anwender darauf hingewiesen, dass diese Applikation vollen Systemzugriff haben möchte, und abhängig vom Zertifikat kann der Benutzer dann entscheiden, ob er dies gewährt. In Abbildung 4.39 sehen Sie, dass bei selbst signierten Zertifikaten die ausdrückliche Warnung erscheint, den Code nicht auszuführen. Daher sollten Sie für den produktiven Einsatz ein kommerzielles Zertifikat erwerben, das von diversen so genannten *Certificate Authorities* (CA) herausgegeben wird.

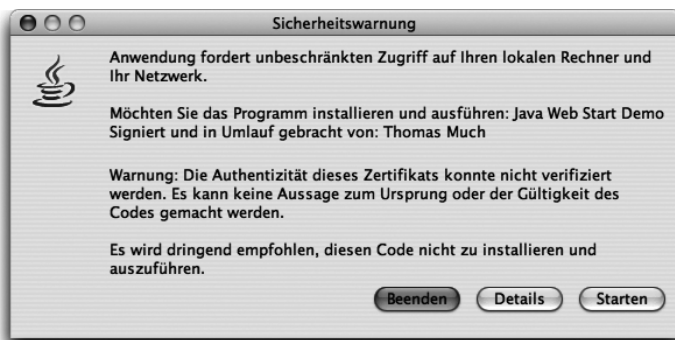


Abbildung 4.39 Sicherheitswarnung bei selbst signierten Zertifikaten

Ausführlicher ist das Erzeugen von Zertifikaten bei Sun auf der Seite <http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html> beschrieben.

Dort finden Sie auch einen Link zur `jarsigner`-Dokumentation. Und wie immer können Sie diese Informationen auch im Terminal mit dem `man`-Befehl abfragen.

Wenn das Signieren nicht in Frage kommt oder Sie gar nicht vollen Zugriff auf das System benötigen, bietet die **JNLP-API** einzelne, ganz spezielle und kontrollierte Zugriffsmöglichkeiten, die nicht Bestandteil der normalen J2SE-API sind. Die Funktionen sind in etwa vergleichbar mit dem, was ein Web-Browser mit Cookies und HTML-Formularen erreichen kann. Das Anzeigen einer URL im Browser sieht beispielsweise wie folgt aus:

```
import javax.jnlp.*;
// ...
try {
    BasicService bs =
        (BasicService)ServiceManager.lookup("javax.jnlp.BasicService");
    bs.showDocument("http://www.galileocomputing.de/");
}
catch(UnavailableServiceException e) {
    // Dienst steht nicht zur Verfügung
}
```

Zunächst wird also der `ServiceManager` nach einem bestimmten Dienst der JNLP-API gefragt, hier `javax.jnlp.BasicService`. Wenn dieser Dienst gefunden wurde, kann man damit die Methode `showDocument()` ansprechen. Das Interface `BasicService` bietet grob dieselben Möglichkeiten, die Sie im folgenden Abschnitt bei den Applets mit dem `AppletContext` kennen lernen werden.

Damit Sie Quelltext, der die JNLP-API verwendet, kompilieren können, müssen Sie sich das »JNLP Developer's Pack« von <http://java.sun.com/products/javawebstart/download-jnlp.html> herunterladen. Das darin enthaltene Archiv `jnlp.jar` müssen Sie beim Übersetzen auf den Klassenpfad setzen. Falls Sie die Kommandozeilenbefehle verwenden, sieht das also wie folgt aus:

```
javac -classpath .:jnlp.jar *.java
```

Zum Ausführen der Applikation wird das JAR-Archiv nicht mehr benötigt. Wenn obiger Code dann tatsächlich durchlaufen wird und sicherheitskritisch ist, erhält der Anwender einen Hinweis, ob er dies zulassen möchte. In Abbildung 4.40 hat ein Programm z.B. versucht, lesend auf die Zwischenablage (das *Clipboard*) zuzugreifen. Apple stellt auf der Seite <http://developer.apple.com/java/javawebstart/> ein kleines Testprogramm dafür zur Verfügung.

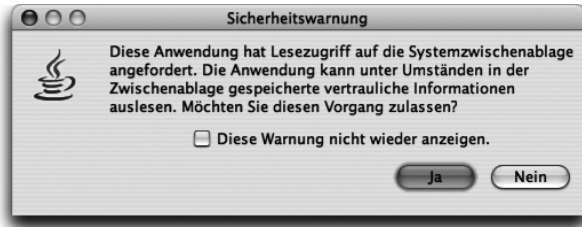


Abbildung 4.40 Sicherheitswarnung durch die JNLP-API

Folgende Interfaces und Dienste sind in der JNLP-API 1.0.1 vorhanden, deren Namen weitestgehend selbsterklärend sein sollten:

- ▶ BasicService
- ▶ ClipboardService
- ▶ DownloadService
- ▶ FileOpenService
- ▶ FileSaveService
- ▶ PrintService
- ▶ PersistenceService

Die einzelnen JNLP-API-Dienste sind mit kurzen Beispielen auf der Seite <http://java.sun.com/j2se/1.4.2/docs/guide/jws/developersguide/examples.html> beschrieben.

Mac OS X-Besonderheiten

Die Java Web Start-Implementierung von Mac OS X hält sich an die aktuelle Spezifikation, allerdings gibt es zwei Besonderheiten zu beachten:

- ▶ Die Windows- und die Solaris-Version unterstützen das Herunterladen zusätzlicher Java-Versionen (JREs). Das ist praktisch, wenn die JAWS-Anwendung eine bestimmte Version voraussetzt, diese aber nicht auf dem Rechner installiert ist. Bei Mac OS X entscheidet aber leider alleine Apple, welche Java-Versionen auf Ihrem System zur Verfügung stehen – derzeit Java 1.3.1 und eine 1.4er-Version. Wenn Ihre Applikation also beispielsweise Java 1.4.2 benötigt, müssen Sie die Anwender auf Ihrer Webseite darauf hinweisen, dass sie sich entweder über die Software-Aktualisierung oder von <http://www.apple.com/downloads/macosx/apple/> die passende Java-Version herunterladen und installieren.

Achten Sie unbedingt darauf, niemals eine bestimmte Version vor 1.3.1 vorauszusetzen – Java 1.2 gibt es für Mac OS X ganz einfach nicht. Versions-

angaben, die neuere JREs mit einschließen, beispielsweise »1.2+«, funktionieren aber problemlos.

- ▶ Es ist weder nötig noch möglich, im Programm-Manager Proxy-Einstellungen für den Internet-Zugang vorzunehmen. Diese Informationen besorgt sich Java Web Start automatisch aus dem Netzwerk-Dialog in den Systemeinstellungen.

Weitere Informationen zu Java Web Start finden Sie auf folgenden Webseiten:

- ▶ <http://java.sun.com/products/javawebstart/developers.html>
- ▶ <http://java.sun.com/products/javawebstart/faq.html>
- ▶ <http://www.vamphq.com/jwsfaq.html>

4.6 Applets

Wie Java Web Start-Applikationen werden Applets von einem Server geladen und dann in der Java-Laufzeitumgebung des Clients ausgeführt. Während bei ersterem der Programmcode von Java Web Start geladen und gestartet wird, dann aber als eigenständige Applikation läuft, werden Applets von einem Web-Browser geladen und dann vom Browser selbst als Teil eines HTML-Dokuments ausgeführt.

Applets sind die ursprüngliche Idee hinter Suns Java-Motto »Write once, run anywhere« (einmal schreiben und kompilieren, überall ausführen) – es sollte möglich sein, Java-Programme auf jeder Plattform, die einen Web-Browser mit einer Java Virtual Machine besaß, einfach und anwenderfreundlich auszuführen. In den vergangenen Jahren sind Applets aber etwas in Ungnade gefallen, denn es gab zu viele Probleme auf Anwenderseite. Die Browser griffen auf unterschiedliche, teils veraltete, teils fehlerhafte Java-Implementationen zurück. Zudem mussten die Applets je nach Browser mit unterschiedlichem HTML-Code in die Webseite eingebunden werden. Das Ergebnis dieser Inkompatibilitäten war dann oft, dass ein Applet gar nicht oder nur eingeschränkt funktionierte – ein für den Anwender nicht akzeptabler Zustand. Mittlerweile sind die Browser bezüglich Java zwar weitestgehend kompatibel, aber auf Internet-Webseiten befinden sich Applets immer noch auf dem Rückzug, denn in der Zwischenzeit konnten sich andere Techniken – Java Server Pages und Servlets, die auf einem Server laufen und nur noch reinen HTML-Code zum Client-Browser schicken, sowie neuerdings Java Web Start – etablieren. Nach wie vor werden Applets dagegen auf Intranet-Webseiten eingesetzt, wo die Systemadministratoren sicherstellen können, dass alle Anwender denselben Browser mit derselben Java-Version verwenden.

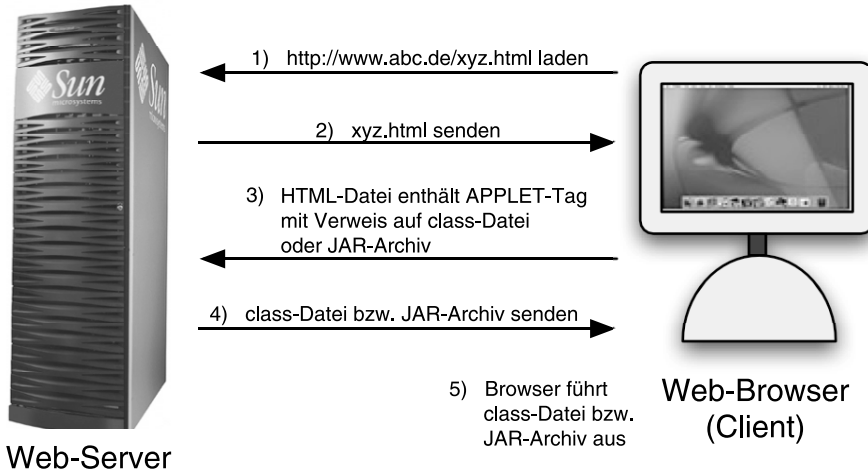


Abbildung 4.41 Wie Web-Browser Java-Applets einbinden

Auch wenn es Unterschiede gibt, wer letztendlich den Java-Code auf dem Client-Computer ausführt, ist die Funktionsweise von Applets recht ähnlich zu der von Java Web Start-Applikationen. Zunächst lädt der Anwender eine Webseite (ein HTML-Dokument) in seinem Browser (Schritte 1 und 2). Im Dokument befindet sich ein APPLET-Element, das einen Verweis auf eine Java-Klasse oder ein JAR-Archiv enthält. Der Browser lädt nun diese Klasse bzw. dieses Archiv (Schritte 3 und 4) und führt den Java-Code in einer Java Virtual Machine aus – die Anzeige des Applets erfolgt dabei innerhalb des HTML-Dokuments im Browser.

4.6.1 Applets programmieren und einsetzen

Applets werden im HTML-Dokument des Web-Browsers ausgeführt und somit nicht über die Kommandozeile oder anderswie als eigenständige Applikation gestartet. Sie besitzen keine `main()`-Methode, sondern erben von bestimmten Klassen, in denen Sie die passenden Initialisierungsmethoden überschreiben können. Applets mit AWT-Oberfläche erben von `java.applet.Applet`, Swing-Applets sollten `javax.swing.JApplet` als Oberklasse einsetzen. Beide Applet-Klassen sind dabei ganz normale Container (genauer: Unterklassen von `Panel`), die Sie schon in Kapitel 3, *Grafische Benutzungsoberflächen*, bei der Programmierung von Benutzungsoberflächen verwendet haben – die Dialogkomponenten werden also mit `add()` zum Applet hinzugefügt. Das folgende Listing zeigt Ihnen ein kleines AWT-Applet. Ein Beispiel für ein Swing-Applet finden Sie am Ende des Abschnitts bei der Applet-Browser-Kommunikation.

```

//CD/examples/ch04/applets/AppletDemo/AppletDemo.java
package com.muchsoft.macjava.appletdemo;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;

public class AppletDemo extends Applet {
    private Label labelZeit;

    public void init() {
        this.setLayout( new BorderLayout() );
        Panel panelTexte = new Panel( new GridLayout(3,2) );
        Button buttonAktualisieren = new Button("Aktualisieren");
            labelZeit          = new Label(new Date().toString());

        panelTexte.add(new Label("java.version:"));
        panelTexte.add(new Label(System.getProperty("java.version")));
        panelTexte.add(new Label("mrj.version:"));
        panelTexte.add(new Label( System.getProperty("mrj.version")));
        panelTexte.add(new Label("Zeit:"));
        panelTexte.add(labelZeit);

        this.add(panelTexte,          BorderLayout.CENTER);
        this.add(buttonAktualisieren, BorderLayout.SOUTH );

        buttonAktualisieren.addActionListener( new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                labelZeit.setText( new Date().toString() );
            }
        });
    }
}

```

Listing 4.11 Ein einfaches AWT-Applet

Sie sehen, dass diese Applet-Klasse keinen Konstruktor besitzt. Das wäre zwar durchaus erlaubt, aber leider ist die Applet-Umgebung (der so genannte `AppletContext`) im Konstruktor noch nicht richtig initialisiert, daher greift man lieber auf die Methode `init()` zurück. Diese Methode wird vom Browser aufgerufen, sobald das Applet-Objekt und die Applet-Umgebung erzeugt und

initialisiert wurden. Hier können Sie dann die Oberfläche des Applets zusammenbauen und weitere Initialisierungen vornehmen. `init()` gehört mit den Methoden `start()`, `stop()` und `destroy()` zu den Lebenszyklus-Methoden eines Applets. Die genaue Funktionsweise dieser und weiterer Methoden der Applet-API hat Sun auf der Seite <http://java.sun.com/docs/books/tutorial/applet/> dokumentiert.

Beachten Sie, dass bei diesem Beispiel am Anfang von `init()` zunächst ein neuer Layout-Manager gesetzt wird. Dies ist nur nötig, wenn Sie das Standardlayout nicht verwenden möchten – Applets mit der Oberklasse `java.applet.Applet` besitzen nämlich im Gegensatz zu Frames als Standardlayout ein `FlowLayout`.

Während hier ein typischer AWT-Dialog konstruiert wird, ist es bei Applets – wie generell bei allen AWT-Komponenten – alternativ möglich, die `paint()`-Methode zu überschreiben. Diese Methode wird von der Laufzeitumgebung zum passenden Zeitpunkt aufgerufen, damit sich die jeweilige Komponente selbst zeichnen kann. Mithilfe eines `Graphics`- oder `Graphics2D`-Kontexts können Sie dort eigene Grafiken mit Linien, Rechtecken, Bildern, Texten usw. malen, wie der folgende Code-Ausschnitt illustriert:

```
import java.awt.*;
import java.awt.geom.*;
// ...
public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    Dimension d = this.getSize(); // Größe dieser Component in Pixel
    g2.draw( new Line2D.Double( 5, 5, d.width - 5, d.height - 5 ) );
}
```

Listing 4.12 Überschriebene `paint()`-Methode

Auch wenn Applets aus vielen Klassen bestehen können, reicht für dieses Beispiel schon eine einzige Klasse als Programmcode. Nun benötigen Sie nur noch ein HTML-Dokument, das diesen Code einbindet, damit ein Browser die Beziehung zwischen Webseite und Java-Applet herstellen kann. Weitere Konfigurationsdateien sind normalerweise nicht nötig, es sei denn, Sie möchten die Sicherheitseinstellungen eines Applets verändern (s. u.).

Wenn Applets aus mehreren Klassen bestehen und weitere Ressourcen mitbringen (Bilder usw.), werden sie normalerweise als JAR-Archiv verteilt, damit nicht alle Teile einzeln (und zeitintensiv) vom Server heruntergeladen werden müssen. Im Folgenden wird das Beispielapplet daher im `<APPLET>`-Element mit

dem ARCHIVE-Attribut eingebunden – dies ist auch der Standard bei Applet-Projekten in Xcode.

```
<!-- CD/examples/ch04/applets/Web-Sites/appletdemo.html -->
<html>
<head>
  <title>Applet Demo</title>
</head>
<body>
<h2>Applet mit dem APPLELET-Tag eingebunden:</h2>

<applet
  archive="AppletDemo.jar"
  code="com.muchsoft.macjava.appletdemo.AppletDemo.class"
  width="300"
  height="100">
Der Browser unterst&uuml;tzt Java nicht oder Java wurde
in den Browser-Einstellungen deaktiviert.
</applet>

<!-- ... -->
</body>
</html>
```

Listing 4.13 HTML-Datei zum Einbinden eines Applets mit dem APPLELET-Element

Da im Quelltext das `package`-Statement verwendet wird, ist beim `CODE`-Attribut der vollqualifizierte Klassenname inklusive Paketname eingetragen, ansonsten verwenden Sie hier einfach nur den Klassennamen. Oft wird das `».class«` am Ende weggelassen, aber es schadet nicht, den Dateinamen des gewünschten Java-Codes komplett anzugeben. Falls Sie Ihr Applet nicht als JAR-Archiv ausliefern, lassen Sie das `ARCHIVE`-Attribut einfach weg.

Sie sehen, dass alle nötigen Parameter als Attribute im `<APPLET>`-Element angegeben werden. Zwischen dem Start- und Ende-Tag des Elements befindet sich nur noch Text, der angezeigt wird, falls das Applet nicht ausgeführt werden kann, beispielsweise weil Java in den Browser-Einstellungen deaktiviert wurde. Dort – zwischen den Tags, vor dem Text – können Sie bei Bedarf eine Liste von `<PARAM>`-Elementen (ohne schließendes Tag) nach folgendem Muster eintragen:

```
<param name="farbe" value="blau">
```

Innerhalb des Applets können Sie dann mit der `Applet`-Methode `getParameter("farbe")` auf diese Werte zugreifen und erhalten in diesem Fall »blau« als Rückgabe (oder `null`, falls der Name mit keinem Wert belegt ist).

Das obige HTML-Dokument setzt voraus, dass sich die Applet-Klasse (bzw. das JAR-Archiv) und das Dokument im selben Verzeichnis des Web-Servers befinden. Soll der Java-Code aus einem anderen Verzeichnis geladen werden, dürfen Sie nicht etwa den gewünschten Pfad in das `ARCHIVE`- bzw. `CODE`-Attribut einfügen, sondern müssen das Verzeichnis mit dem `CODEBASE`-Attribut festlegen!

Die HTML-Datei können Sie zum Testen nun einfach auf das Programmsymbol eines Web-Browsers ziehen. Der Browser lädt das Dokument von der Festplatte, zeigt die HTML-Seite an, lädt dann das eingebundene Applet und startet es (siehe Abbildung 4.42).

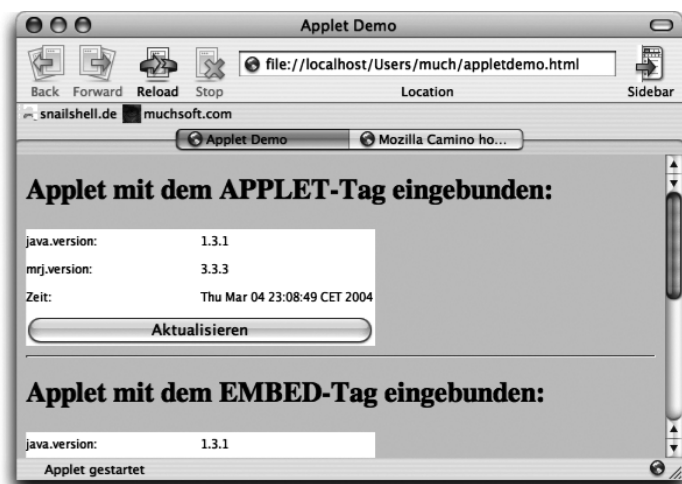


Abbildung 4.42 Ein Applet im Web-Browser Camino

Weil das `<APPLET>`-Element vor einigen Jahren (vor Java 1.3) Probleme hatte, aktuelle Java-Versionen anzusprechen, und insbesondere das Java-Plugin damit nicht verwendet werden konnte, wurden andere HTML-Elemente entwickelt, mit denen die Browser die Applet-Java-Umgebung flexibler einbinden konnten. Netscape erfand das `<EMBED>`-Element, das HTML-Standardisierungsgremium (»W3C«) das `<OBJECT>`-Element. Moderne Browser werten nun auch `<APPLET>` flexibel aus – Sun empfiehlt daher mittlerweile wieder, dieses Element einzusetzen –, kommen aber unterschiedlich gut mit den beiden anderen Elementen zurecht. Im Endeffekt müssten Sie zum Einbinden eines Applets also HTML-Code schreiben, der aus einer Kombination aller drei Elemente besteht. Und das ist von Hand viel zu aufwändig.

Die praktikable Lösung zum Einbinden von Applets für möglichst viele verschiedene Browser und Java-Versionen besteht darin, dass Sie zunächst ein einfaches `<APPLET>`-Element schreiben. Diesen HTML-Code lassen Sie sich dann von Suns **HTML Converter** in komplexeren Code umwandeln. Laden Sie sich dazu von der Seite <http://java.sun.com/products/plugin/1.3/converter.html> die Version 1.3.0_02 des HTML Converters herunter. Wenn Sie das Archiv auspacken, finden Sie darin ein JAR-Archiv, das Sie im Terminal mit folgendem Befehl starten können:

```
java -jar htmlconv1_3.jar -gui
```



Abbildung 4.43 HTML Converter

Ohne den Parameter `-gui` rufen Sie die Kommandozeilen-Version auf, die Sie gut bei komplett automatisierten Projekterzeugungen einsetzen können. Sobald Sie »Konvertieren« anwählen, sucht der HTML Converter im angegebenen Verzeichnis nach allen passenden Web-Dokumenten, in denen Applets eingebunden sein können, und tauscht darin den Code zum Einbinden gegen flexibleren Code aus. Wenn Sie als »Schablonendatei« den Eintrag »Erweitert (Standard + sämtliche Browser/Plattformen)« auswählen, wird beispielsweise auch JavaScript-Code in die HTML-Datei eingefügt. Diese Schablone ist bei Mac OS X vor allem für den Internet Explorer wichtig, der Applets nur mit einer speziellen Form des `<OBJECT>`-Elements vernünftig einbindet. Eine ausführliche Dokumentation des HTML Converters finden Sie auf der Seite http://java.sun.com/products/plugin/1.3/docs/htmlconv_01.html, die drei betroffenen HTML-Elemente sind auf <http://java.sun.com/products/plugin/1.3/docs/tags.html> beschrieben.

Um das Applet auszuführen, reicht es, das einbindende HTML-Dokument vom Browser lokal von der Festplatte ausführen zu lassen. Wenn Sie ein Applet aber außerhalb eines Browsers testen wollen, stehen Ihnen dafür zwei spezielle Programme zur Verfügung. Der Applet Launcher im Verzeichnis `/Programme/Dienstprogramme/Java/` wurde von Apple für Mac OS X entwickelt und erlaubt es Ihnen, Applets über eine grafische, wenn auch sehr einfach gehaltene Oberfläche zu starten. Alternativ können Sie im Terminal das Kommando `appletviewer` ausführen, dem Sie als Argument den Namen der HTML-Datei übergeben müssen. Dieses Programm gehört auch zum Standard-SDK von Sun und wird verwendet, wenn Sie Applets direkt aus Xcode über den Menüpunkt **Build • Build and Run** starten.

Falls sich bei Ihrem Xcode-Applet-Projekt nach dem Aufruf dieses Menüpunkts nichts tut, sind normalerweise zwei Fehlerquellen dafür verantwortlich (siehe Abbildung 4.44). Stellen Sie zum einen sicher, dass im Bereich »Executables« der Radiobutton beim Eintrag »appletviewer« gesetzt ist. Zum anderen muss in der Tabelle »Launch Arguments« der korrekte Name der HTML-Datei eingetragen sein.

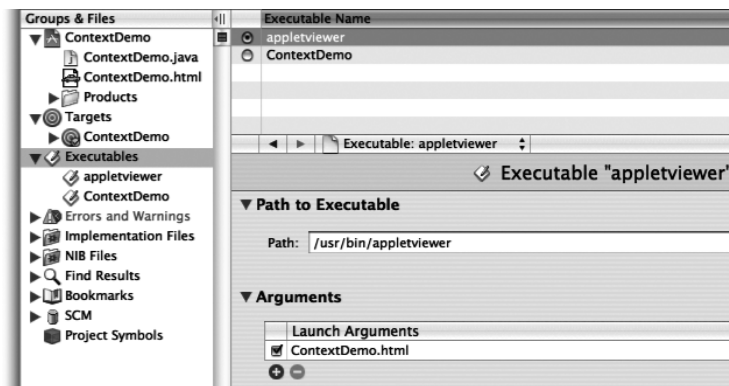


Abbildung 4.44 Xcode-Einstellungen, damit Applets gestartet werden können

Natürlich werden Sie Applets nur zum Testen im lokalen Dateisystem ausführen lassen. Für den realen Einsatz greifen Sie dann wieder auf das »Personal Web Sharing« mit dem in Mac OS X eingebauten Apache-Web-Server zurück, oder Sie übertragen den Applet-Code und die HTML-Datei per FTP auf einen Server im Internet. Beides funktioniert genau so, wie Sie es schon bei Java Web Start kennen gelernt haben. Wenn Sie Applets einmal »live« erleben wollen, finden Sie auf der Seite <http://java.sun.com/applets/> viele Beispiele zum Einstieg.

Java Plugin und Java Embedding Framework

Sie als Applet-Programmierer möchten im Wesentlichen, dass Ihr Applet von allen Browsern ausgeführt werden kann – nehmen Sie dazu am besten immer das `<APPLET>`-Element und lassen Sie dies gegebenenfalls vom HTML Converter umwandeln. Falls Sie aber zum kleinen Kreis der Programmierer gehören, die einen Web-Browser entwickeln, müssen Sie sich Gedanken darüber machen, wie das Einbinden der Applets in ein HTML-Dokument genau funktioniert. Mac OS X bietet Ihnen dafür zwei Techniken, das Java Embedding Framework und das Java Plugin.

Das Java Embedding Framework ist eine Systembibliothek, die Sie im Verzeichnis `/System/Library/Frameworks/JavaEmbedding.framework` finden. Mit den darin enthaltenen Funktionen können Sie die Java Virtual Machine, die das Applet letztendlich ausführt, selbst verwalten und steuern. Dies stellt die ursprüngliche und bis Mac OS X 10.1 einzige Möglichkeit dar, Applets in Nicht-Java-Anwendungen (wie z.B. einem Browser) einzubetten.

Seit Mac OS X 10.2 ist das Java Plugin verfügbar, das sich wie alle allgemeinen Browser-Plugins im Verzeichnis `/Library/Internet Plug-Ins/` befindet. Das Plugin ist von Sun auch für andere Betriebssysteme verfügbar und ist mittlerweile der von Apple empfohlene Weg, Applets in eigene Programme einzubinden. Von der Verwendung des Java Embedding Frameworks für neue Projekte rät Apple ab – zum einen, weil Apple diese Programmierschnittstelle nicht weiterentwickelt, zum anderen, weil das Java Plugin folgende Vorteile bietet:

- ▶ Die Textausgaben nach `System.out` und `System.err` können in einer Java-Konsole aufgezeichnet werden.
- ▶ Die Einstellungen für Applets (z.B. ob die Java-Konsole aktiv ist) muss der Anwender nur einmal für alle Browser vornehmen, und zwar in den Plugin-Einstellungen.
- ▶ JAR Caching: Applet-Archive werden automatisch in einem Zwischenspeicher abgelegt und verkürzen die Ladezeit bei wiederholtem Aufruf.
- ▶ Signierte JAR-Archive: Wenn der Benutzer einem Applet-Zertifikat zugestimmt hat, wird dies an zentraler Stelle gespeichert, so dass der Anwender nicht immer wieder dasselbe Zertifikat bestätigen muss – selbst wenn er das Applet in verschiedenen Browsern verwendet.

Aus HTML-Sicht ist das Element `<OBJECT>` zum Ansprechen beliebiger Plugins gedacht, d.h. auch für Java. Da einige Browser aber Probleme mit diesem Element haben und vor allem die aktuellen Windows-Browser auch bei `<APPLET>` das Java Plugin ansteuern, kann dies auch bei Mac OS X als Standard angenommen werden. Idealerweise sollten alle HTML-Elemente, die Applets ein-

binden können, auf das Java Plugin zurückgreifen – Apples Browser »Safari« macht es korrekt vor (s. u.).

Nebenbei kümmert sich das Plugin um eine nahtlose Systemintegration. So verwenden seit Java 1.4.2 alle über das Plugin eingebundenen Applets automatisch die HTTPS-Proxy-Konfiguration aus den Systemeinstellungen. Für Java 1.4.1 und 1.3.1 müssen Sie dies noch bei den Laufzeitparametern nach folgendem Schema explizit angeben:

```
-Dhttps.proxyHost=sichererproxy.meinefirma.de -Dhttps.proxyPort
  =443
```

Diese Properties können Sie dem `appletviewer` mit dem Parameter `-J` übergeben, allgemein tragen Sie diese Werte aber besser in den Plugin-Einstellungen ein. Die HTTP-Properties `http.proxyHost` und `http.proxyPort` werden übrigens schon immer aus den Systemeinstellungen gelesen.

Plugin-Einstellungen

Mit den Java Plugin-Einstellungen können Sie die grundlegende Konfiguration von Applets für alle Browser vornehmen, die das Java Plugin unterstützen. Wenn sich also jedes Mal die Java-Konsole öffnet, sobald ein Applet gestartet wird, können Sie dies hier ausschalten. Sie finden das Plugin-Bedienungsfeld im Verzeichnis `/Programme/Dienstprogramme/Java/`. Genau genommen liegen dort zwei Programme, `Java 1.3.1 Plugin Einstellungen` und `Java 1.4.2 Plugin Einstellungen` – für jede der Java-Hauptversionen eines. Je nachdem, mit welcher Java-Version ein Applet ausgeführt wird, müssen Sie die Einstellungen im einen oder im anderen Bedienungsfeld vornehmen. Leider können Sie im Applet- oder im HTML-Code nicht vorgeben, welche Java-Version verwendet werden soll (Java Web Start ist hier deutlich flexibler). Es hängt also vom Browser ab, welche Laufzeitumgebung benutzt wird. Derzeit unterstützt leider nur Apples eigener Browser »Safari« Java 1.4, alle anderen Browser kennen nur Java 1.3 – sofern sie überhaupt das Plugin verwenden.

Auf der Seite <http://javaplugin.sourceforge.net/> entsteht derzeit das Open Source-Projekt »Java Embedding Plugin«, mit dem auch andere Browser – derzeit Mozilla, Camino und Firefox – Java 1.4 für Applets verwenden können.

Im Plugin-Bedienungsfeld können Sie festlegen, ob die Java-Konsole verwendet wird oder nicht, ebenso können Sie sich auf Wunsch Laufzeitfehler (Exceptions) anzeigen lassen. Im Feld »Java-Laufzeitparameter« können Sie Argumente an die Java Virtual Machine übergeben – hier legen Sie z. B. Properties mit der `-D`-Option fest.

In dem Popup-Menü »Java Runtime Environment« steht bei Mac OS X derzeit nur der Eintrag »Java Plug-in-Standard verwenden« zur Verfügung, es ist auch keine manuelle Angabe eines anderen JRE möglich. Das liegt daran, dass Apple die Installation von nur jeweils einer 1.3er- und 1.4er-Version zulässt (beispielsweise ersetzt Java 1.4.2 eine bestehende 1.4.1-Installation), und für beide Hauptversionen gibt es ja ein separates Plugin-Bedienungsfeld.

Des Weiteren können Sie Speicherort und -größe des Applet-Zwischenspeichers (Cache) verändern und die Listen der gültigen Zertifikate verwalten (siehe Abbildung 4.45). Die Java 1.3.1 Plugin Einstellungen bieten generell weniger Optionen als die neuere Version, haben aber sonst einen ähnlichen Aufbau.

Während bei dieser Abbildung noch das lange gebräuchliche unsichtbare Verzeichnis `~/ .java/deployment/cache/` für den Zwischenspeicher eingetragen ist, verwendet das Plugin ab dem Java 1.4.2 Update 1 das besser zugängliche Verzeichnis `~/Library/Caches/Java Applets/`.

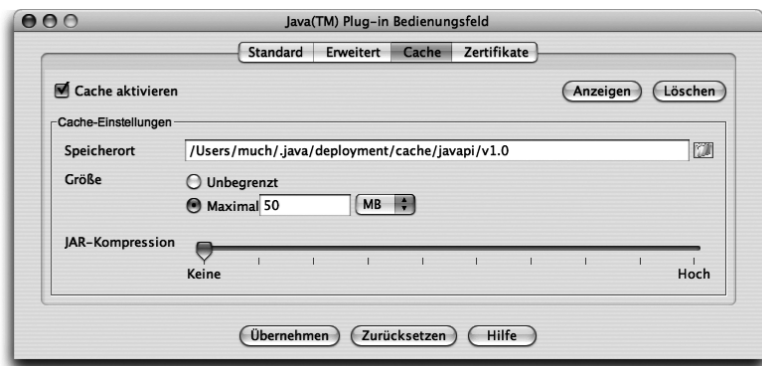


Abbildung 4.45 Java 1.4.2-Plugin-Einstellungen

JAR Caching

In den Plugin-Einstellungen für Java 1.4.1 können Sie festlegen, wo Applet-Archive zwischengespeichert werden und wieviel Platz dieser Cache maximal belegen darf. Beim 1.3.1-Plugin fehlen diese Konfigurationsmöglichkeiten, hier wird als Speicherort immer das Verzeichnis `~/Library/Caches/Java Applets/` verwendet.

Während dies die generellen Einstellungen sind, *falls* ein Applet zwischengespeichert wird, können Sie im HTML-Code für jedes Applet festlegen, *ob* es

überhaupt gespeichert werden soll. Das geschieht innerhalb des Applet-Elements (`<APPLET>`, `<EMBED>` oder `<OBJECT>`) mit speziellen `<PARAM>`-Elementen:

- ▶ `<param name="cache_option" value="plugin">`
Diese Zeile schaltet das JAR-Caching für das jeweilige Applet ein. Fehlt dieser Parameter, wird dieses Applet nicht vom Plugin zwischengespeichert. Andere erlaubte Werte im `VALUE`-Attribut sind »browser« (nicht das Plugin, sondern der Browser kümmert sich um den Cache; damit würde jeder Browser seine eigene Archiv-Kopie verwalten) und »no« (das Applet soll überhaupt nicht gespeichert werden).
- ▶ `<param name="cache_archive" value="a.jar, b.jar, c.jar">`
Hiermit legen Sie fest, welche Archive gespeichert werden sollen. Wenn Ihr Applet auf mehrere JAR-Archive zurückgreift, werden diese durch Komma getrennt angegeben. Sie dürfen hier keine absoluten URLs verwenden, die Archive werden relativ zum `CODEBASE`-Attribut des Applet-Elements gesucht! Achten Sie auch darauf, dass ein Archiv entweder nur im `ARCHIVE`-Attribut des Applet-Elements oder aber nur hier im »cache_archive«-Parameter angegeben ist.
- ▶ `<param name="cache_version" value="1.2.0.1, 2.1.1.2, 1.1.2.7">`
Dieser optionale Parameter erlaubt eine Versionierung der verwendeten JAR-Archive. Die Versionsnummern werden in derselben Reihenfolge wie die Archive im »cache_archive«-Parameter angegeben und können aus bis zu vier Ziffern (hexadezimal von 0 bis 9 und A bis F), durch Punkte getrennt, bestehen. Fehlt dieser Parameter, wird bei jedem Aufruf des Applets auf dem Server nachgesehen, ob eine neuere Version des JAR-Archivs existiert (ermittelt wird dies anhand der Länge des Archivs und des Datumsstempels, wann die Datei zum letzten Mal verändert wurde). Bei korrekter Angabe der Version gibt es dagegen keine unnötigen Server-Zugriffe – das JAR-Archiv wird nur dann geladen, wenn es noch nicht oder nur in einer älteren Version (mit einer niedrigeren Versionsnummer) zwischengespeichert wurde.
- ▶ `<param name="cache_archive_ex" value="d.jar;preload, e.jar;preload;1.1.0.0, f.jar">`
Bei Java 1.4 gibt es zwei wichtige Änderungen beim Applet-Caching durch das Java Plugin: Zum einen werden JAR-Archive aus dem `ARCHIVE`-Attribut nun automatisch zwischengespeichert (eine Versionierung ist bei dem dort angegebenen Archiv aber nach wie vor nicht möglich). Zum anderen gibt es den neuen »cache_archive_ex«-Parameter, der die beiden Parameter »cache_archive« und »cache_version« zusammenfasst und zusätzliche Konfigurationsmöglichkeiten bietet. Wenn alle drei Parameter angegeben sind, hat »cache_archive_ex« Vorrang.

Hinter jedem Archivnamen können Sie hier das optionale Flag »preload« und die optionale Versionsnummer, durch Semikolon getrennt, angeben. Ein Applet wird dann erst gestartet, wenn alle seine »preload«-Archive komplett geladen wurden.

Um kompatibel mit den meisten Mac-Browsern zu sein, die in der Regel nur Java 1.3 unterstützen, empfiehlt Apple derzeit noch, »cache_archive_ex« nicht zu verwenden und stattdessen die alte (auch unter Java 1.4 gültige) Kombination der beiden Parameter »cache_archive« und »cache_version« einzusetzen.

Sun hat die genaue Funktionsweise des Applet-JAR-Cachings auf den Seiten <http://java.sun.com/products/plugin/1.3/docs/appletcaching.html> und http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/applet_caching.html beschrieben.

Sicherheit

Grundsätzlich laufen Applets sicher, d.h. für den ausführenden Rechner ungefährlich, ab. Das »Sandbox«-Sicherheitsmodell sorgt unter anderem dafür, dass das Applet keinen Zugriff auf das lokale Dateisystem hat – es gelten im Wesentlichen dieselben Einschränkungen wie für Java Web Start-Applikationen. Folgende Besonderheiten sollten Sie bei Applets beachten:

- ▶ `System.exit()` darf nicht aufgerufen werden. Der Browser kümmert sich um das Beenden der Applets, spätestens dann, wenn er selbst beendet wird. Üblicherweise wird ein Applet schon beim Verlassen einer Seite inaktiv, aber es ist durchaus erlaubt, dass ein Applet-Thread länger weiterläuft.
- ▶ Von Applets geöffnete Fenster (z.B. mit `new Frame()`) sollten einen Warnhinweis im Titel oder in der Statuszeile anzeigen, damit der Anwender weiß, dass es sich nicht um ein normales Browser-Fenster handelt. Leider hält sich Apples Java-Implementation nicht daran und stellt die Fenster ganz normal dar.
- ▶ Obwohl ein Applet nicht auf Systemkomponenten zugreifen kann, ist eine Kommunikation mit dem Browser, dem Applet-Dokument und weiteren Applets im selben Dokument erlaubt (s.u.).

Dies ist das Standard-Sicherheitsmodell, das auf der Seite <http://java.sun.com/docs/books/tutorial/applet/practical/security.html> genauer beschrieben ist. Im Gegensatz zu Applets aus dem Internet haben lokal ausgeführte Applets üblicherweise keine Sicherheitseinschränkungen.

Während Java 1.0 nur dieses Sandbox-Modell vorsah und Java 1.1 als Ergänzung signierte Applets einführte, kann seit Java 1.2 mit so genannten Sicher-

heitsrichtlinien (Security Policies) sehr detailliert festgelegt werden, was ein Programm darf und was nicht – und diese Vorgaben sind nicht auf Applets beschränkt, Sie können sie auch bei normalen Anwendungen einsetzen. Mit folgender Richtliniendatei, die Sie beispielsweise als `VorsichtAllesErlaubt.txt` speichern können, erlauben Sie einem Applet den vollen Systemzugriff:

```
grant {
    permission java.security.AllPermission;
};
```

Listing 4.14 Einfache, aber gefährliche Sicherheitsrichtliniendatei

In einem oder mehreren `grant`-Blöcken sind so genannte Berechtigungen (Permissions) aufgeführt. Hier wird einfach nur die Berechtigung `java.security.AllPermission` gesetzt – alles ist erlaubt. Bei dieser Datei handelt es sich nicht um Java-Quelltext, sondern um ein spezielles Textformat für die Sicherheitsrichtlinien.

Bei real eingesetzten Sicherheitsrichtlinien sollten Sie wesentlich genauer festlegen, was erlaubt ist, beispielsweise der Zugriff nur auf bestimmte Rechner. Außerdem können Sie für verschiedene Applet-Quellen (unterschiedliche Server oder lokale Verzeichnisse) getrennt festlegen, welche Einschränkungen gelten. Sun hat den Aufbau der Policy-Dateien auf <http://java.sun.com/j2se/1.4.2/docs/guide/security/PolicyFiles.html> dokumentiert, eine Liste aller vorhandenen Berechtigungen finden Sie auf der Seite <http://java.sun.com/j2se/1.4.2/docs/guide/security/permissions.html>.

Die systemweit gültigen Standardrichtlinien finden Sie in der Datei `/Library/Java/Home/lib/security/java.policy`. Hier könnten Sie zwar Erweiterungen für Ihre Java-Anwendungen einfügen, aber beim nächsten Java-Update werden diese Änderungen wieder mit den Standardeinstellungen überschrieben. Als bessere Alternative können Sie für jeden Benutzer eine eigene Policy-Datei anlegen, die dann als Ergänzung zur systemweit gültigen Datei geladen wird. Java sucht diese Datei unter dem Namen `~/.java.policy` (beachten Sie den Punkt am Anfang des Dateinamens, die Datei ist also unsichtbar!). Am besten setzen Sie die Richtliniendatei aber beim Programmaufruf mit folgender Befehlszeile:

```
java -Djava.security.manager
    -Djava.security.policy=/Users/much/VorsichtAllesErlaubt.txt
    MeineTolleApplikation
```

Die nötigen Properties werden wie gewohnt mit der Option `-D` übergeben. Mit `java.security.policy` geben Sie den Namen Ihrer Richtliniendatei an. Außerdem müssen Sie mit `java.security.manager` den Standard-Sicherheitsmanager aktivieren, denn normale Applikationen laufen zunächst ohne Sicherheitsüberprüfungen.

Bei Applets bereitet dies natürlich ein Problem, denn sie werden nicht mit `java` aufgerufen. Wenn Sie den `appletviewer` verwenden, können Sie ihm mit der Option `-J15` (jeweils direkt vor einem `-D`) Parameter für die Applet-Laufzeitumgebung mitgeben:

```
appletviewer
  -J-Djava.security.policy=
    /Users/much/VorsichtAllesErlaubt.txt
    appletdemo.html
```

In diesem Fall müssen Sie den Standard-Sicherheitsmanager nicht mehr aktivieren, denn Applets besitzen automatisch einen passenden Applet-Sicherheitsmanager.

Für Applets, die im Browser angezeigt werden, legen Sie diese Optionen in den Plugin-Einstellungen fest. Dort tragen Sie die Option `-Djava.security.manager=VorsichtAllesErlaubt.txt` in das Feld »Java-Laufzeitparameter« ein. Bedenken Sie aber, dass diese Sicherheitsrichtlinien dann für alle Applets gelten, falls Sie den `grant`-Block nicht geeignet eingeschränkt haben. Die mit `-D` gesetzte Richtliniendatei wird als Ergänzung zur System- und Benutzerrichtlinie geladen.

Es ist recht mühsam, eine Richtliniendatei von Hand zu schreiben, denn Sie müssen alle verfügbaren Berechtigungen auswendig kennen, und wenn Sie sich vertippen, ist die gesamte Datei ungültig und wird ignoriert. Daher stellt Sun das »Richtlinientool« zur Verfügung, das Sie im Terminal mit dem Befehl `policytool` aufrufen können. In diversen Dialogen können Sie damit bestehende Richtlinien ansehen und verändern sowie neue Richtliniendateien erzeugen (siehe Abbildung 4.46). Die nötigen Berechtigungen können Sie dabei mit allen Parametern bequem aus Listen auswählen.

¹⁵ Achtung, dies ist keine Standardoption und kann sich theoretisch jederzeit ändern.

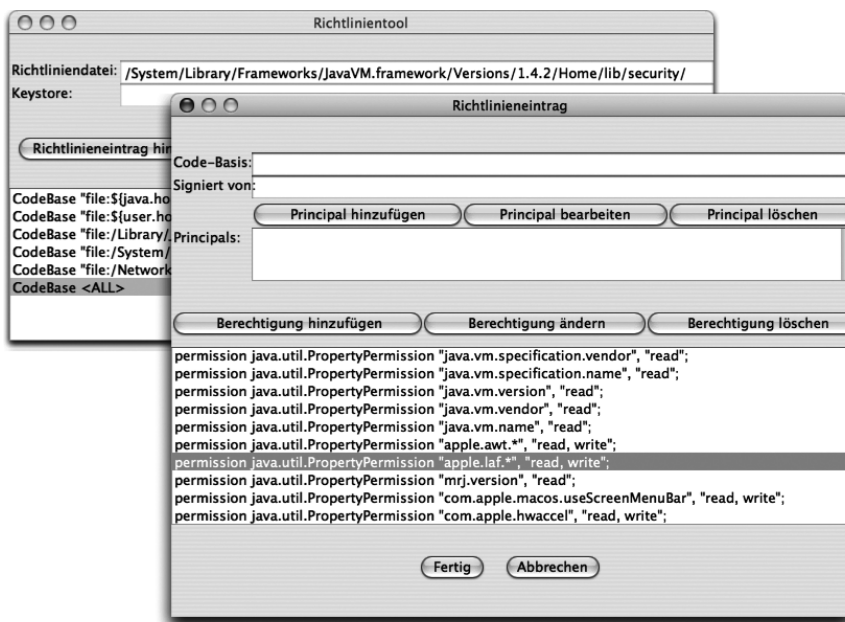


Abbildung 4.46 policytool zum Bearbeiten von Sicherheitsrichtlinien

4.6.2 Web-Browser

Da Applets meistens von einem Browser ausgeführt werden, finden Sie hier eine kurze Übersicht über die wichtigsten für MacOS X verfügbaren Browser – mit Hinweisen auf deren Besonderheiten und Probleme. Alle »großen« Windows-Browser gibt es auch für MacOS X, dazu kommt eine erstaunlich große Zahl von speziell für den Mac entwickelten Browsern. Eine Liste aller Mac-Browser und deren Versionen finden Sie auf der Seite <http://darrel.knutson.com/mac/www/browsers.html>, für alle Systeme listet <http://www.upsdell.com/BrowserNews/> aktuelle Updates auf. Wenn Sie sich für bestimmte ältere Versionen der Browser Netscape, Internet Explorer und Opera interessieren, hat die Seite <http://www.blooberry.com/indexdot/history/browsers.htm> die entsprechenden Informationen parat.

Alle der vorgestellten Browser kommen mit dem `<APPLET>`-Element zurecht, die meisten auch mit `<OBJECT>` und einige mit dem speziellen `<EMBED>`. Da HTML aber relativ unempfindlich gegen Fehler ist und manche Web-Designer daher sehr kreativ beim Schreiben der Applet-Elemente werden (was ein Browser dann gerade noch versteht, ein anderer aber schon nicht mehr), hilft Ihnen im Endeffekt nur eines, um die Funktionsfähigkeit Ihres Applets mit Ihrer HTML-Datei und einem bestimmten Browser sicherzustellen: Testen, testen, testen ...

Safari

Der Browser »Safari« ist eine Eigenentwicklung von Apple. Anfang 2003 war er – damals noch als Betaversion – verfügbar und hat quasi sofort den Internet Explorer als Standardbrowser auf Mac OS X abgelöst. Safari basiert auf KHTML und KJS, zwei Bibliotheken des Linux-KDE-Desktops. Apple hat diese Bibliotheken zum WebKit-Framework zusammengefasst und nennt sie WebCore und JavaScriptCore. Die Ergänzungen und Korrekturen, die Apple vorgenommen hat, wurden wieder dem KDE-Projekt zur Verfügung gestellt.



Abbildung 4.47 Apple Safari 1.2

Safari bindet Applets grundsätzlich über das Java Plugin ein und ist der derzeit einzige Mac-Browser, der Java 1.4 unterstützt. Leider kann aber tatsächlich ausschließlich Version 1.4 verwendet werden, im Browser selbst gibt es keine Möglichkeit, auf die ältere Java-Version umzuschalten. Dies kann aber nötig werden, denn die Applet-Einbindung mit dem 1.4-Plugin ist zurzeit teilweise fehlerhaft.¹⁶ Die einzige Lösung ist dann, das problematische Plugin manuell zu deaktivieren. Beenden Sie dazu Safari und entfernen Sie dann die Datei `JavaPluginCocoa.bundle` aus dem Verzeichnis `/Library/Internet Plugins/` (indem Sie sie z.B. auf den Desktop ziehen oder sonst wo sicher aufheben). Wenn Sie Safari nun wieder starten, werden Applets mit Java 1.3 ausgeführt – verwendet wird dann übrigens das Plugin `Java Applet.plugin`.

- Safari 1.2 – kostenlos (Bestandteil von Mac OS X)

<http://www.apple.com/de/safari/>

¹⁶ Eine gute Auflistung von Applet-Problemen mit Mac-Browsern finden Sie auf der Seite <http://segal.org/macjavabugs/>.

Internet Explorer

Der »Internet Explorer« von Microsoft war lange Zeit der Standardbrowser, sowohl unter dem klassischen Mac OS als auch unter Mac OS X. Microsoft hat die Weiterentwicklung aber mittlerweile eingestellt, was einer der Gründe für Apple war, einen eigenen Browser zu entwickeln.



Abbildung 4.48 Microsoft Internet Explorer 5.2.3

Die Macintosh-Version des Internet Explorers war eine vom Windows-Pendant unabhängige Entwicklung von Microsofts Mac-Abteilung. Obwohl die Versionsnummer an 5.0 erinnert, sind die Features der Mac-Version auf Windows-Seite eher mit den dortigen Versionen 5.5 und 6.0 vergleichbar.

Der IE verwendet beim <APPLET>-Element das Java Embedding Framework, bei <OBJECT> das Java Plugin – und ist damit im ersten Fall nicht mehr auf dem Stand der Technik. Außerdem funktioniert das weiter oben besprochene `appletdemo.html` nicht, da der Internet Explorer ein Applet nicht mehrfach auf einer Seite einbinden kann – kommentieren Sie einfach alle nicht benötigten Elemente aus, um ein bestimmtes Element zu testen.

- ▶ Internet Explorer 5.2 – kostenlos (derzeit noch mit Mac OS X ausgeliefert)
<http://www.microsoft.com/mac/>

Mozilla, Netscape, Camino, Firefox

Der Netscape-Browser war der erste große, bekannte Browser für das WWW. Er wurde zwischenzeitlich »Navigator« (und »Communicator«) genannt, ist

aber vor allem einfach unter dem Firmennamen »Netscape« bekannt und hat Versionsnummern bis 4.8 erreicht. Anfang 1998 wurde der Quelltext als Open Source offen gelegt, seitdem wird er unter dem Namen »Mozilla« neu- und weiterentwickelt. Während Sie also mit Mozilla immer die neuesten Entwicklungen bekommen, erscheinen ab und zu auch neue Netscape-Versionen, die dann auf etwas älteren, als stabil angesehenen Mozilla-Versionen basieren. Mit den Versionsnummern der beiden Browser-Varianten muss man etwas vorsichtig sein – Netscape 6.0 entspricht Mozilla 0.6, Netscape 7.0 entspricht Mozilla 1.0, und Netscape 7.1 basiert schließlich auf Mozilla 1.4.



Abbildung 4.49 Mozilla 1.7

Als »Erfinder« von Applets (zumindest was die HTML-Seite betrifft) kommen diese Browser mit allen Arten der Einbindung gut zurecht. Dabei werden Applets immer über ein Plugin eingebunden, und zwar mit dem speziellen Java Applet Plugin Enabler, was seinerseits auf das Apple-Plugin im selben Verzeichnis /Library/Internet Plug-Ins/ zurückgreift.

Nicht nur Netscape verwendet als Code-Basis den Mozilla-Kern, auch viele andere Browser-Projekte bauen darauf auf. Für MacOS X ist insbesondere »Camino« interessant, da er eine einfache und speziell für den Mac entwickelte Oberfläche besitzt. Offizielle Versionen werden leider nur sehr sporadisch veröffentlicht, laden Sie sich daher am besten einen »Nightly Build« herunter!

- ▶ Mozilla 1.7 – kostenlos
<http://www.mozilla.org/>

- ▶ Camino 0.8 – kostenlos
<http://www.mozilla.org/products/camino/>
<http://ftp.mozilla.org/pub/mozilla.org/camino/nightly/latest/>
- ▶ Firefox 0.9 – kostenlos
<http://www.mozilla.org/products/firefox/>
- ▶ Netscape 7.1 – kostenlos
<http://www.netscape.de/netscapeprodukte/netscape71/download.html>

OmniWeb

OmniGroups »OmniWeb« ist vermutlich der älteste Browser, der speziell für Mac OS X bzw. die darin enthaltenen Technologien entwickelt wurde – er war nämlich schon für das NeXT-System erhältlich, das dann von Apple aufgekauft und als Basis für Mac OS X verwendet wurde.



Abbildung 4.50 OmniGroup OmniWeb 5.0

Seit Version 4.5 benutzt OmniWeb intern Apples WebKit-Framework zur Darstellung der Webseiten, vorher war eine eigene Darstellungskomponente im Einsatz. Der Unterschied zu Safari besteht also im Wesentlichen in einer anderen Benutzungsoberfläche. OmniWeb kommt nur mit dem <APPLET>-Element vernünftig zurecht, bindet die Applets dann aber über das Java Plugin ein.

- OmniWeb 5.0 – ca. 30 US-\$
http://www.omnigroup.com/applications/omniweb/

Opera

»Opera« vom gleichnamigen Hersteller ist unter Windows eine beliebte Alternative zu Netscape und dem Internet Explorer. Die Mac-Version hinkte in der Entwicklung lange Zeit hinter der Windows-Variante her, mit Opera 7.50 wurde die Mac-Version dann aber endlich zeitgleich veröffentlicht.

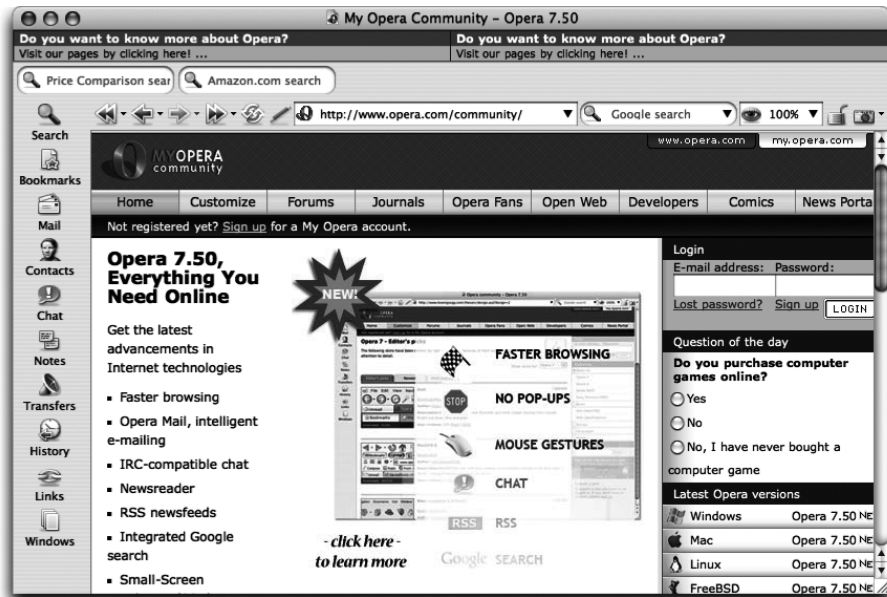


Abbildung 4.51 Opera 7.50

Opera verwendet bei <APPLET> das alte Java Embedding Framework, bei <OBJECT> das Java Plugin. Während Version 6.0 noch insgesamt eher schlecht als recht mit Applets zurechtkam, kommt Version 7.50 nun mit allen Arten der Applet-Einbindung klar.

- Opera 7.50 – kostenlos nutzbar, Registrierung 39 US-\$
http://www.opera.com/mac/

iCab

»iCab« vom gleichnamigen Hersteller ist ein kleiner, schneller Browser, der speziell für den Mac entwickelt wurde und für alle derzeit noch eingesetzten Mac-Systeme verfügbar ist – vom uralten 68k-Rechner bis zum aktuellen Mac OS X-

System. Er bietet seit langem einen eingebauten HTML-Validator, der Fehler in HTML-Seiten aufdeckt, sowie umfangreiche Filtermöglichkeiten, um die Privatsphäre des Anwenders zu schützen und lästige Werbung zu unterdrücken. Das große Problem ist eine mangelhafte CSS-Unterstützung, was aber mit der Version 3.0 gelöst werden soll.

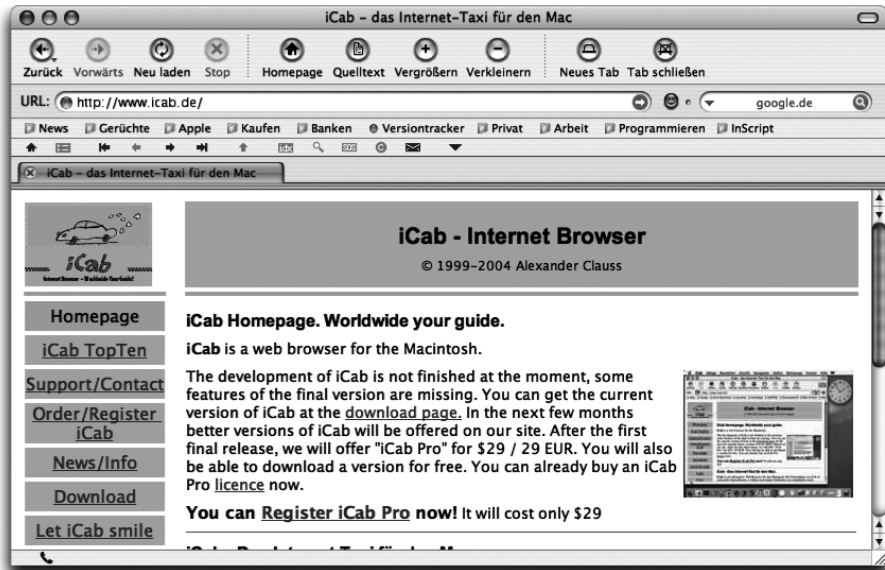


Abbildung 4.52 iCab 2.9.8

iCab verwendet zum Einbinden der Applets normalerweise das alte Java Embedding Framework, da der Browser auch auf alten MacOS X-Versionen funktionieren soll, die das Java Plugin noch nicht kennen. Ab Mac OS X 10.2 können Sie aber die Verwendung des Plugins erzwingen, indem Sie die iCab-eigene Java-Unterstützung über den Dialogeintrag **iCab · Einstellungen · Java · Java-Applets** ausführen ausschalten und dann sicherstellen, dass in **iCab · Einstellungen · Plug-ins** das »Java Plug-in« aktiviert ist.

- ▶ iCab 2.9 – kostenlos nutzbar, Registrierung 29 €
<http://www.icab.de/>

4.6.3 Kommunikation mit dem Browser

AppletContext

Seit Java 1.0 können Applets in begrenztem Umfang mit dem Browser kommunizieren, und zwar über ein `AppletContext`-Objekt. Eingesetzt wird dies hauptsächlich, um den Text in der Browser-Statuszeile mit `showStatus()`

zu verändern oder um im Browser-Fenster ein neues Dokument mit `showDocument()` laden zu lassen. Letzteres wird im folgenden Quelltext am Ende des Listings in der Methode `seiteAnzeigen()` demonstriert. Und nebenbei lernen Sie auch noch ein paar Besonderheiten von Swing-Applets kennen, die von der Klasse `javax.swing.JApplet` erben.

```
//CD/examples/ch04/applets/ContextDemo/ContextDemo.java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import javax.swing.*;

public class ContextDemo extends JApplet {
    private static final String adresse = "http://www.muchsoft.com/";
    private JButton knopf;

    public void init() {
        // hier können wir Swing-Komponenten Thread-sicher
        // zusammenbauen
        knopf = new JButton("<html><center>Seite anzeigen:"
            + "</center><br><b>" + adresse + "</b></html>");

        knopf.addActionListener( new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                seiteAnzeigen();
            }
        });

        this.getContentPane().add( knopf, BorderLayout.CENTER );
    }

    public void start() {
        // später (d.h. nach init) muss der Zugriff auf Komponenten
        // im Event-Dispatch-Thread stattfinden
        SwingUtilities.invokeLater( new Runnable() {
            public void run() {
                knopf.setToolTipText("Ruft den Standard-Browser auf "
                    + "und zeigt die angegebene Seite an");
            }
        });
    }
}
```

```

    }

    private void seiteAnzeigen() {
        try {
            URL url = new URL( adresse );
            AppletContext ac = this.getAppletContext();
            ac.showDocument( url, "_blank" );
        }
        catch (MalformedURLException e) { }
    }
}

```

Listing 4.15 Kommunikation vom Applet zum Browser über den AppletContext

In der Methode `seiteAnzeigen()` wird zunächst der Applet-Kontext mit der von `JApplet` geerbten (genauer gesagt von `Applet` indirekt geerbten) Methode `getAppletContext()` ermittelt. Über diese Referenz können nun alle Kontext-Methoden aufgerufen werden, hier `showDocument()`. Diese Methode bekommt mindestens ein URL-Objekt mit einer Internet-Adresse übergeben. Optional können Sie als zweiten Parameter einen Ziel-Frame bestimmen, in dem der Browser das Dokument anzeigen soll. »_blank« bedeutet dabei, dass die Seite immer in einem neuen Fenster angezeigt wird. Wenn Sie diesen Parameter weglassen, wird das aktuelle Dokument mit dem darin enthaltenen Applet ersetzt.

Die Applet-Kontext-Methoden werden von nahezu allen Browsern unterstützt, obwohl die Aufrufe ignoriert werden dürfen. Teilweise kann man dann auch im Browser einstellen, ob Applets die Statuszeile verändern können. Und beim `appletviewer` haben diese Methoden generell keine Funktion.

Wenn sich in einem Dokument (d.h. auf einer Webseite) mehrere Applets befinden, können diese über den `AppletContext` kommunizieren. Entweder verwendet ein Applet dazu die Methoden `getApplet()` bzw. `getApplets()`, welche andere Applets anhand ihres `HTML-NAME`-Attributes finden können. Über die zurückgegebene Referenz können Sie dann direkt Methoden im anderen Applet aufrufen. Oder Sie übertragen – falls Sie Java 1.4 voraussetzen können – Daten mittels Streams zwischen zwei Applets. Hierzu benötigen Sie die Methoden `setStream()` und `getStream()`.



Abbildung 4.53 Swing-Applet im appletviewer

Die Verwendung von `JApplet` als Oberklasse hat zwei offensichtliche Vorteile für die Oberflächenprogrammierung. Zum einen können Sie `Swing-JButtons` mit HTML-formatierten Texten verwenden. Zum anderen besitzt das Applet dann bereits ein `BorderLayout`, so dass hier auf das Setzen eines anderen Layout-Managers verzichtet werden kann – und bei `add()` am Ende der `init()`-Methode wird dann auch eine passende Konstante verwendet. Und wie Sie es von `Swing-Top-Level-Containern` gewohnt sind, dürfen Sie nicht `this.add()` aufrufen, sondern müssen erst den Inhalts-Container mit `getContentPane()` ermitteln.

Wichtig ist hier vor allem, wie die Oberfläche zusammengebaut wird, denn `Swing` ist nicht Thread-sicher. Alles, was die Oberfläche betrifft (Komponenten hinzufügen oder verändern), muss in einem bestimmten Thread, dem Ereignis-Verarbeitungs-Thread (Event-Dispatch-Thread), ausgeführt werden. Für die Applet-Methode `init()` ist dies automatisch sichergestellt. Bei allen später ausgeführten Methoden – im Beispiel `start()` – müssen Sie aber die Methode `invokeLater()` oder `invokeAndWait()` aus der Klasse `javax.swing.SwingUtilities` zu Hilfe nehmen, um den Code im Event-Dispatch-Thread auszuführen. Hier dient die Aufteilung der Aufrufe auf `init()` und `start()` nur zur Verdeutlichung der prinzipiellen Vorgehensweise, natürlich könnten Sie `setToolTipText()` auch sofort in `init()` einsetzen.

LiveConnect

Während der Applet-Kontext nur eine eingeschränkte Kommunikation vom Applet zum Browser und zu anderen Applets ermöglicht, hat Netscape für den Navigator 3.0 eine deutlich flexiblere Technologie namens »LiveConnect« entwickelt, mit der Java-Applets und in HTML-Seiten eingebetteter JavaScript-Code gegenseitig beliebige Methoden bzw. Funktionen aufrufen können.

Falls Ihnen JavaScript nichts sagt: Dies ist eine Skriptsprache, die unter anderem innerhalb von HTML-Dokumenten genutzt wird, um z.B. HTML-Formulare zu automatisieren oder um eine Button-Darstellung zu verändern, wenn Sie

mit dem Mauszeiger darüber fahren. Bis auf die ersten vier Buchstaben des Namens haben diese beiden Programmiersprachen nicht allzu viel gemeinsam! Leider sieht die Syntax oft recht ähnlich aus, aber die Funktionsweise ist dann meistens trotzdem eine andere. So müssen Sie in Java den Typ einer Variablen genau festlegen, bei JavaScript ist dies (noch) nicht möglich – zudem kann sich der Typ einer Variablen dort jederzeit ändern. Und während Java klassenbasierte Vererbung einsetzt, verwendet JavaScript Prototyp-basierte Vererbung. Eine gute Sammlung der häufigsten Fragen zu JavaScript findet Sie im JavaScript-FAQ auf der Seite <http://www.dcljs.de/faq/>. Die folgenden Beschreibungen setzen voraus, dass Sie sich grob mit HTML- und JavaScript-Programmierung auskennen.

Sehen Sie sich LiveConnect am besten an einem Beispiel an, zunächst aus Sicht von Java. Im Applet verwenden Sie dafür die Klasse `netscape.javascript.JSObject` (die immer genau so heißt, auch wenn sie bei anderen Browsern verwendet wird):

```
//CD/examples/ch04/applets/LiveConnectDemo/LiveConnectDemo.java
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import netscape.javascript.*;

public class LiveConnectDemo extends Applet {
    public String meineVariable = "Hallo von Java!";
    private TextArea ergebnis = new TextArea(7,40);

    public int meineMethode(String s) {
        ergebnis.append("### Zeichenkette aus JS: " + s + "\n");
        return 42;
    }

    public void init() {
        // Oberfläche zusammenbauen ...
    }

    private void javascriptOhneArgument() {
        try {
            JSObject win = JSObject.getWindow(this);
            Object obj = win.call("alert", null);
            ergebnis.append("*** JS-Funktion: " + obj + "\n");
        }
    }
}
```

```

    }
    catch (Exception e) {
        ergebnis.append(" LiveConnect-Fehler: " + e + "\n");
    }
}

// weitere Test-Methoden ...
}

```

Listing 4.16 Java-Applet, das JavaScript-Code aufruft

Das Zusammensetzen der Benutzungsoberfläche in der Methode `init()` wurde hier ausgespart, auf der Buch-CD finden Sie den Quelltext natürlich komplett. Die Oberfläche besteht aus ein paar Schaltflächen, mit denen der Anwender `javascriptOhneArgument()` und weitere Testmethoden aufrufen kann, sowie aus einer `TextArea` für die Ergebnisausgabe.

Innerhalb der Testmethoden findet eine `Exception`-Fehlerbehandlung statt, denn der Aufruf von `LiveConnect`-Methoden kann fehlschlagen, beispielsweise wenn der Browser `LiveConnect` gar nicht unterstützt. Die `Exception`-Klasse sollte in diesem Fall `netscape.javascript.JSException` sein.

Basis für die `LiveConnect`-Programmierung ist die Klasse `JSObject`, die eine wichtige statische Methode besitzt: `getWindow()`. Dieser Methode übergeben Sie die Referenz auf ein `Applet`-Objekt und erhalten ein `JSObject`-Objekt zurück, mit dem das zum `Applet` gehörende Browser-Fenster (bzw. der jeweilige `HTML-Frame`) repräsentiert wird – JavaScript nennt dieses Objekt das »globale« Objekt, das im JavaScript-Code mit `window` angesprochen wird. Als Parameter wird bei `getWindow()` meistens `this` übergeben, das aktuelle `Applet`.

Über diese `JSObject`-Referenz können Sie nun mit der Methode `call()` beliebige JavaScript-Funktionen aufrufen, sofern JavaScript diese in seinem globalen Objekt findet. Dieses Beispiel verwendet die Funktion `alert` ohne Parameter (daher die `null`-Referenz), wodurch eine leere `Alertbox` angezeigt werden sollte. Diese zwei Zeilen Java-Code entsprechen der JavaScript-Anweisung »`window.alert()`«.

Als Rückgabe erhalten Sie eine `Object`-Referenz. Je nach Rückgabewert, der in JavaScript lose typisiert ist, wählt `LiveConnect` einen passenden Java-Typ, beispielsweise `String` für Zeichenketten oder `Double` (die Hüllklasse, nicht der einfache Datentyp!) für Zahlen. Das Beispiel liefert hier einfach `null`, da die `alert`-Funktion keine Rückgabe hat.


```
Object[] args = { "Ein wichtiger Text..." };
Object obj = win.call("alert", args);
```

Wenn Sie `call()` mit Argumenten aufrufen wollen, erzeugen Sie einfach ein Objekt-Array, das irgendwelche Objekte in beliebiger Anzahl enthalten kann – LiveConnect kümmert sich beim Aufruf der JavaScript-Funktion darum, die Java-Typen geeignet umzuwandeln.

```
Object obj = win.getMember("testWert");
```

JavaScript-Objekte können nicht nur Funktionen, sondern auch Variablen enthalten, die Sie mit der `JSObject`-Methode `getMember()` abfragen können. Im Beispiel wird die JavaScript-Variablen `testWert` ausgelesen, die im folgenden HTML-Quelltext mit `-1` initialisiert wird. Auf Java-Seite erhalten Sie ein `Double`-Objekt mit dem Wert `-1.0`.

```
Object obj = win.eval("testWert = testWert + 2");
```

Schließlich können Sie mit der Methode `eval()` beliebige JavaScript-Ausdrücke (»Formeln«) auswerten lassen. Mit obiger Zeichenkette erhöht JavaScript den Wert seiner `testWert`-Variablen um 2, das Ergebnis wird an Java zurückgegeben – und ist dann auch mit `getMember()` abfragbar.

Im Applet-Quelltext tauchen noch die Variable `meineVariable` und die Methode `meineMethode()` auf. Beide werden auf Java-Seite nicht verwendet, sondern später vom HTML-Dokument bzw. dem darin enthaltenen JavaScript-Code angesprochen.

Damit Sie ein LiveConnect-Applet übersetzen können, muss der Compiler beim `import` wissen, wo er das Paket `netscape.javascript` findet. Apple liefert dies seit Java 1.4 als JAR-Archiv mit:

```
/Library/Java/Home/lib/netscape.jar
```

In Xcode machen Sie dies in den Target-Einstellungen bekannt. Ziehen Sie das Archiv einfach aus dem Finder in die »Search Path«-Liste in den Bereich »Java Classes« (siehe Abbildung 4.54). Der Pfad wird dabei zu seinem tatsächlichen Systempfad – hier speziell für Java 1.4.2 – aufgelöst. Damit Sie bei eventuellen Java-Updates auf der sicheren Seite sind, sollten Sie dies auf den oben angegebenen, allgemeineren Pfad ändern (nach einem Doppelklick auf die Pfadangabe können Sie einen anderen Wert eingeben). Beim Ausführen des Applets ist das Archiv `netscape.jar` automatisch bekannt, das Java-Plugin kümmert sich um die korrekte Einbindung.

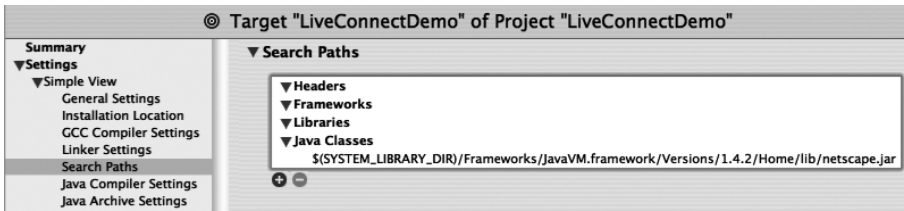


Abbildung 4.54 netscape.jar in Xcode auf den Klassenpfad setzen

Das HTML-Dokument, in dem das Applet eingebunden ist, wird nun etwas aufwändiger als bisher. Zum einen benötigen Sie das übliche Applet-Element, das auf den Java-Code verweist. Damit wäre bereits eine grundlegende Java-nach-JavaScript-Kommunikation möglich, denn jedes HTML-Dokument besitzt automatisch das zugehörige globale JavaScript-window-Objekt (egal ob Sie JavaScript-Code schreiben oder nicht). Zum anderen soll aber auch die umgekehrte Richtung möglich sein, die Kommunikation von JavaScript nach Java, und dafür müssen Sie explizit JavaScript-Code in das HTML-Dokument einfügen.

```

<!--
  CD/examples/ch04/applets/LiveConnectDemo/liveconnect.html -->
<html>
<head>
  <title>LiveConnectDemo</title>
</head>
<body>
<h2>Applet-Buttons</h2>

<applet
  name="meinapplet"
  archive="LiveConnectDemo.jar"
  code="LiveConnectDemo"
  width="400"
  height="250"
  mayscript>
Der Browser unterst&uuml;tzt Java nicht oder Java wurde in den
Browser-Einstellungen deaktiviert.
</applet>

<h2>HTML/JavaScript-Buttons</h2>
<script language="JavaScript1.1" type="text/javascript">
<!--
var testWert = -1;

```

```

function aufrufOhneArgument() {
    alert( document.applets.meinapplet.toString() );
}

// weitere JS-Funktionen ...
// -->
</script>

<!-- HTML-Buttons zum Aufruf der JS-Funktionen ... -->
</body>
</html>

```

Listing 4.17 JavaScript-Code, der Methoden des eingebundenen Applets ausführt

Beim Applet-Element fallen als Erstes zwei Attribute auf. Mit dem `NAME`-Attribut legen Sie einen Namen für das Applet fest, mit dem Sie es innerhalb dieses Dokuments eindeutig ansprechen können. `MAYSCRIPT` aktiviert die `JSObject`-Funktionalität für dieses Applet – ohne dieses Attribut könnten Sie keine Live-Connect-Aufrufe durchführen!

Im darauf folgenden `<SCRIPT>`-Element werden dann JavaScript-Variablen und -funktionen definiert. Die Variable `testWert`, die im Applet-Code abgefragt wird, wird hier auf `-1` initialisiert. Beachten Sie, dass die JavaScript-`var`-Anweisung zwar eine Variable deklariert, aber keinen Typ festlegt!

Die JavaScript-Funktion `aufrufOhneArgument()` soll nun eine Java-Methode ausführen. Dazu fragt JavaScript mit `document.applets.meinapplet` vom Browser zunächst ein Dokument-Objekt ab, das alle Applets der Seite kennt. In der `applets`-Liste können Sie Ihr Applet dann mit dem Namen ermitteln, den Sie im `<APPLET>`-Element festgelegt haben. Die Struktur des Dokument-Objekts und der darin enthaltenen Objekte ist genau festgelegt, und zwar vom so genannten DOM (Document Object Model). Nicht nur JavaScript kennt diese Dokumentstruktur, auch Java kann HTML- (und vor allem XML-) Dokumente nach diesem Standard verwalten. Die entsprechenden Java-Klassen und Interfaces finden Sie im Paket `org.w3c.dom`.

Haben Sie Ihr Applet ermittelt, können Sie nun alle öffentlichen Methoden des Applets ausführen – im Beispiel wird der Einfachheit halber `toString()` verwendet. Lassen Sie sich nun in einem Web-Browser die Seite `liveconnect.html` anzeigen und klicken Sie auf den HTML-Button »Java-Methode ohne Argumente aufrufen«. Wenn der Browser LiveConnect unterstützt, wird beim in JavaScript kodierten `toString()` bereits die entspre-

chende Java-Methode des Applets aufgerufen! In der JavaScript-Alertbox wird dann ungefähr der folgende Text angezeigt:

```
LiveConnectDemo[pane10,0,0,400x100,layout=java.awt.FlowLayout]
```

Dies ist die typische Zeichenkette, wenn Sie eine AWT-Komponente in einen String umwandeln. Ist der Browser nicht LiveConnect-fähig, wird stattdessen die JavaScript-toString()-Funktion ausgeführt, die in etwa folgende Ausgabe liefert:

```
[object HTMLAppletElement]
```

Um Java-Methoden mit Argumenten aufzurufen, übergeben Sie einfach die gewünschten Parameter (durch Komma getrennt, wie Sie es von Java gewohnt sind):

```
var a = document.applets.meinapplet;  
var ergebnis = a.meineMethode(12345);  
alert(ergebnis);
```

Bemerkenswert ist daran, dass `meineMethode()` im Applet-Code einen String-Parameter erwartet, hier aber eine Zahl übergeben wird. LiveConnect kümmert sich wieder um die passende Umwandlung! Öffentliche Variablen eines Applets können wie Methoden angesprochen (und auch verändert) werden, auch wenn dies kein guter Java-Stil ist:

```
alert( document.applets.meinapplet.meineVariable );
```

Während Sie bisher nur Applet-Eigenschaften verwendet (Methoden vorhandener Objekte aufgerufen und die Rückgabe ausgewertet) haben, sieht LiveConnect auch die Möglichkeit vor, Java-Objekte aus JavaScript heraus neu zu erzeugen – Java wird dadurch vollständig Teil der Skriptsprache:

```
var buffer = new java.lang.StringBuffer("ABC");  
var zahl = 123;  
buffer.append( zahl );  
var str = buffer.toString();  
alert(str);
```

Hierbei handelt es sich um JavaScript-Code! In den `var`-Variablen liegen untypisierte JavaScript-Werte, die bei Bedarf intern in Java-Objekte umgewandelt werden. Bei diesem Java-»Scripting« müssen Sie immer die vollqualifizierten Java-Klassennamen angeben, eine Import-Anweisung gibt es nicht. Falls Sie eine Klasse sehr häufig verwenden, können Sie sich aber eine Abkürzung definieren:

```
var StringBuffer = java.lang.StringBuffer; // Ersatz für import
var buffer = new StringBuffer("ABC");
```

LiveConnect und die Klasse `JSONObject` sind bei Sun auf der Seite <http://java.sun.com/products/plugin/1.3/docs/jsobject.html> beschrieben, außerdem finden Sie dort Verweise auf die ursprüngliche Netscape-Dokumentation.

Während LiveConnect bei Windows-Browsern weit verbreitet ist, tut sich auf dem Mac ein Problem auf – LiveConnect wird hier nämlich leider kaum unterstützt. Beim klassischen Mac OS boten dies im Wesentlichen zwei Browser an, Netscape 4 und iCab ab Version 2.6 (wenn in den iCab-Einstellungen die Option »JavaConnect« aktiviert ist). Unter Mac OS X wurde LiveConnect lange Zeit gar nicht unterstützt, bis für Mozilla 1.0 und darauf basierende Browser (Netscape bis Version 6.2.x) ein passendes Plugin veröffentlicht wurde. Sie können das `MRJPluginCarbon` in der Version 1.0.1 von der Seite <http://homepage.mac.com/pcbear/MRJPlugin/> herunterladen und im Verzeichnis `/Library/Internet Plug-Ins/` installieren. Neuere Mozilla- und Netscape-Versionen ignorieren dieses Plugin allerdings.

Seit Anfang 2004 gibt es nun aber endlich einen Lichtblick: Wenn Sie mindestens Java 1.4.2 installiert haben, unterstützt Apples Safari-Browser ab der Version 1.2 LiveConnect. Dabei kommt Safari gut mit den Testfällen im obigen LiveConnect-Demo zurecht, einzig das »Java-Scripting«, das Erzeugen von neuen Java-Objekten im JavaScript-Code, funktioniert noch nicht.

Alles in allem sollten Sie LiveConnect – wenn überhaupt – nur sehr zurückhaltend einsetzen, da Ihre Applets sonst zu sehr vom verwendeten Browser und dessen Version abhängig sind. Für Intranets, wo Sie die Rechnerkonfigurationen kontrollieren können, ist LiveConnect eine akzeptable Technologie, bei Internet-Applets sollten Sie aber besser darauf verzichten.

Als kleine Alternative bietet sich der bekannte Applet-Kontext an. Der Methode `showDocument()` können Sie einfach eine JavaScript-URL übergeben:

```
AppletContext ac = this.getAppletContext();
ac.showDocument( new URL( "javascript:alert('Hallo!')" ) );
```

Das funktioniert bei den meisten Browsern – leider aber gerade bei Safari nicht. Ein weiterer Nachteil dieser Vorgehensweise ist, dass Sie eventuelle Funktionsrückgabewerte nicht auswerten können. Und natürlich ist dies wiederum nur eine einseitige Kommunikation von Java nach JavaScript, nicht aber umgekehrt.

4.7 Literatur & Links

Einige nicht Java-spezifische Themen wurden in diesem Kapitel nur ansatzweise – soweit nötig – behandelt. Zu diesen Internet-Technologien finden Sie in den folgenden Quellen weiterführende Informationen:

► Apache Web-Server

- ▶ M. Hilscher, »Der eigene Webserver«, Galileo Computing 2003
Beschreibt nicht nur die Konfiguration des eigentlichen Apache-Web-Servers, sondern erklärt auch die Installation und Einrichtung weiterer typischer und wichtiger Komponenten.
- ▶ S. Kersken, »Apache 2«, Galileo Computing 2004
Auch wenn Mac OS X derzeit mit einer 1er-Version des Apache-Web-Servers ausgeliefert wird, können Sie eine neuere Version manuell installieren. Hier lernen Sie die Neuerungen kennen.
- ▶ <http://httpd.apache.org/>
Dies ist die Homepage des Open Source-Projekts »Apache HTTP Server«.

► HTML

- ▶ M. Lubkowitz, »Webseiten programmieren und gestalten«, 2. Auflage, Galileo Computing 2005
Dieses Buch stellt nicht nur HTML vor, sondern behandelt auch alle weiteren wichtigen Themen, die Sie bei einem etwas größeren Webauftritt bedenken sollten.
- ▶ D. Ragget et al., »HTML 4. Web-Publishing mit dem neuen HTML-Standard.«, 4. Aufl., Addison-Wesley 1997
Die Referenz für HTML 4.0 – nicht wie es eingesetzt wird, sondern wie es eingesetzt werden sollte. Alt, aber immer noch gut!
- ▶ <http://www.boku.ac.at/html Einf/hein.html>
Die HTML-Einführung bietet eine ausführliche Anleitung, wie man gutes HTML produziert.
- ▶ <http://selfaktuell.teamone.de/>
SELFHTML ist ein umfassendes Nachschlagewerk zu allen Themen rund um HTML.
- ▶ <http://www.w3.org/MarkUp/>
Die Seite des Standardisierungsgremiums W3C bietet eher trockene Spezifikationen, ist aber in Zweifelsfällen maßgebend.

► JavaScript

- ▶ C. Wenz, »Handbuch JavaScript«, 5. Aufl., Galileo Computing 2003
Mit diesem Lern- und Nachlagewerk erfahren Sie alles Wissenswerte über

die Skriptsprache, die einen viel zu ähnlichen Namen, aber ansonsten nicht wirklich viel mit Java gemeinsam hat.

► **XML**

- ▶ H. Vonhoegen, »Einstieg in XML«, 2. Aufl., Galileo Computing 2004
Lernen Sie XML praktisch kennen, vom ersten kleinen Beispiel über Schemata und Transformationen bis hin zu Ausgabeformatierungen.
- ▶ <http://www.boku.ac.at/htmlleinf/xmlkurz.html>
Gibt Ihnen eine kurze und schnelle Einführung in XML.
- ▶ <http://www.w3.org/XML/>
Die W3C-Seite ist wiederum nur für diejenigen interessant, die Spaß am Lesen von Spezifikationen haben.

5 Portable Programmierung

5.1	Strategien zur portablen Programmierung	288
5.2	Häufige Problembereiche	292
5.3	Oft benötigte Lösungen	298
5.4	Literatur & Links	305

- 1 **Grundlagen**
- 2 **Entwicklungsumgebungen**
- 3 **Grafische Benutzungsoberflächen (GUI)**
- 4 **Ausführbare Programme**
- 5 **Portable Programmierung**
- 6 **Mac OS X-spezielle Programmierung**
- 7 **Grafik und Multimedia**
- 8 **Werkzeuge**
- 9 **Datenbanken und JDBC**
- 10 **Servlets und JavaServer Pages (JSP)**
- 11 **J2EE und Enterprise JavaBeans (EJB)**
- 12 **J2ME und MIDP**
- A **Kurzeinführung in die Programmiersprache Java**
- B **Java auf Mac OS 8/9/Classic**
- C **Java 1.5 »Tiger«**
- D **System-Properties**
- E **VM-Optionen**
- F **Xcode- und Project Builder-Einstellungen**
- G **Mac OS X- und Java-Versionen**
- H **Glossar**
- I **Die Buch-CD**

5 Portable Programmierung

»Nur ein nicht portables Programm verlangt, dass alle Plattformen identisch sind.«¹

Bei obigem Zitat ging es um ein nützliches Eclipse-Plugin für J2EE-Anwendungen, das eigentlich komplett in Java programmiert ist und damit auf allen Java-Plattformen laufen müsste. Leider aber verlässt sich das Plugin auf die Anwesenheit eines bestimmten JAR-Archivs in einem bestimmten Verzeichnis, das zwar bei Suns Java-Implementation für Windows existiert, bei Apples Java-Implementation für Mac OS X dagegen nicht. Weil das Plugin also implementationsabhängige, nicht standardisierte Eigenschaften nutzt, verschenkt es unnötigerweise die Portabilität.

Was macht ein portables Java-Programm aus? Portabilität bedeutet, dass das Programm auf unterschiedlichen Plattformen genutzt werden kann, d.h. auf Plattformen, die nicht identisch aufgebaut sind. Es gibt also Eigenschaften einer Plattform, die auf der anderen nicht – oder auf eine andere Weise – genutzt werden können. Ein portables Programm beachtet solche Unterschiede. Die einzige Voraussetzung, die ein portables Java-Programm verlangen darf, ist eine installierte Java-Laufzeitumgebung mit der passenden Klassenbibliothek (beispielsweise J2SE).

In diesem Kapitel geht es nun darum, wie Sie bestimmte Funktionalitäten nutzen, ohne dass Ihr Quelltext dadurch auf eine Zielplattform festgelegt ist. Spezielle Anpassungen an ein Betriebssystem oder an eine Java-Implementation sind also erlaubt, sie dürfen aber nicht dazu führen, dass die Applikation anderswo nicht mehr läuft. Zunächst werden dazu ein paar allgemeine Überlegungen angestellt, wie man Java-Code portabel programmiert. Danach werden die häufigsten Probleme geschildert, die bei der portablen Java-Programmierung auftreten – und die erst dann auffallen, wenn Ihre Software plötzlich auf einem anderen System als beispielsweise Windows eingesetzt werden soll. Zum Schluss werden dann ein paar Speziallösungen vorgestellt, die häufiger benötigt werden, z.B. das Öffnen einer URL in einem Web-Browser. Bei diesen Lösungen kommen teilweise Bibliotheken zum Einsatz, die systemabhängig und damit eigentlich nicht portabel sind. Die Entwickler der Bibliotheken haben Ihnen aber die Portierung der Routinen auf diverse Plattformen bereits abgenommen, so dass Sie diese Bibliotheken als portabel zwischen den unterstützten Zielplattformen betrachten können. Falls Sie solche Portierungen doch

¹ <http://lists.apple.com/archives/java-dev/2004/Mar/17/usinglombozpluginwithecl.005.txt>

einmal selbst vornehmen müssen, finden Sie im folgenden Kapitel über MacOS X-spezifische Programmierung alle nötigen Informationen dazu. Nicht behandelt wird in diesem Kapitel die portable Programmierung von speziell angepassten Benutzungsoberflächen, da dies schon Thema in Kapitel 3, *Grafische Benutzungsoberflächen*, war.

5.1 Strategien zur portablen Programmierung

Archivnamen sind nicht standardisiert

Verlassen Sie sich nicht darauf, dass die Standardklassen in bestimmten JAR-Archiven gespeichert sind. Beispielsweise gibt es die Archive `tools.jar` und `rt.jar` bei der MacOS X-Java-Implementierung nicht, die entsprechenden Klassen sind hier in den Archiven `classes.jar` und `ui.jar` zusammengefasst. Diese liegen zudem nicht im Verzeichnis `$JAVA_HOME/lib/` (dort befinden sich zwar einige der gewohnten Archive, aber eben nicht alle), sondern in `/System/Library/Frameworks/JavaVM.framework/Classes/`.

Die Lösung ist denkbar einfach: Wenn Sie eine bestimmte Klasse benötigen, die nicht zur Standard-Klassenbibliothek der Java-Laufzeitumgebung gehört, laden Sie sie einfach! Sie dürfen dabei allerdings die Klasse nicht direkt im Quelltext als Typ verwenden, sondern Sie müssen sie mit `Class.forName()` laden und dabei auf Exceptions reagieren. Wenn beim Laden ein Fehler auftritt, rufen Sie entweder alternativen Code auf (der eventuell nicht ganz so performant, aber dafür plattformunabhängig ist) oder Sie zeigen – falls es keine Alternativen gibt – dem Benutzer eine aussagekräftige Fehlermeldung an. Der folgende Abschnitt über »Reflection« beschreibt genauer, wie dies funktioniert.

Falls Sie tatsächlich auf die JAR-Archive direkt zugreifen müssen, um irgendwelche Daten daraus auszulesen, gibt es zur Not noch eine andere Möglichkeit. Extrahieren Sie die einzelnen Archivnamen aus dem Klassenpfad und suchen Sie dann Archiv für Archiv nach den benötigten Daten. Dazu lesen Sie mit `System.getProperty()` die System-Properties `java.class.path` und vor allem `sun.boot.class.path` aus, die Sie dann mit einem `java.util.StringTokenizer` in die einzelnen Archivpfade zerlegen können. Beachten Sie dafür aber auf jeden Fall den Abschnitt 5.2.1 über Pfadtrennzeichen.

Reflection statt Import

Falls Sie Klassen verwenden müssen, die nur bei einer bestimmten Java-Implementation zur Verfügung stehen, binden Sie diese *nicht* mit `import` (oder mit einer vollqualifizierten Klassenangabe) in den Quelltext ein!

```
import com.apple.eawt.Application;
// ...
    java.awt.Point p = Application.getMouseLocationOnScreen();
```

Listing 5.1 Problematischer Import von systemspezifischen Klassen

Zum einen kann ein solcher Quelltext nicht ohne weiteres auf anderen Systemen kompiliert werden – es sei denn, es gibt für den Import spezielle »Stub«-Klassen, die einfach nur die Schnittstellen ohne sinnvolle Implementierung enthalten. Zum anderen wird aber allein das Laden der kompilierten Klasse auf anderen Systemen zum Programmabbruch führen. Der Klassenlader (`java.lang.ClassLoader`) versucht nämlich, alle in dieser Klasse verwendeten Klassen zu laden, um sie statisch zu initialisieren.

Die Lösung besteht darin, solche systemabhängigen Klassen dynamisch nachzuladen, und zwar mithilfe von Javas Reflection-API:

```
java.awt.Point p = null;
try {
    Class c = Class.forName("com.apple.eawt.Application");
    java.lang.reflect.Method m =
        c.getMethod( "getMouseLocationOnScreen", null );
    p = (java.awt.Point) m.invoke( null, null );
}
catch (Exception e) {
    e.printStackTrace();
}
if (p == null) {
    // alternative Routine aufrufen oder Fehlermeldung ausgeben
}
System.out.println( p );
```

Listing 5.2 Dynamisches Laden von Klassen mit Reflection

Mit `Class.forName()` wird dabei zunächst die gewünschte Klasse geladen. Dieser Ansatz führt zu keinen Problemen mit dem Klassenlader, denn der Klassenname ist hier als Zeichenkette angegeben und wird daher nicht als Typ interpretiert. Falls die Klasse nicht gefunden wird, erhalten Sie eine `ClassNotFoundException` und können auf alternativen Code zurückgreifen – oder dem Anwender eine Fehlermeldung anzeigen. Konnte die Klasse erfolgreich geladen werden, erhalten Sie als Rückgabe ein Klassen-Objekt, über das Sie nun `getMethod()` aufrufen, um die gewünschte Methode zu ermitteln. Das Method-Objekt schließlich besitzt eine Methode `invoke()`, um den eigent-

lichen Methodenaufruf durchzuführen. In diesem Beispiel nicht gezeigt ist das Erzeugen eines Objektes mit `newInstance()` und das Auffinden spezieller Konstruktoren mit `getConstructor()`. Dieses indirekte Zusammenbauen von Objekten und Methodenaufrufen bezeichnet man als Meta-Programmierung oder (bei Java) als Reflection.

Mit dieser Technik lässt sich der Quelltext auf jedem System übersetzen und die Klasse problemlos laden. Wenn Sie allerdings sehr viele Methoden auf diese Weise aufrufen, wird der Quelltext schnell unübersichtlich. Ein eleganter Ausweg ist die Definition eines Interfaces, das von den systemabhängigen Klassen implementiert wird – dies setzt natürlich voraus, dass Sie Zugriff auf alle Implementationen haben, was bei fremden Bibliotheken nicht immer der Fall sein dürfte.

```
public interface IBank { /* ... */ }
```

Mit einem solchen Interface können Sie nun diverse Implementierungsklassen schreiben, hier `MacImpl` und `WinImpl` genannt. Folgender Ansatz kann dann gefährlich sein:

```
IBank bank = null;
if (Sys.isMacOSX()) {
    bank = new com.muchsoft.bank.MacImpl();
}
else if (Sys.isWindows()) {
    bank = new com.muchsoft.bank.WinImpl();
}
```

Es werden je nach erkanntem System Objekte der unterschiedlichen Implementierungen erzeugt und der Interface-Referenz zugewiesen. Sofern Ihre Programm-Distribution für ein System immer die Implementierungsklassen aller Zielsysteme beinhaltet, läuft das Programm problemlos. Wenn Sie aber jeder Distribution nur die Implementierungsklasse für das jeweilige System beilegen, bekommen Sie wieder die oben beschriebenen Probleme mit dem Klassenlader. Auch hier ist Reflection die passende Lösung:

```
IBank bank = null;
try {
    if (Sys.isMacOSX()) {
        bank = (IBank)
            Class.forName("com.muchsoft.bank.MacImpl").newInstance();
    }
    else if (Sys.isWindows()) {
```

```

        bank = (IBank)
            Class.forName("com.muchsoft.bank.WinImpl").newInstance();
    }
    else {
        // System wird nicht unterstützt, Abbruch
    }
}
catch (Exception e) {
    e.printStackTrace();
}
if (bank == null) {
    // Fehlermeldung und Programmabbruch
}
else {
    // Methoden der IBank-Schnittstelle aufrufen
}
}

```

Je nach erkanntem System werden auch hier die passenden Implementierungen geladen, diesmal allerdings mit `Class.forName()`. Mit `newInstance()` wird dann von der dynamisch geladenen Klasse ein Objekt erzeugt, das im weiteren Quelltext nur über die systemunabhängige `IBank`-Schnittstelle angesprochen wird.

Nicht auf implementationsspezifisches Verhalten verlassen

Verlassen Sie sich nicht auf nicht standardisiertes Verhalten bestimmter Implementationen der Java-Laufzeitumgebung – damit sind auch Fehler gemeint, wenn sich eine Implementation beispielsweise nicht an die Spezifikation der Java Virtual Machine hält.

Dieser Punkt ist sehr schwierig umzusetzen, denn Sie müssen erst einmal erkennen, dass Sie nicht portabel programmieren – schließlich läuft mit der bei Ihnen installierten Java-Laufzeitumgebung ja alles problemlos!? Ein Beispiel hierfür ist die Swing-Programmierung. Swing ist in weiten Teilen nicht Thread-sicher und verlangt daher, dass alle Änderungen an der Benutzungsoberfläche im `AWT-Ereignis-Verarbeitungs-Thread` durchgeführt werden. Dies geschieht üblicherweise mithilfe der Methoden `javax.swing.SwingUtilities.invokeLater()` und `invokeAndWait()` (bzw. mit den entsprechenden Methoden aus `java.awt.EventQueue`). Wenn Sie sich nicht daran halten – und damit inkorrekten Code programmieren –, gibt es Systeme, bei denen die Programmoberfläche trotzdem noch einigermaßen performant zu bedienen ist. Unter `Mac OS X` kann es dagegen zu einem drastischen Performanzeinbruch bei der

Benutzungsoberfläche kommen. Apple hält sich bei der Java-Implementation strikt an die Java-Spezifikationen, die allerdings nicht festschreiben, wie Threads intern zu realisieren sind – und das Apple-Modell ist leider etwas anfälliger gegen schlechte Programmierung.

Wie gehen Sie nun am besten vor, wenn sich Anwender beschweren, dass Ihre Java-Applikation auf einem bestimmten System nicht vernünftig funktioniert, zumal Sie dieses System vielleicht gar nicht zum Testen verfügbar haben? Neben den typischen Lösungen (Debug-Ausgaben, Nachfragen in Internet-Foren) bleibt Ihnen vor allem eines: Suchen Sie in Suns Java-Fehlerdatenbank »Bug Parade« (<http://developer.java.sun.com/developer/bugParade/>), ob es zum betreffenden Thema bekannte Fehler in der von Ihnen verwendeten Java-Laufzeitumgebung gibt.

Andersherum tritt dieses Problem auf, wenn Sie bekannte Fehler einer Java-Implementation mit Workarounds umgehen. Stellen Sie in diesem Fall sicher, dass Ihre Fehlerkorrekturen nur bei dem bestimmten Betriebssystem und der exakten Version der Java-Implementierung verwendet werden – schließlich sollen andere Versionen nicht »kaputt korrigiert« werden. Dies setzt dann natürlich voraus, dass Sie bei neuen Java-Implementationen testen, ob die von Ihnen entdeckten Fehler beseitigt wurden.

Kompatibilität zwischen Java-Versionen

Portabilität betrifft nicht nur verschiedene Betriebssysteme, auch innerhalb eines Systems müssen Sie darauf achten, dass Ihre Anwendung mit verschiedenen Java-Implementierungen zurechtkommt. Bei Mac OS X betrifft dies die verschiedenen installierten Java-Versionen, auf anderen Systemen können auch noch Implementationen derselben Java-Version von unterschiedlichen Herstellern hinzukommen.

Java-Programme sind normalerweise aufwärtskompatibel zu neueren Java-Versionen. Problematisch kann es werden, wenn Sie mit einer neueren Version Software für ältere Java-Umgebungen schreiben möchten. Was Sie in diesem Fall beachten müssen, beschreibt Sun auf den Seiten <http://java.sun.com/j2se/1.3/compatibility.html> und <http://java.sun.com/j2se/1.4/compatibility.html>.

5.2 Häufige Problembereiche

5.2.1 Datei- und Pfadtrennzeichen

Auf verschiedenen Systemen sehen Pfadangaben ganz unterschiedlich aus:

- ▶ `/Benutzer/much/` (Mac OS X, UNIX)
- ▶ `Macintosh HD:Dokumente:` (Mac OS Classic)
- ▶ `C:\Dokumente und Einstellungen\Thomas Much\` (DOS, Windows)

Abgesehen davon, dass die Systeme ganz unterschiedliche Ordner für ähnliche Zwecke anbieten, werden in den Pfadangaben andere Trennzeichen verwendet, um Verzeichnisse und Unterverzeichnisse zu separieren. Mac OS X und die diversen UNIX-Systeme verwenden den Slash (/), Windows den Backslash (\) und das klassische Mac OS den Doppelpunkt. Diese große Vielfalt ist für portable Programmierung natürlich zunächst ein Horror. Versuchen Sie daher am besten, keine absoluten Pfadangaben in Ihrem Programm zu verwenden, um die Problematik der unterschiedlichen Trennzeichen ganz außen vor zu lassen.

Wenn Sie auf systemabhängige Pfadangaben nicht ganz verzichten können, weil Sie gewisse Konfigurationsdateien einlesen müssen, lagern Sie diese Pfade in eine Textdatei aus (beispielsweise als Properties-Datei mit `java.util.Properties` oder im XML-Format). Aus dieser Datei lesen Sie dann einen absoluten Basispfad ein, der vom Anwender (bzw. vom Administrator) korrekt in die Datei eingetragen werden muss. Alle weiteren Pfade werden dann einfach relativ zu diesem Basispfad verwendet:

```
File textVerzeichnis = new File( basisPfad, "texte" );
```

Hier kümmert sich die Standardklasse `java.io.File` darum, zwischen dem Basispfad und dem Unterverzeichnis `texte` das passende Trennzeichen einzufügen.

Von wo aber wird die Datei mit dem Basispfad geladen? Entweder legen Sie sie im Arbeitsverzeichnis ab, in dem das Programm später ausgeführt wird – dann können Sie im Quelltext einfach ein `File`-Objekt ganz ohne Pfadangabe verwenden. Oder Sie lassen vom Start-Skript, das meistens sowieso an das jeweilige System angepasst werden muss, eine System-Property setzen, die Sie im Programm dann einfach mit `System.getProperty()` auslesen können:

```
java -Dbank.config.path=/Users/much/OnlineBank.cfg OnlineBank
```

Natürlich können Sie sich auch einfach auf fertige Bibliotheken verlassen, die für jedes System einen passenden Basispfad ermitteln. Wenn Sie beispielsweise Voreinstellungen speichern wollen, können Sie von der Klasse `System` (auf der Buch-CD) mit der Methode `getPrefsDirectory()` den passenden Pfad erfragen, dort dann ein Verzeichnis mit dem Programmnamen anlegen und darin alle Dateien speichern.

Falls Sie wirklich Pfadangaben von Hand zusammenbauen wollen, können Sie die Konstanten `File.separator` (ein `String`) oder `File.separatorChar` (ein `char`) auslesen und diese Zeichen mit den Verzeichnisnamen verknüpfen. Allerdings enthalten diese Konstanten jeweils nur ein einzelnes Zeichen – wenn Sie also ganz korrekt programmieren und den Fall vorsehen wollen, dass der Dateitrenner auf einem System mehr als ein Zeichen lang ist, werten Sie stattdessen `System.getProperty("file.separator")` aus.

Ein ganz anderes Thema betrifft die maximal erlaubte Länge von Dateinamen, die beispielsweise bei MacOS Classic nur 31 Zeichen betrug. Hier hilft letztendlich nur das Testen Ihrer Applikation auf allen gewünschten Zielsystemen. Dies gilt genauso für Leerzeichen, Umlaute und andere Sonderzeichen im Dateinamen – auch wenn MacOS X problemlos damit zurechtkommt, sind solche Namen nicht portabel.

Ein weiteres Problem tritt auf, wenn Sie Pfadlisten auswerten wollen. Der Wert der System-Property `java.library.path` (die Liste der Suchpfade für JNI-Bibliotheken) sieht in etwa wie folgt aus:

- ▶ `./Users/much/Library/Java/Extensions:/Library/Java/Extensions:/System/Library/Java/Extensions:/usr/lib/java (Mac OS X)`
- ▶ `.;C:\WINDOWS\system32;C:\WINDOWS;C:\j2sdk1.4.2_03\bin (Windows)`

Mac OS X und UNIX bzw. Linux verwenden den Doppelpunkt zum Trennen der einzelnen Pfadangaben, Windows setzt hierfür das Semikolon ein. Das Pfadtrennzeichen des jeweiligen Systems können Sie aus den Konstanten `File.pathSeparator` bzw. `File.pathSeparatorChar` oder – besser – mit `System.getProperty("path.separator")` auslesen.

5.2.2 Zeilenenden

Wenn Sie Zeilenenden (Zeilenumbrüche) innerhalb eines Textes erkennen müssen, können Sie die dafür verwendeten Zeichen mit `System.getProperty("line.separator")` abfragen. Auf den meisten Systemen werden dafür die ASCII-Zeichen CR (»Carriage Return«, Dezimalwert 13, Escape-Sequenz `\r`) oder LF (»LineFeed«, Dezimalwert 10, Escape-Sequenz `\n`) oder eine Kombination von beiden verwendet:

- ▶ LF bzw. `\n` (Mac OS X, UNIX)
- ▶ CR bzw. `\r` (Mac OS Classic)
- ▶ CR LF bzw. `\r\n` (DOS, Windows)

Wenn Sie Texte mit der Methode `println()` aus den Klassen `java.io.PrintStream` oder `java.io.PrintWriter` ausgeben, wird automatisch der richtige Zeilenumbruch des jeweiligen Systems geschrieben.

Genau dieses für lokale Dateien passende Verhalten von `println()` kann bei der Netzwerkkommunikation gefährlich werden, wenn Sie selber die Netzwerkprotokolle realisieren und dafür Texte über Sockets schicken. Sehen Sie sich beispielsweise das erste Kommando beim POP3-Protokoll zum Abrufen von E-Mails an:

```
out.println("HELO ich@irgendwo.de");
```

Wenn diese Zeile unter Mac OS X ausgeführt wird, kommt beim Server dann Folgendes an:

```
HELO ich@irgendwo.de\n
```

Die meisten Internet-Protokolle verlangen aber, dass solche Kommandos mit `\r\n` (CR LF) abgeschlossen sind. Daher lautet der korrekte Java-Code

```
out.print("HELO ich@irgendwo.de\r\n");
```

Apple hat zu diesem Thema eine Technote auf <http://developer.apple.com/technotes/tn/tn1157.html> veröffentlicht. Der Text wurde ursprünglich für Mac OS Classic geschrieben, ist heute aber immer noch gültig. Allerdings tritt der dort beschriebene Fehler kaum noch auf, da die meisten aktuellen Server keine Probleme mehr mit falschen Protokoll-Zeilendenen haben.

5.2.3 Zeichenkodierung

Computer verarbeiten nur Zahlen – damit daraus Buchstaben und andere Zeichen werden, wird jedem Zeichen eine bestimmte Zahl (Nummer) zugeordnet. Diese Zuordnung ist die Zeichenkodierung, und nahezu jedes Betriebssystem besitzt mindestens eine eigene, spezielle Kodierung. Oft fassen solchen Kodierungen maximal 256 Zeichen, weshalb mehrere Kodierungen, die so genannten »Codepages«, zum Einsatz kommen, die bei Bedarf umgeschaltet werden müssen. Da der Dokumentenaustausch zwischen so vielen Kodierungen sehr aufwändig und fehleranfällig ist, wurde vor einigen Jahren der **Unicode**-Standard (<http://www.unicode.org/>) entwickelt, der in einer einzigen, einheitlichen

Tabelle die Zeichenkodierung für alle Betriebssysteme, Programme und Sprachen festlegt.

Java arbeitet intern mit Unicode, viele Betriebssysteme kommen damit aber leider noch nicht vernünftig zurecht. Daher werden Textdateien, die Sie in Java bearbeiten, üblicherweise in der System-Kodierung auf der Festplatte abgelegt. Beim Einlesen und Speichern der Texte findet also eine Umwandlung nach und von Unicode statt. Selbst bei MacOS X, was Unicode problemlos verarbeitet (TextEdit beispielsweise kann Unicode-Texte in den Kodierungen UTF-8 und UTF-16 laden und speichern), findet eine Umwandlung statt.

Den Namen der System-Zeichenkodierung können Sie mit `System.getProperty("file.encoding")` abfragen. Windows liefert Ihnen dabei »Cp1252«, MacOS und MacOS X »MacRoman« (jeweils für Westeuropa und Amerika). Eine kurze Übersicht über den Mac-Zeichensatz finden Sie auf der Seite <http://de.wikipedia.org/wiki/MacRoman>, eine ausführliche Liste mit einem Vergleich zur Unicode-Kodierung und den HTML-Entities auf <http://www.alan-wood.net/demos/macroman.html>.

Wenn Sie zwischen beliebigen Kodierungen konvertieren wollen, können Sie dies mit den Klassen `java.io.InputStreamReader` und `java.io.OutputStreamWriter` erledigen, indem Sie den Namen der gewünschten Kodierung als Konstruktorparameter mitgeben. Eine Übersicht der verfügbaren Kodierungen finden Sie auf der Seite <http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html>.

Sie können eine Zeichenkonvertierung nicht nur bei der Ein-/Ausgabe vornehmen, sondern auch direkt im Speicher. Dies ist nützlich, wenn Sie den Codewert einzelner Zeichen in einer bestimmten Kodierung kennen müssen, um beispielsweise einen Type- oder Creator-Code für die Methoden der Klasse `com.apple.eio.FileManager`² zu berechnen. Die Umwandlung geschieht mit der `String`-Methode `getBytes()`:

```
byte[] b = "tmAM".getBytes("MacRoman");
int creator = (b[0] << 24) | (b[1] << 16) | (b[2] << 8) | b[3];
```

Andere Plattformen verwenden als Standardkodierung »ISO-Latin-1« oder »WinLatin-1« – beides sind Untermengen von UTF-8. Wenn sich Java-Programme darauf verlassen und die so kodierten Bytes einfach in ein `char` umwandeln, können sie Probleme unter MacOS X bekommen.

² <http://developer.apple.com/documentation/Java/Reference/1.4.2/appledoc/api/com/apple/eio/FileManager.html>

Dateinamen werden auf der Festplatte nicht in der System-Kodierung abgelegt, sondern immer in UTF-8. Dies ist normalerweise kein Problem, da Dateien meistens mit `java.io.File` angesprochen werden. Allerdings werden zusammengesetzte Zeichen, beispielsweise »é«, je nach Dateisystem unterschiedlich behandelt: HFS+ speichert dieses Zeichen zerlegt als zwei einzelne Zeichen (»e« und »'«), SMB speichert das zusammengesetzte Zeichen und bei UFS und NFS ist das Verhalten nicht festgelegt. Am besten vermeiden Sie also Sonderzeichen in Dateinamen.

5.2.4 Runtime.exec()

Auch wenn `Runtime.exec()` zur Standardschnittstelle von Java gehört, kann ein Aufruf dieser Methode niemals plattformunabhängig sein – denn es wird damit ein Programm des zugrunde liegenden Betriebssystems gestartet. Wenn Sie aber die passenden Programme auf allen Zielsystemen kennen, können Sie mit einer System-Fallunterscheidung die jeweiligen Programmaufrufe durchführen. Idealerweise sehen Sie dabei auch einen Sonderfall für alle nicht direkt unterstützten Systeme vor – vielleicht gibt es ja eine reine Java-Lösung, die nur nicht ganz so performant ist oder die nicht alle Möglichkeiten zur System-Interaktion hat.

Falls Sie `Runtime.exec()` einsetzen, verwenden Sie immer die Variante mit dem `String`-Array als Parameter! Damit vermeiden Sie Probleme mit Leerzeichen im Datei- bzw. Pfadnamen. Das folgende Beispiel setzt das `open`-Kommando ein, um Programme zu starten oder Dokumente zu öffnen:

```
// Achtung, so bitte nicht:
Runtime.getRuntime().exec("open Pfad auf Ihre Applikation");
```

Listing 5.3 Probleme mit Leerzeichen im Pfad

```
// wenn Runtime.exec(), dann bitte wie folgt:
Runtime.getRuntime().exec(
    new String[] {
        "open",
        "Pfad auf Applikation oder Dokument"
    }
);
```

Listing 5.4 Korrekte Verwendung von `Runtime.exec()`

Das `open`-Kommando von MacOS X hat zwei mögliche Optionen. Die Option `-a` startet die angegebene Applikation:

```
open -a /Applications/TextEdit.app /Users/much/MeinDokument.doc
```

Allerdings klappt diese Zuordnung leider nicht immer, manchmal wird trotzdem eine andere Anwendung aufgerufen. Mit der Option `-e` (»edit«) wird die übergebene Datei immer in `TextEdit` geöffnet:

```
open -e /Users/much/MeinDokument.doc
```

Wenn Sie keine Option angeben, wird die im System konfigurierte Standardapplikation für den jeweiligen Dokumenttyp aufgerufen.

5.3 Oft benötigte Lösungen

Denken Sie daran, dass die im Folgenden vorgestellten Lösungen eventuell nicht immer »100% Java« und damit nicht wirklich plattformunabhängig sind. Dadurch, dass andere Entwickler die jeweiligen Bibliotheken aber bereits auf diverse Systeme portiert haben, sind die Lösungen zumindest zwischen diesen Zielsystemen portabel, was oft ausreicht – und in der Regel besser ist als eine selbst gestrickte Lösung.

5.3.1 PDF-Dokumente

Um PDF-Dokumente anzuzeigen, können Sie ganz einfach ein geeignetes Programm außerhalb der Java-Laufzeitumgebung nachstarten. Dazu verwenden Sie `Runtime.exec()`, wie es in Abschnitt 0 vorgestellt wurde. Wenn es Ihnen nicht darauf ankommt, mit welchem Programm das Dokument letztendlich dargestellt wird, geben Sie dabei einfach nur den Dateinamen des Dokuments an:

```
String[] cmd = null;
if (Sys.isMacOSX()) {
    cmd = new String[] {
        "open",
        "file://localhost/Volumes/Macintosh HD/datei.pdf"
    };
}
else if (Sys.isWindows()) {
    cmd = new String[] {
        "cmd", "/c", "start", "file://C:/datei.pdf"
    };
}
```

```

try {
    Runtime.getRuntime().exec( cmd );
}
catch (Exception e) {
    e.printStackTrace();
}

```

Listing 5.5 PDF-Dokumente unter Mac OS X und Windows anzeigen lassen

Durch die `file`-URLs wird sichergestellt, dass entweder das System die URL auswertet und eine passende Helfer-Applikation aufruft oder dass ein Web-Browser das Dokument mithilfe eines PDF-Plugins darstellt. Für andere Betriebssysteme können Sie generell auf einen Browser zurückgreifen, den Sie am besten mit einer der im folgenden Abschnitt vorgestellten Bibliotheken starten.



Abbildung 5.1 PDF-Ausgabe als Option im Druckdialog

Das Erzeugen von PDF-Dateien ist fester Bestandteil von Mac OS X – im Druckdialog des Systems gibt es den Knopf »Als PDF sichern...«, der damit auch Java-Applikationen zur Verfügung steht (siehe Abbildung 5.1). Dies ist zwar ungewein praktisch, aber leider keine portable Lösung. Zur systemübergreifenden PDF-Erzeugung eignen sich diverse frei verfügbare Bibliotheken, beispielsweise Cocoon (<http://cocoon.apache.org/>) und iText (<http://sourceforge.net/projects/itext/>).

5.3.2 Web-Browser und HTML-Dokumente

Zur Anzeige von HTML-Dokumenten in einem Web-Browser außerhalb der Java-Laufzeitumgebung können Sie wie bei den PDF-Dokumenten `Runtime.exec()` verwenden, wobei Sie eine URL mit einer `*.html`-Datei übergeben.

Es gibt auch eine Mac-spezifische Methode `openURL()` in den Klassen `com.apple.mrj.MRJFileUtils` (für Java 1.3) und `com.apple.eio.FileManager` (für Java 1.4). Diese Lösung ist aber nicht nur nicht portabel, die Methode `openURL()` ist auch teilweise gar nicht funktional implementiert – abgesehen davon, dass der Wechsel der Programmierschnittstellen zwischen zwei Java-Versionen recht lästig ist.

Daher verwenden Sie am einfachsten eine der Bibliotheken, die sich darauf spezialisiert haben, auf möglichst vielen Systemen einen Browser zu starten und ein Dokument anzuzeigen (intern werden dabei zum Teil die oben erwähnten Techniken eingesetzt):

► **Browser Opener**

Diese Bibliothek ist speziell an Windows, Mac OS X und UNIX angepasst, ist Open Source und unterliegt der GNU GPL. Es handelt sich um reinen Java-Code, der mit den im ersten Abschnitt vorgestellten Strategien defensiv programmiert wurde – und der damit trotzdem auf Mac-spezifische Techniken wie das `open`-Kommando, die `openURL()`-Methode oder die Cocoa Java-Klasse `NSWorkspace` zugreifen kann. Die Verwendung ist denkbar einfach, nach einer einmaligen Initialisierung rufen Sie die Methode `displayURL()` auf:

```
Browser.init();
Browser.displayURL("http://www.muchsoft.com/java/");
```

Die aktiv weiterentwickelte Bibliothek dürfte derzeit die beste Wahl zum Öffnen eines Web-Browsers sein. Sie können die Software von der Seite <http://ostermiller.org/utils/Browser.html> herunterladen.

► **Browser Launcher**

Der Browser Launcher war lange Zeit *die* Standardbibliothek zum Nachstarten eines Web-Browsers. Leider ist die Code-Basis mittlerweile veraltet und berücksichtigt neuere Mac OS X-Versionen nur ungenügend. Wenn Sie sich trotzdem den Quelltext ansehen oder diesen auf den neuesten Stand bringen wollen, finden Sie das Projekt auf der Seite <http://browserlauncher.sourceforge.net/>.

► **MRJAdapter**

Die MRJAdapter-Bibliothek haben Sie schon in Kapitel 3, *Grafische Benutzungsoberflächen*, zur einfachen Umsetzung der Mac-spezifischen GUI-Anpassungen kennen gelernt. Die Bibliothek besitzt auch den einfachen Aufruf

```
net.roydesign.mac.MRJAdapter.openURL( url );
```


Dieser Aufruf funktioniert allerdings nur dann systemübergreifend, wenn auch der Browser Launcher auf dem jeweiligen System installiert ist. Insofern sollten Sie dem Browser Opener den Vorzug geben.

► **BrowserControl**

Bereits 1999 wurde auf »JavaWorld« in einem Programmiertipp die Klasse `BrowserControl` vorgestellt, die unter Windows und UNIX einen Browser aufrufen konnte. Es gab zwar noch eine Erweiterung für das klassische MacOS, aber Sie sollten auf jeden Fall dem Browser Opener den Vorzug geben – zumal die beiden oben genannten Bibliotheken letztendlich Erweiterungen von `BrowserControl` sind. Den ursprünglichen Tipp finden Sie auf http://www.javaworld.com/javatips/jw-javatip66_p.html.

5.3.3 Zugriff auf serielle Schnittstellen

Mac-Rechner besitzen keine »normale« serielle Schnittstelle – dadurch (und durch das seit dem iMac fehlende Diskettenlaufwerk) erregten sie in der PC-Welt einiges Aufsehen. Lange Zeit setzte Apple auf ADB (»Apple Desktop Bus«), seit einigen Jahren kommt nun die USB-Schnittstelle (»Universal Serial Bus«) zum Einsatz, die sich dann ja auch im PC-Lager durchgesetzt hat. Mit passenden USB-Adaptern³ sind aber Schnittstellen wie RS232 auch unter Mac OS X problemlos nachrüstbar.

Die Programmierschnittstelle für serielle und parallele Schnittstellen wurde von Sun als »Java Communications API« standardisiert, siehe <http://java.sun.com/products/javacomm/>. Einige Implementationen dieser API sind auch für Mac OS X verfügbar, darunter die kostenlose Open Source-Lösung RXTX (<http://www.rxtx.org/>) sowie die kommerzielle Bibliothek »Serial Port« (<http://www.serialio.com/>).

5.3.4 Hochauflösende Zeitmessung

Unter anderem für Spiele wird eine genaue und schnelle Zeitmessung benötigt. Der übliche Aufruf `System.currentTimeMillis()`, der die aktuelle Zeit in Millisekunden liefert, ist hierfür meist zu ungenau – die effektive Auflösung beträgt je nach Betriebssystem einige hundert Millisekunden. Ab Java 1.4.2 kann daher zur genaueren Zeitmessung die Klasse `sun.misc.Perf` verwendet werden – die aber nur bedingt portabel ist, da sie nicht zur Standard-Klassenbibliothek gehört. Zumindest unter Windows, Linux und Mac OS X können Sie diese Klasse wie folgt einsetzen:

³ Beispielsweise von Keyspan, die auch geeignete Treiber für Mac OS X mitliefern:
<http://www.keyspan.com/>

```

//CD/examples/ch05/time/NanoTime/NanoTime.java
import sun.misc.Perf;
public class NanoTimer {
    private Perf timer;
    private long freq;

    public NanoTimer() {
        timer = Perf.getPerf();
        freq = timer.highResFrequency();
    }

    public long nanoTime() {
        return ( timer.highResCounter() * 1000000000L / freq );
    }
}
// ...
}

```

Listing 5.6 Hochauflösende Zeitmessung mit Java 1.4.2

Der Konstruktor lässt sich von der Klasse `Perf` ein Zeitmessungsobjekt liefern, mit dem er dann die Zählfrequenz ermittelt. Danach können Sie dann wiederholt `nanoTime()` aufrufen und die Zeitdifferenz bestimmen.

Ab Java 1.5 gibt es in der Klasse `java.lang.System` die Methode `nanoTime()`, die denselben Zweck erfüllt – und die den Vorteil hat, wirklich portabel zu sein:

```

long startzeit = System.nanoTime();
// zu messende Anweisungen...
long nanoDifferenz = System.nanoTime() - startzeit;

```

Listing 5.7 Hochauflösende Zeitmessung mit Java 1.5

5.3.5 Rendezvous

»Rendezvous« (oder »OpenTalk«, wie es aus markenrechtlichen Gründen ab MacOS X 10.4 vermutlich genannt wird) ist eine Technologie zur automatischen Netzwerkkonfiguration und zum einfachen Erkennen bereitgestellter Dienste darin (Web-Server, FTP-Server, Drucker usw.). Die Ideen bauen auf AppleTalk auf und wurden unter maßgeblicher Mitwirkung von Apple als offenes Protokoll »Zeroconf« (<http://www.zeroconf.org/>) standardisiert.

Ab Mac OS X 10.4 wird Rendezvous auch aus Java heraus ansprechbar sein. Für Linux, Solaris und FreeBSD können Sie alle nötigen Quelltexte von <http://developer.apple.com/darwin/projects/rendezvous/> herunterladen, und für Windows

steht bereits jetzt eine einfach zu installierende Rendezvous-Preview-Version auf der Seite <http://developer.apple.com/macosx/rendezvous/> zur Verfügung.

Das folgende Beispiel zeigt Ihnen einen einfachen Rendezvous-Browser für http-Dienste. Das Programm ist unter Windows ausführbar, wenn Sie die Rendezvous-Preview-Version installiert haben. Damit Sie die Klasse unter MacOS X 10.3 kompilieren können, müssen Sie sich das Archiv `dns_sd.jar` von einer Windows-Installation besorgen.

```
//CD/examples/ch05/rendezvous/RendezvousTest/RendezvousTest.java
import java.awt.*;
import java.awt.event.*;
import com.apple.dnssd.*;
```

Die Rendezvous-Java-Klassen gehören zum Paket `com.apple.dnssd`. Das »DNSSD« steht für »DNS Service Discovery.«

```
public class RendezvousTest extends Frame
    implements BrowseListener, WindowListener {
    private DNSSDService browser;

    public RendezvousTest() {
        try {
            browser = DNSSD.browse( "_http._tcp", this );
        }
        catch (DNSSDException e) { /* ... */ }
        this.addWindowListener( this );
    }
}
```

Der Konstruktor lässt sich von der Klasse `DNSSD` ein `Service`-Objekt zur Erkennung von http-Diensten erzeugen. Der Aufbau der bei `browse()` übergebenen Zeichenkette ist vom Rendezvous-Protokoll festgelegt. Statt »_http« könnten Sie auch »_ftp« angeben, und Rendezvous sieht es auch vor, eigene Dienstypen zu definieren. Je nachdem, ob Sie zuverlässigen Transport benötigen oder nicht, folgt am Ende entweder »_tcp« oder »_udp«. Außerdem wird mit dem Argument `this` das Objekt benannt, das bei Rendezvous-Ereignissen informiert wird. Aus diesem Grund implementiert die Fensterklasse das Interface `BrowseListener`.

```
public void serviceFound(DNSSDService browser, int flags,
    int ifIndex, String serviceName,
    String regType, String domain)
{ /* ... */ }
```

```

public void serviceLost(DNSSDService browser, int flags,
                        int ifIndex, String serviceName,
                        String regType, String domain)
{ /* ... */ }

public void operationFailed(DNSSDService service,
                             int errorCode)
{ /* ... */ }

```

Die drei obigen Methoden sind vom Interface `BrowseListener` vorgegeben. Sie werden von `Rendezvous` aufgerufen, wenn ein neuer Dienst im Netzwerk angeboten wird, ein bestehender Dienst abgemeldet wurde oder wenn es Probleme mit `Rendezvous` gibt. Auf der Buch-CD finden Sie die Implementationen der Methoden, die die empfangenen Daten einfach im Fenster anzeigen.

```

public void windowClosing(WindowEvent e) {
    if (browser != null) {
        browser.stop();
    }
    this.setVisible(false);
    this.dispose();
    System.exit(0);
}
// ...
}

```

Listing 5.8 Ein einfacher `Rendezvous`-http-Dienst-Browser

Bevor Sie das Programm beenden, sollten Sie das `Service`-Objekt mit `stop()` beenden, damit `Rendezvous` die verwendeten Ressourcen sofort wieder freigeben kann. Sofern Sie sich für eine genauere Beschreibung der Programmierschnittstellen interessieren, finden Sie auf der Seite http://developer.apple.com/documentation/Java/Reference/DNSServiceDiscovery_JavaRef/index.html die entsprechende Javadoc-Dokumentation.

Wenn Sie die Java-Applikation nun starten, sehen Sie eventuell erst einmal ein leeres Fenster – genau dann, wenn es keine `Rendezvous`-http-Dienste in Ihrem Netzwerk gibt. In Abbildung 5.2 läuft das Programm unter Windows XP und hat auf zwei Mac-Rechnern mehrere Benutzerkonten mit aktiviertem »Personal Web Sharing« (konfigurierbar unter **Systemeinstellungen · Sharing**) gefunden.

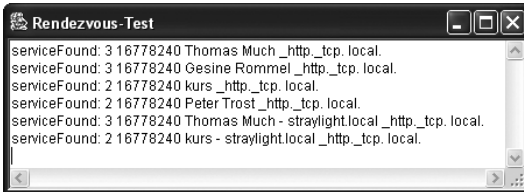


Abbildung 5.2 Rendezvous-Java-Client unter Windows XP

Es gibt aber auch eine reine Java-Implementation des Rendezvous-Protokolls: »JmDNS« (früher »JRendezvous«). Die fertige Bibliothek, die Quelltexte und die Dokumentation des kostenlosen Open Source-Projekts können Sie von <http://jmdns.sourceforge.net/> herunterladen. Als Beispielanwendung ist ein Rendezvous-Dienste-Browser enthalten, der mit einem Doppelklick auf das JAR-Archiv `lib/jmdns.jar` aufgerufen wird (siehe Abbildung 5.3).

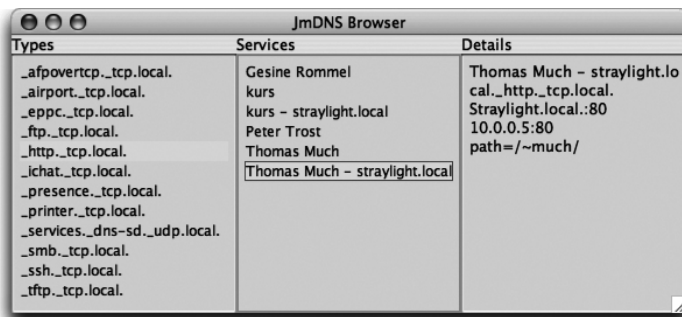


Abbildung 5.3 Rendezvous-Browser in reinem Java

5.4 Literatur & Links

- ▶ <http://developer.apple.com/unix/crossplatform.html>

Apples Webseite zur plattformübergreifenden Programmierung – nicht nur für Java, sondern auch für Perl, Python, Ruby, C und C++

6 Mac OS X-spezifische Programmierung

6.1	JNI	309
6.2	JDirect und JManager	324
6.3	Zugriff auf Datei-Metadaten	327
6.4	Cocoa Java	330
6.5	AppleScript	340
6.6	Speech & Spelling Frameworks	343
6.7	Weitere systemabhängige Bibliotheken	347
6.8	Literatur & Links	349

1	Grundlagen
2	Entwicklungsumgebungen
3	Grafische Benutzungsoberflächen (GUI)
4	Ausführbare Programme
5	Portable Programmierung
6	Mac OS X-spezielle Programmierung
7	Grafik und Multimedia
8	Werkzeuge
9	Datenbanken und JDBC
10	Servlets und JavaServer Pages (JSP)
11	J2EE und Enterprise JavaBeans (EJB)
12	J2ME und MIDP
A	Kurzeinführung in die Programmiersprache Java
B	Java auf Mac OS 8/9/Classic
C	Java 1.5 »Tiger«
D	System-Properties
E	VM-Optionen
F	Xcode- und Project Builder-Einstellungen
G	Mac OS X- und Java-Versionen
H	Glossar
I	Die Buch-CD

6 Mac OS X-spezielle Programmierung

*»Ist das nun Liebe oder ist das nur ein schwacher Trost? Ich sage immer, was ich denke, und ich sagte: Prost, auf dein Spezielles, Lola.«
(Heinz Rudolf Kunze)*

Dieses Kapitel behandelt Programmier Techniken, mit denen Sie Java-Programme um Fähigkeiten ergänzen können, die systemabhängig sind und daher nicht von der Java-Klassenbibliothek angeboten werden können. So verführerisch es auch ist, Systemfunktionen aufzurufen – oft kann man dadurch auf tolle Oberflächen-Effekte zurückgreifen, die Applikation ist besser ins System eingebunden oder wird etwas performanter ausgeführt: Sie verlieren dabei die Portabilität Ihres Java-Programms, d.h., der große Vorteil der Plattformunabhängigkeit von Java geht verloren. Überlegen Sie sich den Einsatz dieser Techniken also gut.

Häufig werden systemabhängige Techniken eingesetzt, weil für eine bestimmte Aufgabe kein portabler Weg bekannt ist. Machen Sie sich die Mühe, im Web nach geeigneten Lösungen zu suchen! Einige grundlegende Techniken haben Sie ja gerade im vorangegangenen Kapitel über portable Programmierung kennen gelernt.

Wenn Sie sich dennoch dafür entscheiden, Ihre Applikation nur für Mac OS X anzubieten, oder wenn Sie einfach über den Tellerrand schauen möchten, wie das Mac-System außerhalb von Java funktioniert, finden Sie im Folgenden die nötigen Informationen.

6.1 JNI

Das Java Native Interface (JNI) dient dazu, Routinen aufzurufen, die in einer anderen Programmiersprache – meistens C oder C++ – realisiert sind und die damit auch vollen Zugriff auf das zugrunde liegende Betriebssystem und die Hardware haben können. Dies ist direkt in Java ja bewusst nicht erlaubt, damit Java-Programme so weit wie möglich stabil und plattformunabhängig sind. JNI ist somit zwar eine Standardschnittstelle von Java, aber eine Schnittstelle zu nicht portablen nativen Routinen, die Sie für jedes System, auf dem Ihre Java-Applikation laufen soll, neu schreiben und kompilieren müssen.

JNI basiert darauf, dass eine Methode mit `native` gekennzeichnet wird und keine Java-Implementierung bekommt – in etwa so, wie Sie es von abstrakten Methoden kennen:

```
public native long getPrimaryMACAddress();
```


Das »MAC« bezeichnet nicht den Apple-Rechner (der wird »Mac« abgekürzt), sondern die Ethernet-Hardware-Adresse¹(»Media Access Control«), mit der ein Rechner (bzw. seine Netzwerkkarte) eindeutig identifiziert werden kann. Diese Information wird ab und an benötigt, wird aber derzeit noch nicht von Java in der Standard-Klassenbibliothek zur Verfügung gestellt.²

Damit diese Methode dann zur Laufzeit ganz normal aufgerufen werden kann, muss vor dem ersten Aufruf eine passende native Bibliothek mit `System.loadLibrary()` geladen worden sein, d.h. kein JAR-Archiv, sondern beispielsweise ein C-Modul. Diese Bibliothek muss eine Routine mit einem Namen enthalten, den die Java-Laufzeitumgebung eindeutig der oben gezeigten `native`-Methode zuordnen kann. Zum Glück müssen Sie sich den eindeutigen Namen und die nötigen Funktionsparameter nicht selbst überlegen, sondern Sie überlassen dies üblicherweise dem Standardwerkzeug `javah`.

6.1.1 JNI-Bibliotheken mit Xcode erzeugen

Xcode unterstützt das Erzeugen von Java-JNI-Anwendungen mit einem eigenen Projekttyp. Legen Sie mit **File • New Projekt...** ein neues Projekt als »Java JNI Application« an. Als Name wird im folgenden Beispiel »JNITest« verwendet.

Wenn Sie sich die automatisch erzeugten Quelltexte genauer ansehen, werden Sie feststellen, dass es sich dabei um ein einfaches, voll funktionsfähiges »Hallo Welt«-Beispiel handelt, in dem zwar eine C-Routine aufgerufen wird – in dieser Routine findet dann aber kein Zugriff auf das Betriebssystem statt. JNI ist dafür also eigentlich unnötig. Daher sehen Sie nun, wie die weiter oben bereits angesprochene Ermittlung der MAC-Adresse auf Mac OS X konkret funktioniert. Nichtsdestotrotz ist das generierte JNI-Projekt eine gute Grundlage, um darauf aufbauend Anpassungen für eigene Anwendungen vorzunehmen.

Löschen Sie dafür zunächst die Targets »JNITest« und »BuildUsingMake« (entweder über den Menüpunkt **Edit • Delete** oder einfach mit ). Das letztgenannte Target können Sie einsetzen, falls Sie das Projekt mit `gnumake` (bzw. `make`) übersetzen wollen; ein passendes `Makefile` wird ebenfalls automatisch von Xcode generiert.

1 Siehe <http://mindprod.com/jgloss/mac.html>

2 Wenn Sie zur Ermittlung der MAC-Adresse auf JNI verzichten wollen, können Sie auch die Kommandos `ifconfig -a` oder `netstat -i` mit `Runtime.exec()` ausführen. Da diese Kommandos aber nicht auf allen Systemen zur Verfügung stehen, ist diese Vorgehensweise genauso systemabhängig.

Dann legen Sie über das Menü **Project · New Target...** ein neues Target vom Typ **Java · Application** an. Damit wird eine MacOS X-Applikation (Bundle) erzeugt, die nicht nur den Java-Code enthält, sondern auch die JNI-Bibliothek – dadurch ist die Software-Installation für den Anwender später extrem einfach. Achten Sie darauf, nicht den Typ **Java · Package** zu verwenden, denn damit würde eine Java-Bibliothek (d.h. ein JAR-Archiv) generiert werden. Dem neuen Target können Sie wieder den Namen »JNITest« geben. Machen Sie nun dieses Target mit dem Popup-Menü oben links im Fenster zum aktiven Target, so dass ein kleines grünes Häkchen im Target-Symbol angezeigt wird.

Unabhängig von dem verwendeten Projektnamen generiert Xcode immer auch eine Datei `JNIWrapper.java`. Die Klasse dient als Java-Hülle oder -Verpackung (»Wrapper«) für die nativen JNI-Routinen. Passen Sie diese Datei an, damit sie in etwa wie folgt aussieht:

```
//CD/examples/ch06/jni/JNITest/JNIWrapper.java
public class JNIWrapper {
    static {
        System.loadLibrary("JNITest");
    }

    public JNIWrapper() { }

    public native long getPrimaryMACAddress();
}
```

Der leere Konstruktor ist nur der Vollständigkeit halber enthalten, damit man den Unterschied zum statischen Initialisierungsblock davor besser erkennt. In diesem Block, der bei der ersten Verwendung der Klasse ausgeführt wird (also noch vor dem Aufruf irgendwelcher Objekt-Eigenschaften), wird die JNI-Bibliothek geladen. Wo diese Bibliothek gesucht wird, hängt von der jeweiligen Implementierung der Java Virtual Machine ab, und welche exakten Dateinamen dabei verwendet werden, legt das zugrunde liegende Betriebssystem fest. MacOS X erwartet als Präfix »lib« und als Suffix »jnilib«. In diesem Beispiel wird also die Datei `libJNITest.jnilib` geladen – und genau dieser Name taucht im Xcode-Projekt auch bei den »Products« auf. Falls Sie die entsprechende Bibliothek für Windows implementieren würden, müssten Sie dort als Dateinamen `JNITest.dll` verwenden, bei Solaris `libJNITest.so`.

Die vorhandene native Methode ersetzen Sie durch `getPrimaryMACAddress()`. Allgemein tragen Sie an dieser Stelle alle Methoden ein, die von nativem Code in der JNI-Bibliothek ausgeführt werden sollen. Um die

Programmierung der C-Routinen in der Datei `JNITestjniLib.c` kümmern Sie sich jetzt noch nicht, löschen Sie einfach den kompletten Inhalt der Datei!

Die Methode `main()` wurde in die Datei `JNITest.java` ausgelagert (siehe Beispiel auf der Buch-CD). Darin wird im Wesentlichen ein Objekt von `JNIWrapper` erzeugt und dann die Methode zur MAC-Ermittlung aufgerufen:

```
JNIWrapper wrap = new JNIWrapper();
long mac = wrap.getPrimaryMACAddress();
```

Sehen Sie sich nun die Targets etwas genauer an, die in mehreren Schritten das fertige Programm erzeugen. Zunächst wird die Java-Klasse mit der `native`-Methode kompiliert, dies erledigt das Target »JNIWrapper«. Es wird ein ganz gewöhnliches JAR-Archiv angelegt, das Produkt `JNIWrapper.jar`.

Anhand dieses JAR-Archivs kann das Programm `javah` einen passenden C-Funktionsnamen für die native Methode generieren. Dies wird im »Headers«-Target erledigt, und als Produkt kommt die Header-Datei `Headers/JNIWrapper.h` heraus.

Das Target »JNILib« erzeugt dann die JNI-Bibliothek `libJNITest.jnilib`. Die Übersetzung erfolgt mit GCC. Damit im C-Quelltext JNI-Routinen aufgerufen werden können, ist das `JavaVM.framework` eingebunden.

Alle nötigen C-Header für die native JNI-Programmierung finden Sie im Verzeichnis `/System/Library/Frameworks/JavaVM.framework/Headers/`. Falls der Compiler die Header – beispielsweise `jni.h` – nicht findet, fehlt vermutlich das Java Developer Package, das Sie separat bei Apple herunterladen und installieren müssen.

Zu jeder Java-Version veröffentlicht Apple ein solches Entwicklerpaket, das zusätzlich zu Apples sonstigen Entwicklerwerkzeugen installiert sein muss. Insbesondere ist dies der Fall nach der Aktualisierung auf Java 1.4.2 – die alten 1.4.1-Header werden dabei gelöscht, die neuen 1.4.2-Header vom Java-Benutzerpaket aber nicht installiert.

Soweit zu den automatisch erstellten Targets. Es werden bei diesem Projekt mehrere benötigt, weil in den Zwischenschritten verschiedene Produkte erzeugt werden, die dann im Folgeschritt verwendet werden.

Als abschließenden Schritt konfigurieren Sie nun das Target »JNITest«, das aus den einzelnen Produkten eine MacOS X-Applikation zusammenstellt (siehe Abbildung 6.1). Setzen Sie dafür zunächst die wichtigsten Einträge bei

Info.plist Entries · Simple View, vor allem den Executable-Namen, die Hauptklasse und den Klassenpfad, eventuell noch das Programmsymbol.

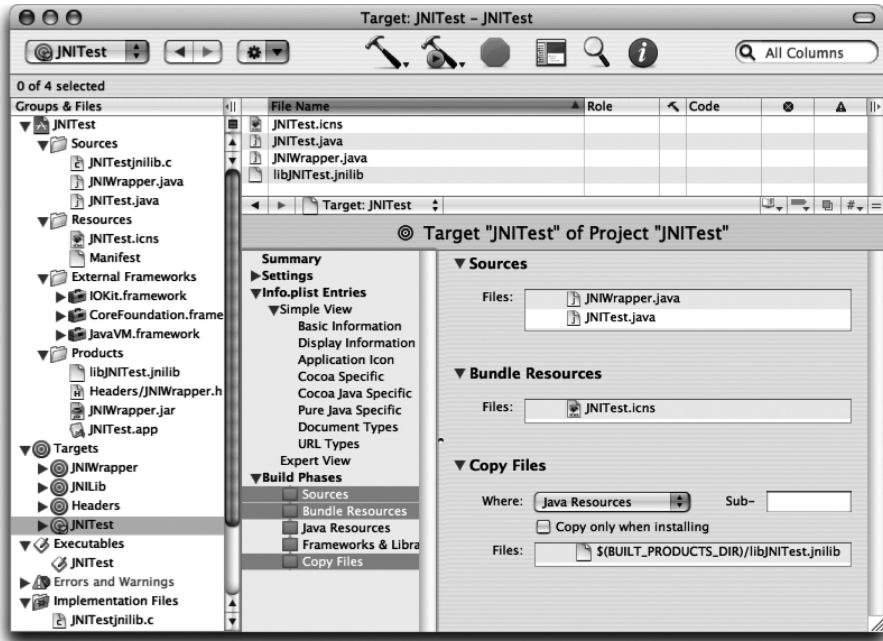


Abbildung 6.1 Build-Phasen des Programmpaket-Targets

Wichtig ist die richtige Konfiguration der Build-Phasen. Bei »Sources« müssen alle benötigten Quelltexte auftauchen; ziehen Sie diese am einfachsten mit der Maus aus der »Groups & Files«-Liste dorthin. Die Datei `JNIWrapper.java` ist damit bei zwei Targets eingetragen und wird auch doppelt übersetzt. Das Ergebnis des »JNIWrapper«-Targets wird aber nur für die Header-Erzeugung benötigt und anschließend verworfen. Bei »JNIWrapper« sind dementsprechend auch nur Quelltexte mit nativen Methoden eingetragen, in diesem Target »JNITest« dagegen alle Quelltexte, die für die Applikation benötigt werden.

Damit dieses Target vollständig ist, muss nun noch die JNI-Bibliothek in das fertige Programmpaket kopiert werden. Dazu rufen Sie den Menüpunkt **Project · New Build Phase · New Copy Files Build Phase** auf (den erscheinenden Dialog können Sie sofort wieder schließen). Bringen Sie die »Copy Files«-Phase ans Ende der Liste, wählen Sie im »Where«-Popup-Menü den Eintrag »Java Resources« aus und ziehen Sie die Datei `libJNITest.jnilib` aus der »Products«-Gruppe auf das »Files«-Feld. Dadurch wird die JNI-Bibliothek in das Verzeichnis `Contents/Resources/Java/` im Programmpaket kopiert.



Abbildung 6.2 Abhängigkeiten des Programmpaket-Targets

»JNITest« ist zwar das aktive Target, die Ausführung hängt aber davon ab, ob die anderen Targets vorher erfolgreich liefen (und ob deren Produkte erzeugt wurden). Diese Abhängigkeiten müssen Sie nun in Xcode definieren. Führen Sie dazu einen `[Ctrl]`-Klick auf das Target aus und rufen Sie im Kontext-Popup-Menü den Eintrag »Get Info« auf. Mit dem »+«-Knopf tragen Sie dann die anderen Targets in genau der Reihenfolge bei »Direct Dependencies« ein, wie es in Abbildung 6.2 zu sehen ist. Die Reihenfolge passt zu den Target-Schritten, die weiter oben schon erklärt wurden.

Damit ist die Konfiguration fertig. Wenn Sie den Menüpunkt **Build • Build** aufrufen, sollte das Projekt ohne Fehler übersetzt werden. Sobald Sie aber versuchen, das Programm zu starten, erhalten Sie eine Exception:

```
[LaunchRunner Error] JNITest.main(String[]) threw an exception:  
java.lang.UnsatisfiedLinkError: getPrimaryMACAddress  
  at JNIWrapper.getPrimaryMACAddress(Native Method)  
  at JNITest.main(JNITest.java:8)
```

Die Java-Laufzeitumgebung kann den Programmcode für die C-Funktion `getPrimaryMACAddress()` nicht finden – natürlich nicht, denn den Inhalt der Datei `JNITestjniLib.c` hatten Sie ja vorhin gelöscht. Also holen Sie die Implementierung der nativen Routine nun nach.

Falls so ein Fehler auch später noch auftritt, können Sie sich mit dem Shell-Programm `nm` anzeigen lassen, ob in der Bibliothek überhaupt die richtigen Funktionsnamen gespeichert sind: `nm -g libJNITest.jnilib`.

Der erste Build-Durchlauf war aber dennoch nicht nutzlos, denn dabei wurde vom Target »Headers« in der »Products«-Gruppe die Header-Datei erzeugt, die Sie sich nun anzeigen lassen können:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class JNIWrapper */
#ifndef _Included_JNIWrapper
#define _Included_JNIWrapper
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:      JNIWrapper
 * Method:     getPrimaryMACAddress
 * Signature:  ()J
 */
JNIEXPORT
    jlong JNICALL Java_JNIWrapper_getPrimaryMACAddress
        (JNIEnv *, jobject);
#ifdef __cplusplus
}
#endif
#endif
```

Listing 6.1 C-Header-Datei `JNIWrapper.h`

Das meiste in dieser Header-Datei ist für Sie unwichtig. Sie kopieren einfach nur die Funktionsdeklaration (im Listing fett gedruckt) in die derzeit leere Datei `JNITestjnilib.c`.

Falls Sie sich etwas besser mit C bzw. C++ auskennen und mehrere JNI-Header-Dateien zusammenfassen möchten, achten Sie auf jeden Fall darauf, dass Sie auch den Block `extern "C" { }` samt der zugehörigen Präprozessoranweisungen übernehmen, damit die nativen Routinen im C-Format in der JNI-Bibliothek gespeichert werden.

Die Implementierung der C-Routine wird hier nicht gezeigt, Sie finden sie vollständig auf der Buch-CD. Der Code basiert auf einem Apple-Beispiel, das Sie von der Seite <http://developer.apple.com/samplecode/GetPrimaryMACAddress/GetPrimaryMACAddress.html> herunterladen können. Damit Sie sich im C-Quelltext einigermaßen zurechtfinden, sehen Sie hier die wesentlichen, für JNI interessanten Elemente:

```
#include "JNIWrapper.h"
#include <CoreFoundation/CoreFoundation.h>
#include <IOKit/IOKitLib.h>
/* ... */
JNIEXPORT
jlong JNICALL Java_JNIWrapper_getPrimaryMACAddress
( JNIEnv *env, jobject thisObj ) {
    jlong mac = 0L;
    /* ... */
    return mac;
}
```

Listing 6.2 C-Implementierungsdatei JNITestjniLib.c

Am Anfang wird mit `#include` die vom Projekt erzeugte Header-Datei eingebunden. Dadurch wird indirekt auch der JNI-Standardheader `jni.h` verwendet, in dem unter anderem die Datentypen `jlong` und `jobject` definiert sind. Danach werden dann alle Header eingebunden, die für die eigentliche C-Programmierung benötigt werden. Schließlich wird die native Methode implementiert, die als Parameter einen `JNIEnv`-Zeiger auf die Java-Laufzeitumgebung und einen `jobject`-Zeiger auf das Objekt bekommt, zu dem die Methode gehört (in Java kennen Sie dies als `this`).

Da die Implementierung die System-Frameworks »CoreFoundation« und »IOKit« verwendet, müssen Sie diese beiden Frameworks auch noch in das Projekt einbinden. Rufen Sie dazu **Project • Add Frameworks...** auf und wählen Sie die beiden Frameworks aus. System-Frameworks finden Sie im Verzeichnis `/System/Library/Frameworks/`, aber dieser Pfad sollte bereits passend vor-eingestellt sein. Beim Hinzufügen müssen Sie noch angeben, in welchen Targets das jeweilige Framework genutzt wird (siehe Abbildung 6.3). Hier im Beispiel ist dies nur im Target »JNILib« der Fall.

Wenn Sie das Projekt nun erneut erzeugen lassen, sollte das Programm danach korrekt starten und Ihnen die MAC-Adresse Ihres Rechners anzeigen.³

3 Kontrollieren können Sie dies unter **Systemeinstellungen • Netzwerk**, wenn Sie dort auf »Konfigurieren...« klicken und dann die Dialogseite »Ethernet« aufrufen.

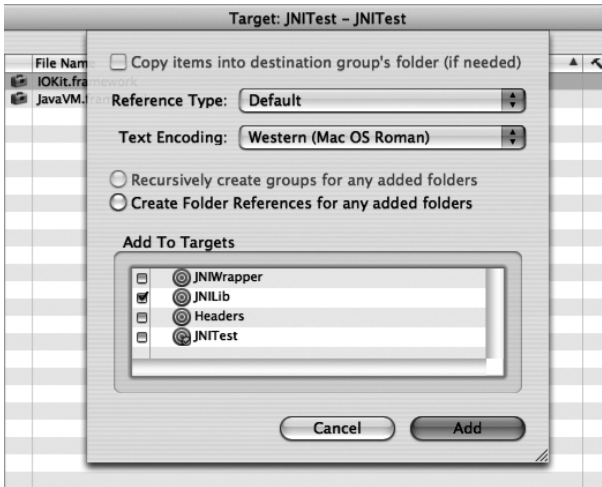


Abbildung 6.3 System-Framework zum JNI-Bibliothek-Target hinzufügen

6.1.2 Installationsverzeichnisse für JNI-Bibliotheken

Wenn Sie Ihre JNI-Bibliothek als Bestandteil eines Mac OS X-Programmpakets ausliefern, kann die Bibliothek zur Laufzeit problemlos geladen werden. Bei anderen Auslieferungsformen, beispielsweise einem JAR-Archiv, sind die JNI-Bibliotheken aber oft als separate Dateien vorhanden. Wenn diese dann an einer falschen Stelle im Dateisystem abgelegt werden, kann das Laden der Bibliothek fehlschlagen, was Ihnen die folgende Fehlermeldung beschert:

```
[Launcher Error] JNITest.main(String[]) threw an exception:
java.lang.UnsatisfiedLinkError: no JNITest in java.library.path
  at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1491)
  at java.lang.Runtime.loadLibrary0(Runtime.java:788)
  at java.lang.System.loadLibrary(System.java:834)
  at JNIWrapper.<clinit>(JNIWrapper.java:5)
  at JNITest.main(JNITest.java:6)
```

Wo also müssen JNI-Bibliotheken installiert werden, damit sie geladen werden können? Als erste Möglichkeit haben Sie bereits das Verzeichnis `Contents/Resources/Java/` innerhalb eines Programmpakets gesehen, also genau dort, wo auch die JAR-Archive liegen. Dies ist auch zugleich die beste Lösung – der Anwender sieht nur eine einzige Programmdatei ohne externe Abhängigkeiten.

Ansonsten können Sie die Bibliotheken in das aktuelle Arbeitsverzeichnis oder in eines der Standard-Erweiterungsverzeichnisse (z.B. `~/Library/Java/Extensions/`) packen. Im Verzeichnis `/usr/lib/java/` werden die Biblio-

theiken zwar auch gefunden, dieser Ordner wird aber für eigene JNI-Bibliotheken nicht empfohlen.

Falls Sie eine Bibliothek an irgendeinem anderen Ort installieren wollen, müssen Sie diesen Pfad beim Aufruf der Java-Laufzeitumgebung explizit mit der System-Property `java.library.path` angeben – entweder mit der Kommandozeilenoption `-D` oder innerhalb eines Programmpaketes in der Datei `Info.plist` (unter `Properties` im Java-Dictionary).

Obiger Fehler tritt auch auf, wenn Sie im Xcode-Projekt die »Copy Files«-Build-Phase weglassen. Unter Umständen fällt Ihnen dies gar nicht sofort auf, denn im selben Verzeichnis, wo das Programmpaket ausgeführt wird, liegt auch eine Kopie der JNI-Bibliothek. Testen Sie Ihre JNI-Applikation also immer auch außerhalb des Projektverzeichnisses!

6.1.3 JNI-Bibliotheken mit gcc erzeugen

Wenn Sie das Generieren von JNI-Bibliotheken in andere Entwicklungsumgebungen integrieren möchten, müssen Sie eventuell direkt auf `gcc` zurückgreifen (bzw. auf `cc` – beides ist bei Mac OS X ein symbolischer Link für `gcc3`). Das Erzeugen einer dynamisch geladenen Bibliothek (»Dynamic Shared Library«, manchmal mit »Dylib« abgekürzt) geschieht dabei mit den folgenden Schritten, die Ihnen bisher Xcode abgenommen hat. Zunächst übersetzen Sie die Quelltexte, hier `quelle1.c` und `quelle2.c`:

```
cc -c
-I/System/Library/Frameworks/JavaVM.framework/Headers quelle1.c
cc -c
-I/System/Library/Frameworks/JavaVM.framework/Headers quelle2.c
```

Aus den C-Quelltexten werden dabei Objektdateien – die Module, aus denen beim Linken das Programm zusammengesetzt wird. Bei Mac OS X bzw. seinem Darwin-Kern wird dieses Dateiformat »Mach-O« genannt. Weil die Quelltexte JNI-Funktionalität verwenden und dafür den Header `jni.h` einbinden, müssen Sie das `Headers`-Verzeichnis aus dem Java-Framework angeben.

Dann muss die eigentliche Bibliothek erzeugt werden, die später unter dem Namen »Test« zur Verfügung stehen soll. Dafür geben Sie alle beteiligten Objektdateien sowie das passende Framework an. In diesem Schritt wird der Linker aufgerufen (siehe `man ld`):

```
cc -dynamiclib -o libTest.jnilib -framework JavaVM
    quelle1.o quelle2.o
```

Wenn Ihre JNI-Anwendung noch mit Mac OS X 10.0 kompatibel sein soll, dürfen Sie keine dynamisch geladene Bibliothek verwenden, sondern müssen die JNI-Bibliothek als Bundle erzeugen, das intern wie ein Programmpaket aufgebaut ist:

```
cc -bundle -I/System/Library/Frameworks/JavaVM.framework/Headers
-o libTest.jnilib -framework JavaVM quelle1.c quelle2.c
```

Sie können dieses Bibliotheksformat auch heute noch verwenden, sollten dies aber nach Möglichkeit nicht mehr tun. Dyllibs erlauben so genanntes »Prebinding«, was schnellere Ladezeiten ermöglicht.⁴ Außerdem gibt es mit ihnen keine Probleme mit Abhängigkeiten zwischen Bibliotheken.

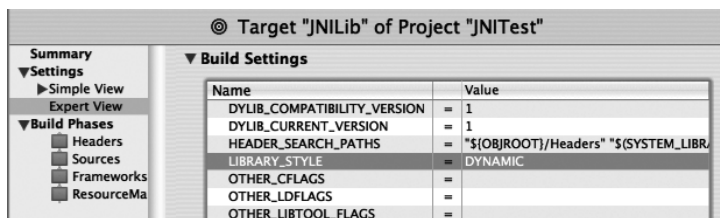


Abbildung 6.4 JNI-Bibliotheken dynamisch oder als Bundle anlegen

Auch in Xcode können Sie JNI-Bibliotheken als Bundle erzeugen. Dazu tragen Sie in der »Expert View« der Target-Einstellungen unter `LIBRARY_STYLE` entweder `DYNAMIC` oder `BUNDLE` ein (siehe Abbildung 6.4). Aus genannten Gründen sollten Sie aber auch hier Dyllibs den Vorzug geben, und dies ist auch die Standardeinstellung.

6.1.4 Abhängigkeiten von anderen Bibliotheken

Betrachten Sie folgende einfache Abhängigkeit zwischen zwei Bibliotheken:

- ▶ `libA.jnilib` enthält eine Funktion `funkA()`.
- ▶ `libB.jnilib` muss `libA.jnilib` linken, um auf `funkA()` zugreifen zu können.

Bei dynamischen JNI-Bibliotheken ist dies kein Problem! Bei JNI-Bundles sind dagegen alle Symbole (d.h. auch die Funktionsnamen) privat innerhalb des Bundles und können damit von außen nicht genutzt werden. Wenn Sie auf das Bundle-Format nicht verzichten können, besteht die Lösung dieses Problems darin, die gemeinsam genutzten Funktionen in eine dynamische Bibliothek

⁴ Siehe `man update_prebinding` und <http://developer.apple.com/documentation/MacOSX/Conceptual/BPFrameworks/Concepts/FrameworkBinding.html>

auszulagern, die dann von den Bundle-Bibliotheken eingebunden wird. Dies geschieht in vier Schritten, wobei das Modul `Gemeinsam.c` nun die Funktion `funkA()` enthält:

1. JNI-Dateien übersetzen:

```
cc -g -I/System/Library/Frameworks/JavaVM.framework/Headers
    -c -o B.o B.c
```

2. Datei mit den gemeinsam genutzen Funktionen übersetzen:

```
cc -g -I/System/Library/Frameworks/JavaVM.framework/Headers
    -c -o Gemeinsam.o Gemeinsam.c
```

3. Dynamische Bibliothek erzeugen:

```
cc -dynamiclib -o libGemeinsam.dylib Gemeinsam.o
```

4. JNI-Bibliothek als Bundle erzeugen, das die gemeinsam genutzte dynamische Bibliothek linkt:

```
cc -bundle -lGemeinsam -o libB.jnilib B.o
```

Hier wird nur die Bibliothek `libB.jnilib` erzeugt, für `libA.jnilib` verfahren Sie genauso.

Ein vollständiges Beispiel für eine JNI-Bibliothek als Bundle, die eine andere Bibliothek verwendet, finden Sie auf der Seite http://developer.apple.com/sample_code/JNISample/JNISample.html.

Ein häufiges Problem bei solchen Abhängigkeiten tritt mit Dylibs auf, die keine JNI-Bibliotheken sind (die also auch nicht direkt mit `System.loadLibrary()` geladen werden) – sie werden oft nicht gefunden. Für diese Bibliotheken gelten nämlich etwas andere Pfade, in denen sie installiert sein sollten. Für applikationsspezifische Bibliotheken sind das aktuelle Arbeitsverzeichnis und der Ordner `Contents/Resources/Java/` in einem Programmpaket auch hier die beste Wahl – kopieren Sie die Dylib also einfach zu den anderen JAR-Archiven und JNI-Bibliotheken.

Für anderswo installierte Bibliotheken, die sich beispielsweise mehrere Anwendungen teilen, müssen Sie mit der Umgebungsvariablen `DYLD_LIBRARY_PATH` eine durch Doppelpunkt getrennte Liste von Verzeichnissen angeben, in denen nach den Bibliotheken gesucht werden soll. Wird eine Bibliothek dort nicht gefunden, wird auch in den Verzeichnissen gesucht, die mit `DYLD_FALLBACK_FRAMEWORK_PATH` und `DYLD_FALLBACK_LIBRARY_PATH` angegeben sind – letztere Variable ist mit `$(HOME)/lib:/usr/local/lib:/lib:/usr/lib` vorbelegt. Genauere Informationen können Sie sich im Terminal mit `man dyld` anzeigen lassen.

Um Abhängigkeiten einer Bibliothek von anderen dynamischen Bibliotheken zu ermitteln, können Sie den Befehl `otool` einsetzen:

```
[straylight:~] much% otool -L libJNITest.jnilib
libJNITest.jnilib:
    /Users/much/JNITest/build/libJNITest.jnilib
        (compatibility version 1.0.0, current version 1.0.0)
    /System/Library/Frameworks/JavaVM.framework/Versions/A/JavaVM
        (compatibility version 1.0.0, current version 66.0.0)
    /System/Library/Frameworks/IOKit.framework/Versions/A/IOKit
        (compatibility version 1.0.0, current version 177.0.0)
    /System/Library/Frameworks/CoreFoundation.framework/
        Versions/A/CoreFoundation
        (compatibility version 150.0.0, current version 299.31.0)
    /usr/lib/libSystem.B.dylib
        (compatibility version 1.0.0, current version 71.1.1)
```

Man sieht sehr schön die Abhängigkeiten von den Frameworks »JavaVM«, »IOKit« und »CoreFoundation« – genau so, wie es im Xcode-Projekt konfiguriert wurde. Zusätzlich taucht noch eine systeminterne Bibliothek und ein Verweis auf die eigene Bibliothek auf.

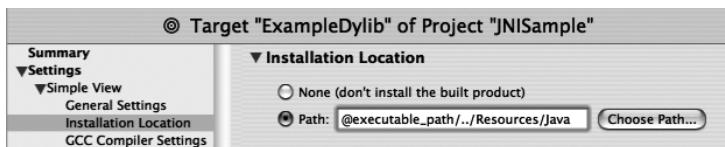


Abbildung 6.5 Installationspfad für Bibliotheken festlegen

Für Bibliotheken kann auch ein Installationspfad angegeben werden (siehe Abbildung 6.5) – das ist der Pfad, der Ihnen von `otool` angezeigt wird. Bei applikationsspezifischen JNI-Bibliotheken ist dies aber unüblich, eher wird dies bei gemeinsam genutzten Bibliotheken eingesetzt. Um solche Pfadnamen nachträglich zu ändern, können Sie das Programm `install_name_tool` verwenden.

6.1.5 Laufzeitumgebungen (JVMs) erzeugen

JNI kann nicht nur verwendet werden, um aus Java heraus native Routinen aufzurufen. Es stehen Ihnen auch native JNI-Routinen zur Verfügung, mit denen Sie Java komplett aus C bzw. C++ steuern können. Folgender Code erzeugt beispielsweise eine Java-Laufzeitumgebung und ruft darin die `main()`-Methode auf:

```
#include <jni.h>
#define USER_CLASSPATH "."
#define USER_MAINCLASS "HalloWelt"
```

Zunächst wird die C-Headerdatei `jni.h` eingebunden, damit die JNI-Routinen genutzt werden können. Danach werden der Klassenpfad und die Hauptklasse fest definiert (für den praktischen Einsatz sollten Sie dies natürlich flexibler gestalten).

```
int main() {
    JavaVMInitArgs vm_args;
    JavaVMOption options[1];
    options[0].optionString = "-Djava.class.path=" USER_CLASSPATH;
    vm_args.version = JNI_VERSION_1_4;
    vm_args.options = options;
    vm_args.nOptions = 1;
    vm_args.ignoreUnrecognized = JNI_TRUE;
```

In der C-`main()`-Funktion wird nun in einer `JavaVMInitArgs`-Struktur beschrieben, welche Java-Version mit welchen System-Properties gestartet werden soll. Hier wird Java 1.4 benutzt, und der Klassenpfad wird, wie oben festgelegt, auf das aktuelle Verzeichnis gesetzt.

Wenn Sie hier als Versionskonstante `JNI_VERSION_1_2` verwenden, wird aus Kompatibilitätsgründen Java 1.3 gestartet. Wenn Sie zum Auffüllen der `JavaVMInitArgs`-Struktur die Routine `JNI_GetDefaultJavaVMInitArgs()` einsetzen, wird automatisch Java 1.4 (bzw. die jeweils aktuellste Version) benutzt.

```
JNIEnv *env;
JavaVM *jvm;
jint res;
res = JNI_CreateJavaVM( &jvm, (void**)&env, &vm_args );
if (res != 0) return res;
```

Nun wird die Laufzeitumgebung mit der Funktion `JNI_CreateJavaVM()` erzeugt. Die als Parameter übergebenen Pointer zeigen danach auf Speicherstrukturen für die Virtual Machine. Insbesondere mit dem `JNIEnv`-Zeiger können Sie im weiteren Verlauf Routinen zur Klassen- und Objektbearbeitung aufrufen.

```
jclass mainClass;
mainClass = (*env)->FindClass( env, USER_MAINCLASS );
```

```

if (mainClass == NULL) {
    res = -1;
    goto _destroyvm;
}

```

Mit der Funktion `FindClass()` wird die weiter oben definierte Hauptklasse gesucht. Dies ist in etwa mit der Methode `Class.forName()` zu vergleichen, wobei Sie im C-Code im Fehlerfall keine Exception erhalten, sondern einen leeren Zeiger als Rückgabewert. Das Abfragen von Java-Exceptions ist auch im C-Code möglich, was am Ende dieses Beispiels eingesetzt wird.

```

jmethodID mainMethod;
mainMethod = (*env)->GetStaticMethodID(env, mainClass, "main",
                                        "([Ljava/lang/String;)V");

if (mainMethod == NULL) {
    res = -1;
    goto _destroyvm;
}

```

Wenn die Klasse gefunden werden konnte, wird darin nun die statische `main()`-Methode gesucht. Die Zeichenkette `([Ljava/lang/String;)V` gibt die Java-interne Signatur der Methode an: ein `String`-Array als Übergabeparameter, `void` als Rückgabe.

```

jclass      stringClass;
jobjectArray args;
stringClass = (*env)->FindClass( env, "java/lang/String" );
args        = (*env)->NewObjectArray( env,
                                        0, stringClass, NULL );

if (args != NULL) {
    (*env)->CallStaticVoidMethod(env, mainClass, mainMethod, args);
}

```

Vor dem Aufruf der Methode muss das übergebene Array von Hand zusammengebaut werden. Erst wird die `String`-Klasse gesucht, dann wird damit ein Objekt-Array erzeugt (das allerdings leer bleibt). Schließlich findet der eigentliche Methodenaufruf mit `CallStaticVoidMethod()` statt – als Parameter werden die gewünschte Klasse und Methode sowie das gerade zusammengestellte Array übergeben.

Zum Schluss wird nachgesehen, ob bei der bisherigen Ausführung eine Exception aufgetreten ist. Falls ja, wird diese ausgegeben. Anschließend wird die Java-Laufzeitumgebung beendet:

```

_destroyvm:
    if ((*env)->ExceptionOccurred(env)) {
        (*env)->ExceptionDescribe(env);
    }
    (*jvm)->DestroyJavaVM(jvm);
    return res;
}

```

Listing 6.3 Erzeugen einer Java Virtual Machine in C++

Ein vollständiges Beispiel zum Erzeugen einer Laufzeitumgebung inklusive einiger Hilfsroutinen stellt Apple auf der Seite <http://developer.apple.com/samplecode/simpleJavaLauncher/simpleJavaLauncher.html> zur Verfügung. In dessen `main()`-Funktion wird vor allem auch gezeigt, wie die JVM in einem neuen `pthread` gestartet wird. Cocoas AppKit-Framework, auf dem Apples Java-Implementation aufbaut, muss zwingend im Hauptthread laufen. Wenn auch die Java-Laufzeitumgebung diesen Thread benutzen würde, hätte die Oberfläche ein sehr schlechtes Antwortverhalten.

6.2 JDirect und JManager

JDirect⁵ und JManager sind veraltete Apple-spezifische Technologien, die Sie nicht mehr einsetzen sollten bzw. auch gar nicht mehr können. Beide zusammen ergeben ungefähr JNI, wobei JDirect eher eine Ergänzung von JNI darstellt. JDirect ist eine Schnittstelle, um direkt im Java-Quelltext auf beliebige Systemroutinen zuzugreifen. Im Gegensatz zu JNI kann dabei sogar auf native C-Routinen und die aufwändige Header-Erzeugung verzichtet werden. JManager diente dazu, eine Java-Laufzeitumgebung in Anwendungen einzubetten, beispielsweise in Web-Browsern.

Wenn Sie wirklich noch Informationen zu diesen Technologien benötigen, um beispielsweise alte Java-Anwendungen zu pflegen oder zu aktualisieren, laden Sie sich am besten das »MRJ Software Development Kit (SDK) 2.2« von der Seite <http://developer.apple.com/java/download.html> herunter. Die Software ist zwar nur auf alten Mac OS-Versionen (Classic) zu verwenden, die enthaltene Dokumentation gibt Ihnen aber einen guten Überblick.

⁵ Trotz des sehr ähnlichen Namens sollten Sie Apples Technologie nicht mit der JDirect-Technologie von Microsoft verwechseln.

6.2.1 JManager

JManager existiert auf MacOS X überhaupt nicht, es ist eine reine Classic-Technologie. Für die alten Mac OS-Versionen gab es zwar auch JNI, aber die Einbindung einer JVM in eine Applikation wurde eigentlich immer mit JManager erledigt, weil diese Schnittstelle auch die Integration in die Benutzungsoberfläche behandelte.

Bei MacOS X wurde JManager zunächst durch das Java Embedding Framework abgelöst, das Sie im Verzeichnis `/System/Library/Frameworks/Java-Embedding.framework/` finden. Die für die Programmierung notwendigen Header-Dateien liegen im Unterverzeichnis `Headers`. `JavaControl.h` behandelt die Einbindung von Java als Oberflächen-Komponente, `JavaApplet.h` fügen Routinen für die Applet-Verwaltung hinzu – beides ist in `JavaEmbedding.h` zusammengefasst.

Das Java Embedding Framework steht allerdings nur für Java 1.3 zur Verfügung und ist damit selbst schon wieder veraltet. Apple empfiehlt, nur noch das Java Plugin einzusetzen. Wenn Sie nicht gerade einen Web-Browser schreiben, werden Sie dies aber vermutlich eh nicht benötigen. Ansonsten finden Sie weiterführende Informationen zur Plugin-Schnittstelle auf der etwas älteren Seite <http://developer.netscape.com/docs/manuals/communicator/plugin/>, und auf <http://mozilla.org/projects/plugins/> ist der aktuelle, Browser-übergreifende Plugin-Standard (NPAPI) beschrieben.

6.2.2 JDirect

JDirect kann nur bis Java 1.3 genutzt werden, ab Java 1.4 ist diese Schnittstelle nicht mehr vorhanden. Die Technologie stammt noch von den alten Mac OS-Versionen und wurde dort bis zur Version JDirect2 gepflegt, MacOS X beinhaltet JDirect3. Ganz grob erfolgt die Programmierung wie folgt:

```
import com.apple.mrj.jdirect.Linker;
public class SysFunktion {
    private static Object link = new Linker( SysFunktion.class );
    private static final String JDirect_MacOSX =
        "/System/Library/Frameworks/CoreServices.framework/"
        + "CoreServices";
    public static native int TickCount();
}
```

Listing 6.4 Zugriff auf Systemfunktionen mit JDirect3

Zunächst müssen Sie ein `Linker`-Objekt erzeugen, dem Sie als Parameter die Klasse mit den Methodendeklarationen für die nativen Routinen übergeben. Die Referenz merken Sie sich in einer Klassenvariablen, damit das Objekt nicht von der Garbage Collection gelöscht wird.

Die Klasse muss eine String-Konstante mit dem Namen `JDirect_MacOSX` enthalten, deren Wert die Bibliothek bezeichnet, die vom Linker geladen werden soll. Hier wird eine Systembibliothek eingebunden, aber natürlich kann dies auch eine eigene Bibliothek sein (z.B. `libmeineBibliothek.dylib`).

Dann folgen die Methodendeklarationen für die nativen Routinen, die – wie bei JNI – mit `native` gekennzeichnet werden. Tatsächlich ist hier JNI im Spiel, denn intern werden von `JDirect` JNI-Funktionen für alle `native`-Methoden angelegt.

`SysFunktion.TickCount()` ruft jetzt die Systemfunktion auf, die die Anzahl der Ticks (ca. 1/60 Sekunden) seit Rechnerstart zurückgibt – ohne dass Sie eine Zeile C-Code geschrieben haben. Die Verbindung zwischen der Java-Methode und der nativen Implementation geschieht allein durch den Methodennamen und die Parameter, die kompatibel mit den Vorgaben in der Bibliothek sein müssen.

Alte `JDirect2`-Anwendungen können nicht ohne weiteres unter Mac OS X kompiliert werden. Es sind kleine Anpassungen nötig, die im Dokument <http://developer.apple.com/technotes/tn/tn2002.html> beschrieben werden.

Falls Sie in aktuellen Java-Anwendungen nicht auf `JDirect`-Funktionalität verzichten können, sollten Sie sich <http://www.jnidirect.org/> ansehen. Diese `JDirect`-Neuimplementierung benötigt keine speziellen Schnittstellen innerhalb der Java-Implementierung, sondern greift auf Standardschnittstellen (JNI) und den C-Compiler des Systems zurück.

Eine Besonderheit gilt es zu beachten, wenn Sie Carbon-Funktionen im System aufrufen (wie Sie es gerade mit `JDirect` und `TickCount()` getan haben). In diesem Fall müssen Sie die Carbon-Aufrufe und die Java-Threads synchronisieren, denn Carbon-Threads sind nicht reentrant. Die Synchronisation erfolgt über die Klasse `CarbonLock`:

```
import com.apple.mrj.macos.carbon.CarbonLock;
// ...
try {
    CarbonLock.acquire();
```

```

        // hier können Sie Carbon-Aufrufe durchführen
    }
    finally {
        CarbonLock.release();
    }

```

Die Sperre (»Lock«) sollten Sie nur so kurz wie möglich halten, damit andere Threads nicht blockiert werden. Achten Sie außerdem unbedingt darauf, dass `release()` im `finally`-Block aufgerufen wird – am besten als erste (oder einzige) Anweisung, damit nicht noch irgendwelche anderen Exceptions auftreten können.

Weil die `CarbonLock`-Routinen nur für Java 1.3 genutzt werden können, rät Apple dringend davon ab, Carbon-Aufrufe mit Java 1.4 durchzuführen – auch nicht mit JNI.

6.3 Zugriff auf Datei-Metadaten

In Kapitel 4, *Ausführbare Programme*, haben Sie beim »Jar Bundler« bereits die `Type`- und `Creator`-Codes kennen gelernt, mit denen beim klassischen Mac OS und auch bei Mac OS X Informationen über den Dateityp (Applikation, Bilddatei, Textdatei usw.) und über das erzeugende Programm festgelegt werden. Diese Mac-spezifischen Informationen werden im Dateisystem außerhalb der eigentlichen Nutzdaten abgelegt und daher auch Metadaten (Attribute) genannt. Mit dem Programm `GetFileInfo` können Sie diese Daten abfragen:

```

[straylight:~] much% /Developer/Tools/GetFileInfo tiger.jpg
file: "tiger.jpg"
type: "JPEG"
creator: "GKON"
attributes: avbstClinmed
created: 01/03/2003 01:30:12
modified: 01/03/2003 01:30:12

```

Als `Type` der Bilddatei wird hier `JPEG` gemeldet (denkbar wären auch andere Bildtypen wie `TIFF`). Als erzeugendes Programm ist `GKON` eingetragen, der »GraphicConverter« von Lemke Software. Ob der `GraphicConverter` das Bild wirklich erzeugt hat, weiß man nicht genau, der Eintrag bedeutet nur, dass das Bild bei einem Doppelklick automatisch mit diesem Programm geöffnet wird. Wenn Sie für Ihre Applikation einen `Creator`-Code registrieren möchten, finden Sie auf der Seite <http://developer.apple.com/dev/cftype/> alle nötigen Informationen dazu. Bei der `Attribut`-Zeichenfolge ist das `C`-Bit gesetzt, die Bilddatei

besitzt also ein eigenes Dateisymbol («custom icon»). Genauere Informationen zu den Attributen erhalten Sie mit `man GetFileInfo`.

`GetFileInfo` (und `SetFile` zum Setzen der Attribute) können Sie wie alle Kommandozeilenprogramme mit `Runtime.exec()` aufrufen. Beispielsweise können Sie bei vom Java-Programm erzeugten Textdateien den Type-Code `TEXT` setzen, damit MacOS X – unabhängig von der Dateinamenserweiterung! – über den Inhalt der Datei Bescheid weiß und der Anwender die Datei bequem per Doppelklick öffnen kann. Der Nachteil dieser Lösung ist, dass der Anwender die Entwicklerwerkzeuge installiert haben muss.

Daher hat Apple spezielle Bibliotheken zum Zugriff auf diese Attribute entwickelt, die fester Bestandteil von MacOS X sind. Die Klassen für Java 1.3 gehören zum Paket `com.apple.mrj` und werden wie folgt verwendet:

```
//CD/examples/ch06/typecreator/MacFileInfo/MacFileInfo.java
import com.apple.mrj.*;
// ...
File    datei    = new File(args[0]);
MRJOSType type   = MRJFileUtils.getFileType(datei);
MRJOSType creator = MRJFileUtils.getFileCreator(datei);
```

Listing 6.5 Abfrage von Type und Creator mit Java 1.3

Mit den statischen Methoden aus der Klasse `MRJFileUtils` fragen Sie also entweder den Type oder den Creator ab (natürlich gibt es auch entsprechende Methoden zum Setzen der Werte). Als Rückgabe erhalten Sie ein `MRJOSType`-Objekt, das im Wesentlichen einfach nur eine Hülle für den jeweiligen vier Zeichen langen Code ist. Diese Aufrufe funktionieren zwar auch noch mit Java 1.4, gelten dort aber als veraltet (siehe Abbildung 6.6). Wenn Ihre Anwendung mit Java 1.3 kompatibel sein soll, haben Sie aber keine andere Wahl.

Die Klassen sind sowohl im Archiv `/System/Library/Frameworks/JavaVM.framework/Versions/1.3.1/Classes/ui.jar` als auch in `/System/Library/Java/Extensions/MRJToolkit.jar` abgelegt und sind damit Bestandteil des Standard-Klassenpfades. Die Dokumentation finden Sie auf der Seite http://developer.apple.com/documentation/Java/Reference/1.3.1/Java131_API_J2SEAndAppleExtensions/api/com/apple/mrj/MRJFileUtils.html.

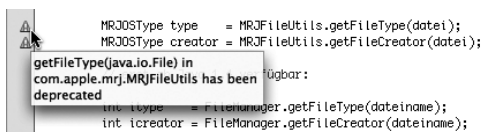


Abbildung 6.6 Warnungen in Xcode vor veralteten MRJ-Methoden

Ab Java 1.4 stellt Apple neu implementierte Methoden für den Zugriff auf den Type- und Creator-Code zur Verfügung, die nun zur Klasse `com.apple.eio.FileManager` gehören. Die Verwendung bleibt weitestgehend gleich, geändert haben sich vor allem die Datentypen – es wird nun nicht mehr mit `MRJOSType`-Objekten gearbeitet, sondern mit `int`-Werten:

```
import com.apple.eio.*;
// ...
String dateiname = args[0];
int itype        = FileManager.getFileType(dateiname);
int icreator     = FileManager.getFileCreator(dateiname);
```

Listing 6.6 Abfrage von Type und Creator mit Java 1.4

Die Klasse ist wieder im Archiv `ui.jar` gespeichert, das in der aktuellsten Version allerdings im Verzeichnis `/System/Library/Frameworks/JavaVM.framework/Classes/` liegt. Ab Java 1.4 sollte diese Klasse nach Möglichkeit anstelle der veralteten Klassen aus `com.apple.mrj` eingesetzt werden. Die Dokumentation finden Sie auf der Seite <http://developer.apple.com/documentation/Java/Reference/1.4.2/appledoc/api/com/apple/eio/FileManager.html>.

Zum Ermitteln der passenden `int`-Werte setzen Sie dennoch am einfachsten die veraltete `MRJToolkit`-API ein:

```
int code = new com.apple.mrj.MRJOSType("TEXT").toInt();
```

Die berechneten Werte sollten Sie dann als Konstanten in Ihren Quelltext eintragen, damit im produktiven Code keine Aufrufe der alten Schnittstelle vorkommen:

```
final int kTypeTEXT = 1413830740;
```

Java-Code mit diesen Klassen und Methoden lässt sich zwar nicht auf fremden Systemen ausführen, Apple stellt aber so genannte »Stub«-Klassen zur Verfügung, damit Sie die Quelltexte dort zumindest übersetzen können. Diese Stubs enthalten nur die Schnittstellen und keine sinnvolle Implementation. Für das Java 1.3-Paket `com.apple.mrj` können Sie die Stub-Klassen von der Seite <http://developer.apple.com/samplecode/MRJToolkitStubs/MRJToolkitStubs.html> herunterladen, für das Java 1.4-Paket `com.apple.eio` stehen sie auf <http://developer.apple.com/samplecode/AppleJavaExtensions/AppleJavaExtensions.html> zur Verfügung. Wichtig: Diese Stub-Klassen sollten Sie wirklich nur zum Kompilieren verwenden und nicht zusammen mit Ihrer Programm-Distribution ausliefern!

In den erwähnten Klassen existiert jeweils auch eine Methode `findFolder()`, mit der Sie bestimmte spezielle Verzeichnisse von Mac OS X ermitteln können. Unter Java 1.3 fragen Sie beispielsweise das temporäre Verzeichnis des Benutzers wie folgt ab:

```
import com.apple.mrj.*;
//...
File tmp =
    MRJFileUtils.findFolder( MRJFileUtils.kTemporaryFolderType );
```

Für Java 1.4 ist folgender Aufruf nötig:

```
import com.apple.eio.*;
//...
final int kTemporaryFolderType = 1952804208;
String tmp = FileManager.findFolder( kTemporaryFolderType );
```

6.4 Cocoa Java

Neben Java und Carbon ist Cocoa die dritte Programmierschnittstelle für Mac OS X. Das Cocoa-Framework ist dabei die Standardschnittstelle für neue Applikationen, die speziell für Mac OS X entwickelt werden. Üblicherweise wird Cocoa mit Objective-C programmiert, Apple stellt aber auch eine Schnittstelle für Java zur Verfügung, die so genannte »CocoaJavaBridge« oder kurz »Cocoa Java«. Das Wort »Bridge« (Brücke) ist dabei wörtlich zu nehmen, denn Apple hat das Cocoa-Framework nicht in Java neu programmiert, sondern Java-Hüllklassen entwickelt, die intern auf die vorhandenen Cocoa-Objective-C-Klassen zugreifen.

Wann ist die Verwendung von Cocoa Java sinnvoll und wann nicht? Cocoa Java können Sie nur einsetzen, wenn Sie eine Anwendung speziell für Mac OS X programmieren. Apple empfiehlt, Cocoa-Komponenten nicht mit Swing- bzw. AWT-Komponenten zu mischen, d.h., Ihr Java-Programm ist entweder Mac-spezifisch und verwendet nur Cocoa Java, oder es ist portabel und setzt Swing bzw. AWT ein. Wenn Sie aber sowieso nur für den Mac entwickeln, sollten Sie sich den Einsatz von Cocoa Java gut überlegen – man merkt vor allem an neuen Cocoa-Klassen, dass diese zuerst für Objective-C und erst später auch für Cocoa Java zur Verfügung stehen. Insofern ist Cocoa Java nur zweite Wahl. Andererseits nimmt Ihnen Cocoa Java das Lernen einer neuen Sprache ab, Sie können also einfach mit der bekannten Java-Syntax weiter programmieren. Vielleicht benutzen Sie daher Cocoa Java, um das Cocoa-Framework kennen zu lernen – als Motivation, um bei größeren Cocoa-Projekten dann Objective-C einzu-

setzen. Aus diesem Grund wird die Software-Entwicklung mit Cocoa Java hier kurz vorgestellt.

Xcode unterstützt die Entwicklung von Cocoa Java-Applikationen mit einem speziellen Projekttyp. Im folgenden Beispiel entwickeln Sie eine kleine Rechner-Applikation, die eine Zahl aus einem Eingabefeld einliest, verdoppelt und im selben Feld wieder ausgibt. Legen Sie ein neues Projekt mit **File • New Project...** als **Application • Cocoa-Java Application** an und geben Sie ihm den Namen »CocoaJavaRechner«. Es werden diverse Gruppen und Dateien erzeugt. Die Gruppe »Classes« ist zunächst noch leer, hier können Sie später Ihre Java-Klassen einsortieren. »Other Sources« enthält die Objective-C-Datei `main.m`, mit der das Programm initialisiert wird – normalerweise müssen Sie diese Datei nicht verändern. Gleiches gilt für die Datei `CocoaJavaRechner_Prefix.h`. Sie wird von Xcode eingesetzt, um Übersetzungszeiten zu minimieren, für dieses Beispiel ist sie aber nicht weiter relevant. Abbildung 6.12 später in diesem Abschnitt kann Ihnen als Überblick über die Projektstruktur dienen.

Was bei diesem Projekt besonders auffällt, ist die Datei `MainMenu.nib` im Ordner »Resources«. Eine NIB-Datei enthält Informationen über die Benutzungsoberfläche: serialisierte Oberflächenelemente, Referenzen auf »externe« Objekte (beispielsweise Steuerungsobjekte) und die Verbindungen dazwischen. Der Name »NIB« stammt vom NeXTSTEP Interface Builder, mit dem solche Oberflächen für den Vorgänger von Mac OS X bzw. Cocoa entworfen wurden. Jede Cocoa-Anwendung mit Benutzungsoberfläche besitzt mindestens eine NIB-Datei. Führen Sie einen Doppelklick auf `MainMenu.nib` aus, damit der Interface Builder (manchmal mit IB abgekürzt) gestartet wird (das Programm befindet sich, wie Xcode auch, im Verzeichnis `/Developer/Applications`).

Nach dem Start des Interface Builders sind bereits einige Fenster offen. Rufen Sie den Menüpunkt **Tools • Show Info** auf, damit auch das nützliche Info-Fenster angezeigt wird. Zunächst ist nun das Fenster mit dem Titel »Window« interessant, dies wird der Hauptdialog für die Beispielapplikation. Wählen Sie im Info-Fenster oben im Pop-up-Menü den Eintrag »Attributes« aus. Ändern Sie dort dann den Text bei »Window Title« auf »Rechner« und schalten Sie das Kästchen »Resize« bei den »Controls« aus. Achten Sie außerdem darauf, dass das Kästchen »Visible at launch time« aktiv ist.

Nun gestalten Sie die Benutzungsoberfläche, indem Sie Komponenten aus diversen Paletten in das »Rechner«-Fenster ziehen. Das Paletten-Fenster ist normalerweise sichtbar – falls nicht, können Sie es mit **Tools • Palettes • Show Palettes** einblenden. Ziehen Sie aus der Palette »Cocoa-Text« ein eckiges Textfeld in das

Dialogfenster (siehe Abbildung 6.7). Das Besondere am Interface Builder ist, dass beim Zusammenstellen der Oberfläche vernünftige Hilfslinien eingeblendet werden, damit das Layout den von Apple standardisierten »Aqua Human Interface Guidelines«, also den Richtlinien zur Oberflächengestaltung, entspricht. Wenn Sie sich an diese Hilfen halten, wird das Layout Ihrer Dialoge von den Benutzern als angenehm und zum System passend empfunden. Ziehen Sie das Textfeld also gerade so weit in die linke obere Ecke des Dialogs, dass Ihnen eine horizontale und eine vertikale Hilfslinie angezeigt wird. Wenn Sie das Element nun »fallen lassen«, rastet es automatisch auf der richtigen Position ein.

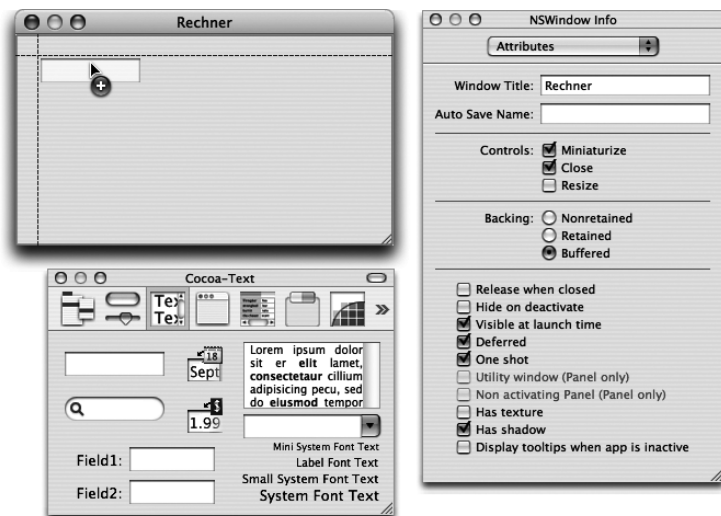




Abbildung 6.7 Oberflächenentwurf mit dem Interface Builder

Machen Sie das Fenster ein bisschen kleiner und ziehen Sie dann das Textfeld so breit, dass rechts am Fensterrand eine Hilfslinie angezeigt wird. Anschließend fügen Sie noch eine horizontale Trennlinie (die Sie in der »Cocoa-Controls«-Palette oberhalb der horizontalen Bildlaufleiste finden) unterhalb des Textfelds ein – es werden dabei etwas andere Hilfslinien angezeigt, die näher an den Fensterrand heranreichen.

Zum Schluss ziehen Sie noch eine Schaltfläche aus derselben Palette ins Fenster. Mit einem Doppelklick auf das Element können Sie die Beschriftung ändern. Tragen Sie hier »*2« ein – der Rechner soll einfach nur Zahlen verdoppeln können. Markieren Sie den Knopf und rufen Sie **Layout • Size to Fit** auf. Nun können Sie das Element unter der Trennlinie rechts im Fenster anordnen (achten Sie auf die Hilfslinien!). Wenn Sie im Info-Fenster bei den Attributen unter »Key Equiv« im Popup rechts daneben »Return« auswählen, kann der Knopf nicht nur per Maus, sondern auch mit der -Taste bedient werden. Abschlie-

ßend verringern Sie die Fensterhöhe noch so, dass unterhalb der Schaltfläche eine Hilfslinie angezeigt wird.

Man kann im Interface Builder auch pixelgenau positionieren. Dazu markieren Sie einfach ein Element und bewegen es dann mit den Pfeiltasten. Auch hierbei werden bei passender Positionierung Hilfslinien angezeigt. Wenn Sie ein Element markieren und  gedrückt halten, werden Ihnen die Pixelabstände zu dem Element angezeigt, über dem sich gerade der Mauszeiger befindet (siehe Abbildung 6.8).

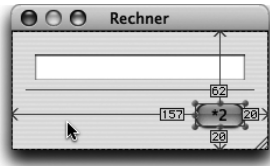





Abbildung 6.8 Layout-Informationen und pixelgenaues Positionieren

Die Benutzungsoberfläche ist soweit fertig gestaltet, sichern Sie Ihr Werk mit **File • Save**. Mit **File • Test** können Sie die Oberfläche ausprobieren (beendet wird dieser Modus mit  + ). Zusätzlich ist unter dem Menüpunkt **File • Compatibility Checking...** eine Kompatibilitätsprüfung zu den verschiedenen Mac OS X-Versionen vorgesehen.

Nach der Oberfläche müssen Sie nun noch die Klassen für die Steuerung und für das Datenmodell entwerfen. Falls Ihnen diese Trennung bekannt vorkommt: Cocoa verwendet das MVC-Muster («Model – View – Controller», Datenmodell/Geschäftslogik – Darstellung – Steuerung), das beispielsweise auch Swing zugrunde liegt. Dazu aktivieren Sie im Fenster »MainMenu.nib« die »Classes«-Dialogseite (siehe Abbildung 6.9). Wählen Sie in der Spalte ganz links den Eintrag `NSObject` aus und drücken Sie die -Taste; es wird eine Unterklasse von `NSObject`⁶ angelegt. `NSObject` ist die Cocoa-Wurzelklasse, vergleichbar mit `java.lang.Object` bei Java (von der die Cocoa Java-Klassen natürlich auch erben). Geben Sie der neuen Klasse den Namen `RechnerController`. Selektieren Sie die neue Klasse und achten Sie bei den Attributen im Info-Fenster darauf, dass bei »Language« der »Java«-Knopf aktiv ist – dies ist wichtig, denn `NSObject`-Unterklassen werden normalerweise als Objective-C-Code erzeugt.

Nun legen Sie die Eigenschaften der Controller-Klasse fest. Im Info-Fenster werden im Bereich »n Outlets« («Ausgänge« oder »Steckdosen«) Referenzen auf

⁶ Das Präfix »NS« leitet sich von NeXTSTEP ab, dem Cocoa-Vorgänger. Apple hat die Benennung der Klassen aus Kompatibilitätsgründen beibehalten.

andere Objekte definiert. Legen Sie mit dem »Add«-Knopf zwei Stück davon an, ein Outlet `feld` für das Ein-/Ausgabefeld sowie ein Outlet `rechenlogik` für die Klasse mit den Rechenroutinen (die in diesem Beispiel das Modell bzw. die Geschäftslogik darstellen). Den Wert für die Spalte »Type« können Sie aus dem Popup-Menü auswählen oder von Hand eingeben – letzteres ist bei der Klasse `Berechnung` nötig, denn diese Klasse werden Sie gleich erst noch schreiben. Sie könnten den Typ auch auf »id« stehen lassen, was in Objective-C eine Objektreferenz ohne speziellen Typ bezeichnet, dann müssten Sie aber später den generierten Java-Code von Hand anpassen.

Im Bereich »Actions« deklarieren Sie Methoden, die andere Objekte (z.B. Oberflächenelemente) in Ihrem Controller aufrufen können. Fügen Sie mit dem »Add«-Knopf eine Methode hinzu, die Sie `klick` nennen. Sobald Sie `↵` drücken, fügt der Interface Builder automatisch leere Klammern ein. Diese Methode wird später aufgerufen, wenn der Benutzer den »*2«-Knopf anklickt. Der Methodenname kann dabei beliebig gewählt werden, denn im Gegensatz zum Java-Ereignismodell werden keine Listener-Interfaces o.Ä. benötigt.

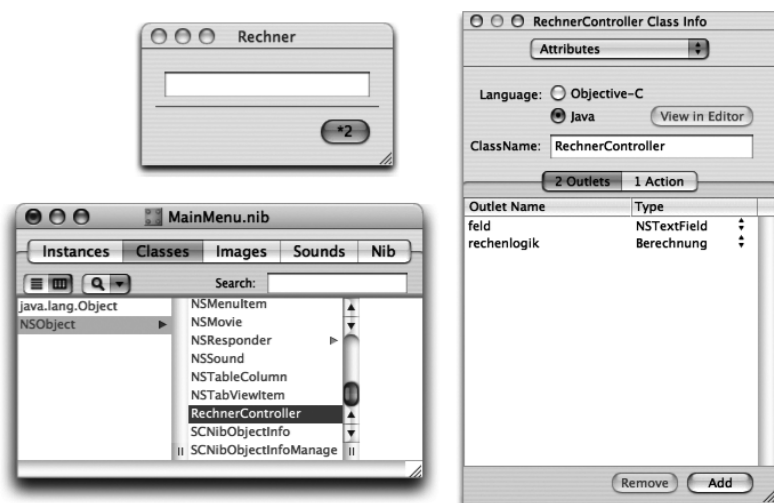


Abbildung 6.9 Steuerungsklasse (Controller) deklarieren

Legen Sie nun noch im Fenster »MainMenu.nib« auf der »Classes«-Dialogseite eine Unterklasse von `java.lang.Object` an, die Sie `Berechnung` nennen – hier wird später den Rechenlogik implementiert. Outlets und insbesondere Actions werden für diese Klasse nicht definiert, denn die Methoden dieser Klasse werden vom Controller und nicht von Cocoa-Elementen aufgerufen.

Die Klassen alleine nützen Ihnen nichts, Sie müssen Objekte daraus erzeugen. Wählen Sie dazu auf der »Classes«-Dialogseite die Klasse `RechnerController`

aus und rufen Sie den Menüpunkt **Classes · Instantiatiere RechnerController** auf. Führen Sie dies ebenso für die Klasse `Berechnung` durch. Die so erzeugten Objekte tauchen nun auf der »Instances«-Dialogseite auf (siehe Abbildung 6.10). Neben dem `RechnerController` werden Sie ein kleines Ausrufungszeichen entdecken: Der Interface Builder weist Sie darauf hin, dass die Outlets (also die Referenzen auf andere Objekte) noch nicht mit geeigneten Objekten verknüpft sind.

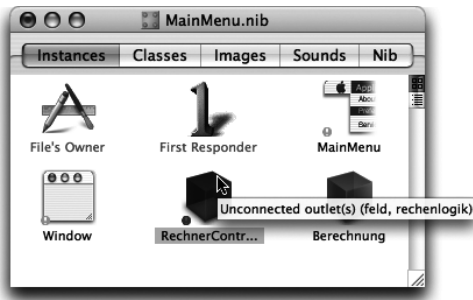


Abbildung 6.10 Controller-Objekt mit fehlenden Referenzen

Diese fehlenden Verknüpfungen stellen Sie nun grafisch her. Klicken Sie das `RechnerController`-Symbol bei gedrückter `[Ctrl]`-Taste an und ziehen Sie den Mauszeiger auf das Textfeld im »Rechner«-Dialog (siehe Abbildung 6.11).

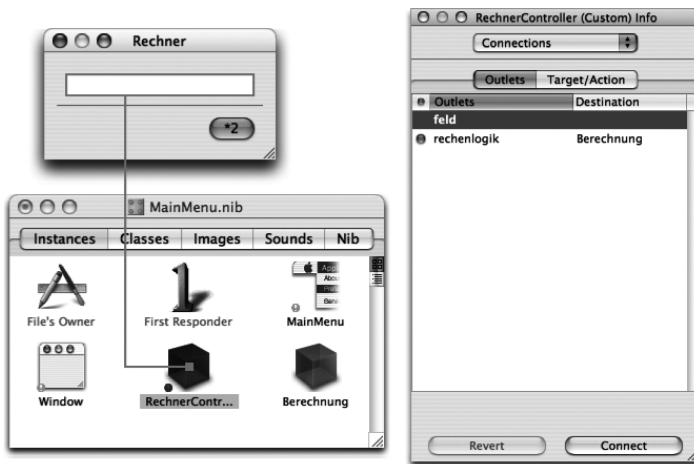


Abbildung 6.11 Referenzen zwischen Objekten grafisch definieren

Wählen Sie im Info-Fenster unter »Connections« das Outlet `feld` aus und klicken Sie auf »Connect«. Damit kennt Ihr Steuerungsobjekt eine Referenz auf das Ein-/Ausgabefeld. Bei der Abbildung wurde vorher bereits die Verbindung

zwischen dem Controller und der Klasse `Berechnung` hergestellt. Ziehen Sie dazu bei gedrückter `Ctrl`-Taste eine Verbindung vom Controller-Symbol zum `Berechnung`-Symbol und wählen Sie das Outlet `rechenlogik` aus.

Sie haben alle Outlets verknüpft, nun muss noch die Action `klick()` verbunden werden. Dazu stellen Sie eine Verbindung vom »*2«-Knopf zum Controller her. Wählen Sie dann im Info-Fenster im Bereich »Target/Action« die Methode `klick()` aus und bestätigen Sie die Verbindung mit »Connect«. Der Button wird jetzt jedes Mal, wenn er angeklickt wird, diese Methode aufrufen. Das Steuerungsobjekt ist damit vollständig verknüpft, speichern Sie die NIB-Datei.

Achten Sie darauf, dass Sie auf GUI-Elemente, die auf Java-Code zurückgreifen (z.B. eine »CustomView«), immer eine explizite Referenz halten, beispielsweise durch ein Outlet. Ansonsten kann es zu Abstürzen kommen, wenn die Garbage Collection Java-Objekte löscht, die eigentlich noch zum Zeichnen der CustomView benötigt werden.

Was jetzt noch fehlt, ist die Implementierung der Klassen in Xcode. Dazu lassen Sie sich zunächst ein passendes Quelltext-Gerüst vom Interface Builder generieren. Wählen Sie auf der »Classes«-Dialogseite die Klasse `RechnerController` aus und rufen Sie den Menüpunkt **Classes · Create Files for RechnerController** auf. Das Gleiche machen Sie für die Klasse `Berechnung`. Die beiden Quelltexte werden in Xcode irgendwo in der Liste »Groups & Files« einsortiert, aber Sie können sie natürlich in die Gruppe »Classes« ziehen. Den Interface Builder können Sie nun beenden. Der Quelltext für `RechnerController.java` wird vom InterfaceBuilder wie folgt generiert:

```
import com.apple.cocoa.foundation.*;
import com.apple.cocoa.application.*;
public class RechnerController extends NSObject {
    public NSTextField feld; /* IBOutlet */
    public Berechnung rechenlogik; /* IBOutlet */

    public void klick(Object sender) { /* IBAction */
    }
}
```

Listing 6.7 Generierter Controller-Quelltext

Am Anfang werden die Cocoa Java-Klassen importiert. Die Cocoa-API ist in zwei Bereiche aufgeteilt, in »Foundation« (allgemeine Routinen und Datenstrukturen) und in das »Applikation Kit« (kurz »AppKit«) zur Anwendungs- und

Oberflächenverwaltung. Die Klasse `RechnerController` erbt von der Klasse `NSObject`, wie sie dies im Interface Builder festgelegt haben.

Die Outlets sind als öffentliche Objektvariablen realisiert. Sie werden aber nirgendwo im Java-Programmcode das Erzeugen der entsprechenden Objekte finden – dies passiert automatisch, wenn das Programm gestartet und die NIB-Datei geladen wird! Die Informationen, welches Outlet mit welchem Objekt verbunden ist, stecken ja in der NIB-Datei. Lassen Sie die generierten Kommentare daher am besten stehen, damit Sie wissen, welche Referenzen aus den NIB-Dateien gefüllt werden

Das Laden der NIB-Datei geschieht indirekt in der `main()`-Funktion in der Datei `main.m`. Dort wird die Funktion `UIApplicationMain()` aufgerufen, die alle relevanten Cocoa-Aufrufe zum Programmstart durchführt und dann die Schleife zur Ereignisverarbeitung startet.

Die Methode `klick()`, die Sie als »Action« des Controllers definiert hatten, bekommt (wie alle Cocoa-Actions) nur einen einzigen Parameter: `sender` vom Typ `Object`. Bei Bedarf können Sie mit dieser Referenz auf das aufrufende Objekt zugreifen, um weitere Daten abzufragen. In diesem Beispiel wissen Sie durch die Verknüpfungen, die Sie im Interface Builder hergestellt haben, dass der Sender hier immer vom Typ `NSButton` ist. Erweitern Sie die Methode wie folgt:

```
public void klick(Object sender) { /* IBAktion */
    double eingabe = feld.doubleValue();
    double ausgabe = rechenlogik.malZwei(eingabe);
    feld.setDoubleValue(ausgabe);
    feld.selectText(this);
}
```

Zunächst lesen Sie mit `doubleValue()` den eingegebenen Wert ein, den Sie danach mithilfe der Methode `malZwei()` aus der Klasse `Berechnung` verdoppeln. Schließlich geben Sie den neu berechneten Wert wieder im selben Feld aus. Die letzte Zeile macht das Feld sofort wieder zum aktiven Feld und selektiert den darin enthaltenen Text (damit er bequem kopiert und anderswo eingefügt werden kann). Eine zusätzliche Methode sollten Sie noch implementieren:

```
public void awakeFromNib() {
    feld.window().makeKeyAndOrderFront(this);
    feld.selectText(this);
}
```

Diese Methode wird beim Programmstart aufgerufen, unmittelbar nachdem die NIB-Datei geladen wurde und alle Verbindungen der Outlets aufgebaut wurden (d.h., nachdem die Objekte erzeugt und die Referenzen gespeichert wurden). Hier wird einfach das Fenster, in dem sich das Textfeld befindet, als das Fenster definiert, das die Tastatureingaben bekommt – und es wird nach vorne geholt, d.h. zum aktiven Fenster gemacht. Danach wird der Eingabefokus auf das Textfeld gesetzt.

Die Klasse `Berechnung` wurde als leere Klasse generiert. Als Geschäftslogik implementieren Sie darin nur die Methode `malZwei()`:

```
public class Berechnung {
    public double malZwei(double wert) {
        return wert * 2.0;
    }
}
```

Das war es auch schon! Mit erstaunlich wenig Java-Code haben Sie eine interaktive Anwendung mit grafischer Oberfläche programmiert. Apple bietet eine ähnliche kleine Anleitung unter der Adresse <http://developer.apple.com/documentation/Cocoa/Conceptual/JavaTutorial/> an.



Abbildung 6.12 Target-Einstellungen für Cocoa Java-Anwendungen

Xcode erzeugt ein Mac OS X-Programmpaket, das – verglichen mit »normalen« Java-Programmen – ein paar Besonderheiten aufweist, vor allem bei den Target-Einstellungen (siehe Abbildung 6.12). Bei »Pure Java Specific« ist nichts ein-

getragen – kein Klassenpfad, keine Hauptklasse. Die entsprechenden Einträge finden Sie dafür bei »Cocoa Java Specific«, und bei »Cocoa Specific« ist die NIB-Datei angegeben. Die Einträge von »Cocoa Java Specific« finden Sie dann auch in der Datei `Info.plist` im Programmpaket wieder:

```
<key>NSJavaNeeded</key>
<string>YES</string>
<key>NSJavaPath</key>
<array>
    <string>CocoaJavaRechner.jar</string>
</array>
<key>NSJavaRoot</key>
<string>Contents/Resources/Java</string>
```

Listing 6.8 Ausschnitt aus `Info.plist` mit Cocoa Java-Schlüsseln

Wenn Sie sich intensiver mit Cocoa Java beschäftigen wollen, finden Sie eine Referenz der Cocoa-API auf den Seiten <http://developer.apple.com/documentation/Cocoa/Reference/ApplicationKit/Java/> und <http://developer.apple.com/documentation/Cocoa/Reference/Foundation/Java/>. Die wichtigsten Unterschiede der Cocoa-Programmierung mit Java bzw. Objective-C beschreibt Apple im Dokument <http://developer.apple.com/documentation/Cocoa/Conceptual/CocoaObjects/Articles/JavaCocoa.html>.

Die Cocoa Java Bridge kann auch ohne Interface Builder und NIB-Dateien direkt aus normalen Java-Anwendungen benutzt werden. Sie binden die Klassen in den Unterverzeichnissen von `/System/Library/Java/com/apple/cocoa/` ein, indem Sie das Verzeichnis `/System/Library/Java` auf den Klassenpfad setzen – wie im AppleScript-Beispiel im folgenden Abschnitt geschehen. Apple rät aber dringend davon ab, Cocoa-Komponenten mit Swing- oder AWT-Komponenten zu mischen, auch wenn dies zu funktionieren scheint. Dadurch kommt die Verwendung der Cocoa Java-Bridge eigentlich nur noch für reine Cocoa Java-Anwendungen in Frage.

Der empfohlene Weg, Cocoa-Funktionalität aus normalen Java-Applikationen heraus zu nutzen, ist JNI. Dazu leiten Sie eine Unterklasse von `com.apple.eawt.CocoaComponent` ab, die dann mit JNI über Objective-C den eigentlichen Cocoa-Aufruf durchführt. Ein gelungenes Beispiel dafür ist das »WebKit«, das die Cocoa-Web-Browser-Komponente als AWT-Klasse zur Verfügung stellt. Die Software kann von <http://www.concord.org/~dima/jws/webkit/> heruntergeladen werden.

Wenn Sie Gefallen am Oberflächenentwurf mit dem Interface Builder gefunden haben, sollten Sie sich »nib4j« ansehen. Mit dieser Java-Bibliothek können Sie Oberflächen mit dem Interface Builder entwerfen und als NIB-Datei speichern, die dann von Swing-Applikationen genutzt werden. Derzeit laufen solche Anwendungen aber nur unter Mac OS X. Herunterladen können Sie die für nicht kommerzielle Projekte kostenfreie Bibliothek von <http://www.nib4j.com/>.

6.5 AppleScript

AppleScript ist eine einfache, aber mächtige Skriptsprache zur Automatisierung von Applikationen. Sie stammt bereits vom alten Mac OS (Classic), wurde auf Mac OS X übernommen und erweitert. Mit umgangssprachlichen (englischen) Formulierungen können Sie beispielsweise iTunes fernsteuern.

Von zentraler Bedeutung ist dabei der Skripteditor im Verzeichnis `/Programme/AppleScript/`. Wenn Sie wissen möchten, mit welchen Aktionen ein Programm fernsteuerbar ist, ziehen Sie einfach die Anwendung auf das Programmsymbol des Skripteditors. Und natürlich können Sie dort auch Skripte eingeben und ausführen (siehe Abbildung 6.13).



Abbildung 6.13 AppleScript Skripteditor

Um AppleScript aus Java heraus aufzurufen, haben Sie im Wesentlichen zwei Möglichkeiten: die Cocoa Java-Bridge und das Kommandozeilenprogramm `osascript`.

6.5.1 AppleScript mit Cocoa Java

Ab Mac OS X 10.2 gibt es die Cocoa Java-Klasse `NSAppleScript`, die wie folgt verwendet wird:

```
//CD/examples/ch06/applescript/iTunesPlayCocoa/iTunesPlayCocoa.java
import com.apple.cocoa.foundation.*;
```

Zunächst werden die grundlegenden Cocoa Java-Klassen eingebunden, wie Sie dies gerade im vorangegangenen Abschnitt 0 gesehen haben.


```
static String QUELLTEXT = "tell application \"iTunes\" to play";
```

Dann wird der Quelltext des Skripts statisch festgelegt. Bei einem Skript, das sich über mehrere Zeilen erstreckt, fügen Sie einfach `\n` an den passenden Stellen in die Zeichenkette ein. Ohne dass Sie mehr über AppleScript wissen, ist das Skript gut verständlich – iTunes soll die Wiedergabe starten.

```
NSAppleScript skript = new NSAppleScript( QUELLTEXT );
NSMutableDictionary fehler = new NSMutableDictionary();
NSAppleEventDescriptor erg = skript.execute( fehler );
```

Nun wird ein `NSAppleScript`-Objekt mit dem Skript-Quelltext erzeugt. Das folgende Cocoa-Dictionary-Objekt wird benötigt, um eventuelle Fehlermeldungen bei der Ausführung zu speichern. `execute()` kompiliert das Skript dann und führt es aus – so einfach! Als Ergebnis erhalten Sie einen `AppleEvent-Deskriptor` (auch wenn es bei diesem Beispiel kein bzw. ein leeres Ergebnis gibt).

```
if (erg == null) {
    Enumeration e = fehler.keyEnumerator();
    while ( e.hasMoreElements() ) {
        Object key = e.nextElement();
        Object obj = fehler.objectForKey( key );
        System.out.println( key + ":\n\t" + obj );
    }
}
```

Wenn Sie als Ergebnis die `null`-Referenz erhalten, ist ein Fehler aufgetreten. Aus dem Dictionary-Objekt können Sie dann eine ganz normale Java-Enumeration für die Fehler-Schlüssel abfragen. Mit diesen Schlüsseln und der Dictionary-Methode `objectForKey()` ermitteln Sie dann die Fehler-Werte. Sie können dies testen, indem Sie beispielsweise in der Skript-Zeichenkette das zweite »l« von »tell« löschen.

```
else {
    for (int i = 0; i < erg.numberOfItems(); i++) {
        NSAppleEventDescriptor ergI = erg.descriptorAtIndex(i);
        System.out.println( ergI.stringValue() );
    }
}
```

Hat dagegen alles geklappt, fragen Sie vom `AppleEvent-Deskriptor` die Anzahl der Ergebnisse ab, holen sich dann mit `descriptorAtIndex()` für jedes einzelne Ergebnis einen Unter-Deskriptor und geben dessen Wert mit

`stringValue()` aus. Natürlich ist eine solche Auswertung nur nötig, falls Sie überhaupt Ergebnisse erwarten (bei diesem Beispiel also eigentlich nicht). Wenn das Skript nur ein einziges Ergebnis zurückliefert, können Sie `stringValue()` auch direkt auf dem Haupt-Deskriptor `erg` aufrufen.

Damit sind Sie fertig – zum Ausführen dürfen Sie allerdings nicht vergessen, das Verzeichnis `/System/Library/Java` auf den Klassenpfad zu setzen. Im Xcode-Beispielprojekt ist dies schon erledigt: Sie finden den Pfad in den Target-Einstellungen unter **Info.plist Entries · Simple View · Pure Java Specific · Classpath**. Und denken Sie daran, dass Apple die Verwendung von Cocoa Java nur allein, d. h. nicht gemischt mit Swing- und AWT-Komponenten, empfiehlt.

6.5.2 AppleScript mit Shell-Kommandos

Eine Alternative zur Einbindung von AppleScript in Java-Anwendungen ist `Runtime.exec()` und der Befehl `osascript`, der fester Bestandteil von MacOS X ist. Diese Lösung kann auch problemlos in Swing- und AWT-Applikationen eingesetzt werden. `osascript` dient zum Aufruf von OSA-kompatiblen Skriptsprachen (»Open Scripting Architecture«). Unter anderem gibt es im Internet auch ein passendes JavaScript-Modul, vor allem wird OSA aber für AppleScript genutzt. Mit `osalang -L` können Sie sich alle installierten OSA-Sprachen anzeigen lassen.

Der Aufruf von `osascript` ist einfacher als die Verwendung der Cocoa Java-Klasse, dafür ist die Skriptausführung nicht so detailliert konfigurier- und auswertbar. Meistens ist die hier vorgestellte Funktionalität jedoch vollkommen ausreichend:

```
//CD/examples/ch06/applescript/iTunesPlayExec/iTunesPlayExec.java
static String SKRIPT =
"tell application \"iTunes\" to get the name of the current track";
```

Von iTunes soll der Name des aktuellen (bzw. des zuletzt gespielten) Musikstücks abgefragt werden – das Skript hat also eine Rückgabe, die als Zeichenkette im Java-Programm ankommt.

```
Process p = Runtime.getRuntime().exec(
    new String[] { "osascript", "-e", SKRIPT }
);
```

An die `exec()`-Methode wird ein temporäres `String`-Array übergeben, das als Erstes den Befehl `osascript` enthält. Die nachfolgende `osascript`-Option `-e` bedeutet, dass die folgende Zeichenkette der Skript-Quelltext ist (und nicht

etwa ein Dateiname). Als Rückgabe erhalten Sie ein `Process`-Objekt, mit dem nun der Track-Name gelesen werden kann.

```
BufferedReader in = new BufferedReader(
    new InputStreamReader(
        p.getInputStream() ));
String trackName = in.readLine();
System.out.println( trackName );
in.close();
```

Über das `Process`-Objekt wird `getInputStream()` aufgerufen und damit ganz normal eine Zeichenkette eingelesen. Ist in iTunes kein Musikstück ausgewählt, erhalten Sie wie bei `InputStreams` üblich sofort die `null`-Referenz.

6.6 Speech & Spelling Frameworks

Das `Speech` und das `Spelling` Framework sind Systembibliotheken für Spracherkennung und -ausgabe sowie für eine Rechtschreibprüfung. Für Java sind diese Bibliotheken besonders interessant, weil Apple spezielle Hüllklassen für sie entwickelt hat, mit denen sich die Funktionen ganz normal aus Java heraus ansprechen lassen.

Apple tendiert allerdings dazu, solche Mac-spezifischen Java-APIs in Zukunft nicht mehr zu unterstützen; entsprechend erfahren die Hüllklassen seit längerer Zeit schon keine Weiterentwicklung mehr. Sie sollten diese Technologien also nicht mehr für neue Projekte einsetzen! Wenn Sie zukunftssichere Alternativen benötigen, können Sie `Cocoa Java` oder teilweise auch `AppleScript` verwenden.

6.6.1 Speech Framework

Das `Java Speech Framework` kann nur mit `Java 1.3` genutzt werden. Dass Apple diese Technologie für neuere `Java`-Versionen nicht mehr unterstützt, sehen Sie auch daran, dass Sie das Entwicklerpaket bei Apple nicht mehr herunterladen können. Falls Sie es doch noch anderswo im Netz finden, müssen Sie das Archiv `JavaSpeechFramework.jar` irgendwo auf den Klassenpfad packen, beispielsweise in eines der Erweiterungsverzeichnisse. Als kurzes Anwendungsbeispiel ist im Folgenden die Sprachausgabe (TTS, »Text-to-speech«) skizziert:

```
import com.apple.speech.synthesis.Synthesizer;
// ...
Synthesizer synth = new Synthesizer();
```

```

synth.speakText( "Hello World" );
//...
synth.stopSpeech();

```

Zunächst wird ein `Synthesizer`-Objekt erzeugt. Da kein Parameter übergeben wird, verwendet die Sprachausgabe die Standardstimme aus den Systemeinstellungen (**Systemeinstellungen** · **Sprache** · **Standardstimme**, siehe Abbildung 6.14). Leider gibt es derzeit nur englische Stimmen, und die Betonung deutscher Texte klingt damit nicht wirklich gut. Mit `speakText()` wird dann die Sprachausgabe gestartet, die jederzeit mit `stopSpeech()` wieder abgebrochen werden kann.



Abbildung 6.14 Systemeinstellungen für Spracherkennung und -ausgabe

Eine Spracherkennung können Sie mit den Klassen aus dem Paket `com.apple.speech.recognition` durchführen. Wenn Sie Mac OS X 10.3 einsetzen, erreichen Sie diese Funktionen auch über die Cocoa Java-Klassen `NSSpeechSynthesizer` und `NSSpeechRecognizer`, bei älteren Systemen musste man noch auf Carbon-Aufrufe zurückgreifen. Einen Überblick über Apples Sprachtechnologie finden Sie auf der Seite <http://developer.apple.com/documentation/Cocoa/Conceptual/Speech/>.

Eine ganz andere Möglichkeit zur Sprachausgabe ist der »Humble Narrator«, den Sie von <http://www.amug.org/~glguerin/sw/narrator/about-narrator.html> herunterladen können. Diese Java-Bibliothek steuert intern die AppleScript-Sprachausgabe über `osascript` an.

6.6.2 Spelling Framework

Das Spelling Framework ist eine Bibliothek zur Rechtschreibprüfung und ist seit den ersten MacOS X-Versionen vorhanden. Das Archiv für die Java-Hüllklassen ist auf Apples Entwicklerseiten nicht mehr direkt verlinkt, Sie können es aber von ftp://ftp.apple.com/developer/Sample_Code/Java/JavaSpellingFramework.sit herunterladen. Das Framework funktioniert weitestgehend auch noch mit Java 1.4, wurde aber seit langem nicht mehr gepflegt – der Quelltext enthält teilweise noch Fehler. Und da die Bibliothek auf systeminterne Komponenten zurückgreift, die nirgendwo dokumentiert sind, ist es im Zweifel besser, wenn Sie – falls überhaupt – die entsprechende Cocoa Java-Klasse `NSSpellingChecker` verwenden, um eine längerfristige Kompatibilität sicherzustellen. Einen Überblick über die systemweite Rechtschreibkontrolle hat Apple auf der Seite <http://developer.apple.com/documentation/Cocoa/Conceptual/SpellCheck/> veröffentlicht.

Das von Apple bereitgestellte Java Spelling Framework ist ein Project Builder-Projekt (das Sie natürlich auch mit Xcode öffnen können). Wenn Sie die Software kompilieren, werden die Dateien `libspeller.jnilib` und `JavaSpellingFramework.jar` im `build`-Verzeichnis erzeugt, die Sie beide auch in eigenen Programmen verwenden dürfen (am besten legen Sie sie ins Verzeichnis `Contents/Resources/Java` innerhalb des jeweiligen Programmpakets). Die Warnungen beim Build-Vorgang können Sie ignorieren.

Es gibt zwei Arten, das Spelling Framework einzusetzen: Sie können die Klassen direkt an eine Swing-Oberfläche ankoppeln, oder Sie steuern oberflächenunabhängige Klassen mit den passenden Methodenaufrufen an. Zunächst sehen Sie die Oberflächenintegration, die darauf beruht, dass ein `JTextComponent`-Element von einem `JtxtCmpontDrvr`-Objekt des Frameworks bearbeitet wird:

```
import com.apple.spell.ui.JTxtCmpontDrvr;
// ...
    JTextArea editor = new JTextArea();
    editor.setText("Gahnz fiele valsche Wörtär!");
```

Hier wird die Oberfläche zusammengebaut. Der abgedruckte Quelltext ist nicht vollständig, aber der wesentliche Punkt ist zu sehen: Eine Unterklasse von `JTextComponent` – `JTextArea` – enthält den Text, der korrigiert werden soll.

```
JTxtCmpontDrvr spellchecker = new JTxtCmpontDrvr();
```

Jetzt wird ein Treiber-Objekt angelegt, mit dem Sie nun drei alternative Möglichkeiten haben, den Text zu prüfen:

```
spellchecker.checkSpelling( editor );
```

Bei dieser ersten Möglichkeit wird mit `checkSpelling()` die übergebene Textkomponente interaktiv überprüft. Das Spelling Framework öffnet dazu einen Dialog, den Sie in Abbildung 6.15 sehen.

```
spellchecker.startRealtimeChecking( editor );  
// ...  
spellchecker.stopRealtimeChecking( editor );
```

Diese zweite Möglichkeit erlaubt es Ihnen, die Wörter während der Eingabe überprüfen zu lassen. Dazu können Sie die Kontrolle mit `startRealtimeChecking()` ein- und mit `stopRealtimeChecking()` wieder ausschalten.

```
spellchecker.markMisspelledWords( editor, 0, -1 );
```

Schließlich können Sie mit dem Aufruf von `markMisspelledWords()` alle falsch geschriebenen Wörter in der Textkomponente markieren (rot unterpunkten) lassen. Hier wird der komplette Text geprüft, vom Anfang (0) bis zum Ende (-1).

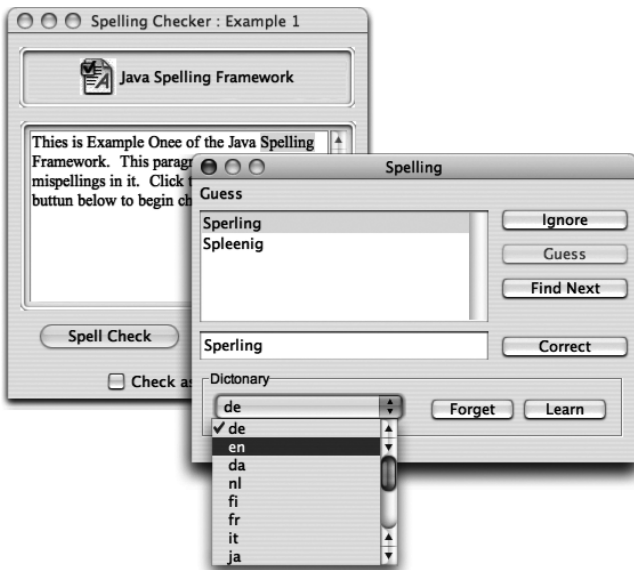


Abbildung 6.15 Interaktive Rechtschreibprüfung

Die zweite Art, mit der Sie das Spelling Framework ansteuern können, besteht aus den statischen Methoden der Klasse `com.apple.spell.SpellingChecker`. Unter anderem stehen Ihnen dort folgende Methoden zur Verfügung:

- ▶ `MisspelledWord findFirstMisspelledWord`
`(String in, int startingAt, String language)`
`MisspelledWord findMisspelledWordInString`
`(String in, String language)`

Diese Methoden suchen falsch geschriebene Wörter in der übergebenen Zeichenkette. Mit dem zurückgegebenen Objekt `com.apple.spell.MispelledWord` können Sie alternative Wortvorschläge abfragen und das Wort zum gelernten Wortschatz hinzufügen. Die Sprache (z.B. »de« oder »en«) muss immer angegeben werden.

- ▶ `String[] suggestGuessesForWord(String word, String language)`
 Mit dieser Methode können Sie eine Liste von Verbesserungsvorschlägen abfragen, siehe `MisspelledWord.getSuggestedCorrections()`.
- ▶ `boolean learnWord(String word, String language)`
 Fügt ein Wort zum erkannten Wortschatz hinzu, siehe `MisspelledWord.learnWord()`.

6.7 Weitere systemabhängige Bibliotheken

Von Drittanbietern gibt es eine Vielzahl weiterer systemabhängiger Java-Bibliotheken. Zwei davon, die Ihnen Posix-Aufrufe und eine Benutzer-Autorisation ermöglichen, werden im Folgenden kurz vorgestellt.

6.7.1 Posix-Aufrufe mit dem Suite/P Toolkit

Mit dem »Suite/P Toolkit« können Sie bestimmte Standard-Posix-Routinen aufrufen, unter anderem solche zur Prozess-, Benutzer-, Gruppen- und Dateiverwaltung. Die Software kann mit dem kompletten Sourcecode kostenlos von <http://www.amug.org/~glguerin/sw/suitep/> heruntergeladen werden. Verwendet wird die Bibliothek dann nach folgendem Schema:

```
import glguerin.suitep.*;
import glguerin.suitep.posix.*;
// ...

Suite suite = Suite.getSuite();
Processing proc =
    (Processing)suite.get( PosixSuites.PROCESS_PROCESSING );
int pgrp = proc.getProcessGroup();
```

```

int pid =
    suite.getNumber( PosixSuites.PROCESS_ID ).intValue();
int ppid =
    suite.getNumber( PosixSuites.PROCESS_ID_PARENT ).intValue();
int pgid =
    suite.getNumber( PosixSuites.PROCESS_GROUP ).intValue();

```

Zunächst wird die Suite initialisiert. Dann können über das `Suite`-Objekt verschiedene weitere Objekte (hier beispielsweise ein `Processing`-Objekt) und Werte abgefragt (bzw. gesetzt) werden.

Intern setzt Suite/P eine JNI-Bibliothek ein. Die Software ist sehr flexibel und erweiterbar – es spricht beispielsweise nichts dagegen, dass diese Bibliothek auch für andere Betriebssysteme implementiert wird. Derzeit ist sie aber nur für Mac OS X verfügbar.

6.7.2 Authorisation Toolkit

Das »Authorisation Toolkit« dient zur Autorisation eines Java-Programms durch den Anwender, damit das Programm mit Administrator- oder Superuser-Rechten laufen kann. Dabei muss der Benutzer keine Sicherheitsrichtlinien-Datei bearbeiten, sondern es wird ihm der normale, fest ins System integrierte Identifikations-Dialog präsentiert (siehe Abbildung 6.16). Der Mechanismus ist also kein Ersatz für das Java-Sicherheitsmodell, sondern er ergänzt es für die Fälle, in denen eine Java-Anwendung Zugriff auf geschützte Systemverzeichnisse und -kommandos benötigt.



Abbildung 6.16 Benutzer-Identifikation zum Zugriff auf Systemkomponenten

Die Programmierung erfolgt ganz grob nach folgendem Schema:

```
import glguerin.authkit.*;
// ...
Class c =
    Class.forName("glguerin.authkit.imp.macosx.MacOSXAuthorization");
Authorization auth = (Authorization) c.newInstance();
auth.isAvailable( Privilege.EMPTY );
Process child =
    auth.execPrivileged( new String[] { "/usr/bin/id" } );
```

Zur Initialisierung der Bibliothek laden Sie mit `Class.forName()` zunächst die passende Implementierungsklasse und erzeugen ein Objekt davon. Entscheidend ist dann der Aufruf von `execPrivileged()`. Damit wird der MacOS X-Autorisationsdialog angezeigt und bei Erfolg das übergebene Kommando ausgeführt. Ebenso ist es möglich, das gerade laufende Programm zu autorisieren, um dann später geschützte Kommandos aufzurufen.

Sie können die Bibliothek inklusive komplettem Quelltext von <http://www.amug.org/~glguerin/sw/authkit/> herunterladen.

6.8 Literatur & Links

► JNI

- S. Liang, »The Java Native Interface«, Addison-Wesley 1999
Das Standardwerk von Sun zur JNI-Programmierung. Auch wenn es auf dem Stand von JNI 1.2 ist, gilt das Allermeiste auch für neuere Versionen.
- <http://java.sun.com/j2se/1.4.2/docs/guide/jni/>
Verweist auf eine JNI-Einführung, die JNI-Spezifikation und aktuelle JNI-Erweiterungen.

► JDirect

- Albert/Hart/Hopkins/Steinberg/Williams , »Early Adopter MacOS X Java«, Wrox Press 2001
Dieses Buch behandelt Java auf MacOS X 10.0 und 10.1 und ist damit zum Teil veraltet. Andererseits erhalten Sie dadurch eine ausführliche Beschreibung von JDirect. Und auch Cocoa Java ist hier gründlich erklärt.

► Cocoa

- A. Hillegass, »Cocoa Programming for MacOS X«, 2nd ed., Addison-Wesley 2004
Eine hervorragende Einführung in die Software-Entwicklung mit Cocoa, sowohl was die Oberflächenerzeugung als auch die zugrunde liegende Programmierung mit Objective-C betrifft.

► **AppleScript**

- ▶ M. Neuburg, »AppleScript – The Definitive Guide«, O'Reilly 2003
Zum Lernen von AppleScript ist diese sehr umfassende Einführung gut geeignet.
- ▶ B.W. Perry, »AppleScript in a Nutshell«, O'Reilly 2001
Kann prima als Nachschlagewerk eingesetzt werden.

► **Speech & Spelling Frameworks**

- ▶ W. Iverson, »Mac OS X for Java Geeks«, O'Reilly 2003
Dieses Buch behandelt die beiden Frameworks für Sprachausgabe und Rechtschreibprüfung mit ausführlichen Beispielen.

7 Grafik und Multimedia

7.1	Java Advanced Imaging (JAI) und Java Image I/O (JIIO)	353
7.2	Java 3D	356
7.3	OpenGL/JOGL	358
7.4	Java Media Framework (JMF)	362
7.5	QuickTime for Java (QTJava)	365
7.6	Sound/Musik	374
7.7	Drucken	375
7.8	Literatur & Links	380

- 1 **Grundlagen**
 - 2 **Entwicklungsumgebungen**
 - 3 **Grafische Benutzungsoberflächen (GUI)**
 - 4 **Ausführbare Programme**
 - 5 **Portable Programmierung**
 - 6 **Mac OS X-spezielle Programmierung**
-
- 7 **Grafik und Multimedia**
 - 8 **Werkzeuge**
 - 9 **Datenbanken und JDBC**
 - 10 **Servlets und JavaServer Pages (JSP)**
 - 11 **J2EE und Enterprise JavaBeans (EJB)**
 - 12 **J2ME und MIDP**
- A **Kurzeinführung in die Programmiersprache Java**
 - B **Java auf Mac OS 8/9/Classic**
 - C **Java 1.5 »Tiger«**
 - D **System-Properties**
 - E **VM-Optionen**
 - F **Xcode- und Project Builder-Einstellungen**
 - G **Mac OS X- und Java-Versionen**
 - H **Glossar**
 - I **Die Buch-CD**

7 Grafik und Multimedia

*»Ich tätowier« dir meinen Namen in allen Farben, in allen Farben/
Bemal mich, mach mich bunt/Bemal mich bunt!«
(Etwas)*

Die Grafikausgabe ist mit der Java-Implementation von Mac OS X im Allgemeinen kein Problem. Die einfachen `java.awt.Graphics`-Methoden werden ebenso unterstützt wie Java 2D mit seinem `java.awt.Graphics2D`-Kontext. Diese Grundlagen werden daher im Folgenden nicht mehr speziell behandelt – bei den Beispielsprogrammen der vorangegangenen Kapitel kamen sie teilweise schon zum Einsatz. Wenn Sie bei der Java 2D-Ausgabe optische Feineinstellungen vornehmen wollen, sollten Sie sich im Anhang den Abschnitt »Grafikdarstellung« in Kapitel 16, *System-Properties*, ansehen.

In diesem Kapitel erhalten Sie einen Überblick über spezielle Themen der Grafik- und Multimediaverarbeitung. Zunächst werden zwei Standard-Erweiterungspakete für Bildmanipulation und 3D-Grafik vorgestellt, letzteres wird dann mit der Java-Implementierung des 3D-Standards OpenGL verglichen. Auch für Multimediadaten wird zunächst Suns Java-Standard besprochen, anschließend dann Apples QuickTime-Lösung. Ganz zum Schluss folgen noch Hinweise zur Sound-Ausgabe und zu den Druckmöglichkeiten unter Java.

7.1 Java Advanced Imaging (JAI) und Java Image I/O (JIIO)

Die Java Advanced Imaging API stellt im Paket `javax.media.jai` hochperformante, plattformunabhängige Schnittstellen zur Bildbearbeitung zur Verfügung, darunter diverse Bildoperationen wie z.B. Filter. JAI ist für die Bildbearbeitung im Netzwerk optimiert, unterstützt viele Bildformate und die Bilddaten können problemlos mit der Java 2D-Ausgabe gemischt werden. Auf der Seite <http://java.sun.com/products/java-media/jai/> hat Sun eine Liste der Dokumentationen und diverser JAI-Anwendungen zusammengestellt.

Apple hat für Mac OS X 10.3 und Java 1.4 die aktuelle JAI-Version 1.1.2 implementiert, allerdings muss diese separat heruntergeladen und installiert werden. Das Installationspaket für JAI und Java 3D finden Sie auf der Seite <http://docs.info.apple.com/article.html?artnum=120289>.

Die vielleicht am häufigsten genutzten Klassen von JAI dienen zur schnellen Ein- und Ausgabe von Bilddateien (»Image I/O«) und gehören zum Paket

`com.sun.media.jai.codec`. Wie Sie am Namen unschwer erkennen, ist dies kein Standardpaket, daher sollten Sie den Einsatz bei portablen Programmen vermeiden. Zum Glück gibt es ab Java 1.4 aber portablen Ersatz, die Java Image I/O API im Paket `javax.imageio`. Diese Klassen sind fester Bestandteil vom J2SE 1.4 und werden wie folgt genutzt:

```
//CD/examples/ch07/imageio/ImageIOTest/ImageIOTest.java
import java.io.*;
import java.net.*;
import java.awt.image.*;
import javax.imageio.*;
//...
try {
    URL url = new File("galileo_logo.gif").toURI().toURL();
    BufferedImage img = ImageIO.read( url );
    if (!ImageIO.write( img, "jpeg", new File("test.jpg") )) {
        System.exit(1);
    }
}
catch (Exception e) {
    e.printStackTrace();
}
```

Sie rufen also die statischen Methoden `read()` und `write()` in der Klasse `javax.imageio.ImageIO` auf – einfacher geht es kaum. Bei Java 1.4 können Sie sich darauf verlassen, dass GIF, JPEG und PNG geladen und JPEG und PNG gespeichert werden können. Unterstützung für weitere Bildformate (z.B. BMP, JPEG2000 und TIFF) erhalten Sie mit den »Java Advanced Imaging Image I/O Tools« von der Seite <http://java.sun.com/products/java-media/jai/downloads/download-iio.html>. Laden Sie dort das Linux-Archiv herunter und packen Sie das enthaltene Archiv `jai_imageio.jar` auf den Klassenpfad, z.B. in das Verzeichnis `/Library/Java/Extensions/`. Leider hat Apple die nativen JIIO-Codex noch nicht implementiert, so dass derzeit nur die portablen, aber relativ langsamen Java-Routinen verwendet werden.

Für die Ein-/Ausgabe von Bilddaten mit älteren Java-Versionen können Sie JIMI (Java Image Management Interface) verwenden, das Sun auf der Seite <http://java.sun.com/products/jimi/> zum Download anbietet. JIMI unterstützt recht viele Formate, und vor allem können Sie damit auch PICT-Bilder verarbeiten. PICT ist ein relativ altes Apple-Bildformat, das sowohl Bitmap- als auch Vektordaten speichern kann und das auch von Mac OS X noch verwendet wird. Der Quelltext zum Einlesen einer PICT-Datei ist kurz und übersichtlich:

```

import java.io.*;
import com.sun.jimi.core.*;
//...
    InputStream in = new FileInputStream( "mein.pict" );
    java.awt.Image img =
        Jimi.getImage( in, "image/pict", Jimi.VIRTUAL_MEMORY );

```

Wenn Sie Bilder möglichst performant darstellen wollen, sollten Sie die Bilddaten nach dem Laden in ein Format umwandeln, das von der Grafikhardware ohne großen Aufwand verarbeitet werden kann. Dazu laden Sie das Bild am besten wie oben beschrieben mit `ImageIO.read()` und geben die geladene `BufferedImage`-Referenz `img` dann in eine optimierte Bitmap aus. Die optimierten Bilddaten lassen Sie dann wie gewohnt innerhalb einer Komponente mit `Graphics.drawImage()` zeichnen:

```

import java.awt.*;
import java.awt.image.*;
//...
    BufferedImage img = javax.imageio.ImageIO.read( url );
    BufferedImage optimiert =
        GraphicsEnvironment.getLocalGraphicsEnvironment()
            .getDefaultScreenDevice().getDefaultConfiguration()
            .createCompatibleImage( img.getWidth(), img.getHeight() );
    optimiert.getGraphics().drawImage( img, 0,0, this );

```

Auch wenn der MIME-Typ für das PICT-Format mittlerweile `image/x-pict` lautet, sollten Sie bei JIMI noch die früher verwendete Zeichenkette angeben. Wenn Sie das Bild aus einer anderen Quelle als einer PICT-Datei einlesen, müssen Sie darauf achten, dass vor den eigentlichen Bilddaten ein 512 Bytes langer Header mit Nullbytes gesendet wird.

Leider besitzt JIMI beim Einlesen von PICTs einige Fehler, so dass PICT-Dateien mit der PackBits-Kompression nicht verwendet werden können. Auch beim Schreiben der Dateien enthält der Originalquelltext Fehler, für die es allerdings unter <http://www.amug.org/~glguerin/other/index.html#PICTWriter> eine Korrektur gibt. Eine andere Möglichkeit zum Bearbeiten von PICT-Bildern bietet QTJava, das weiter hinten in diesem Kapitel vorgestellt wird.

7.2 Java 3D

Java 3D ist ein Standard-Erweiterungspaket, das im Paket `javax.media.j3d` eine ausgefeilte objektorientierte Schnittstelle für die 3D-Grafik-Programmierung anbietet. Natürlich sollten alle Java-Bibliotheken mehr oder weniger objektorientiert sein, aber Sie werden im folgenden Abschnitt zu OpenGL sehen, dass es auch Grafikbibliotheken gibt, bei denen Sie sich deutlich mehr mit den Details beschäftigen müssen. Suns Portalseite für Java 3D ist <http://java.sun.com/products/java-media/3D/> – etwas versteckt befindet sich dort auch die Seite <http://java.sun.com/products/java-media/3D/collateral/> mit einer ausführlichen Anleitung und vielen Beispielen.

Java 3D wird von Apple zusammen mit JAI als Installationspaket angeboten, das Sie unter anderem von <http://www.apple.com/downloads/macosx/apple/java3dandjavaadvancedimagingupdate.html> herunterladen können. Wie bei JAI wird MacOS X 10.3 mit Java 1.4 unterstützt, die von Apple implementierte Java 3D-Version 1.3.1 ist die derzeit aktuelle.

In der Praxis wird häufig OpenGL als programmiersprachenübergreifender Standard eingesetzt. Dass man aber auch gut mit Java 3D entwickeln kann, zeigt folgendes Programm, das an ein Beispiel aus der oben erwähnten Java 3D-Anleitung angelehnt ist:

```
//CD/examples/ch07/java3d/Java3Dtest/Java3Dtest.java
import java.awt.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.SimpleUniverse;
```

Die Klassen aus den Unterpaketen von `com.sun.j3d` gehören zwar nicht direkt zum Java 3D-Standard, aber sie werden auch bei der MacOS X-Implementation mitgeliefert. Sie fassen häufige Anwendungsfälle zusammen und erleichtern die Programmierung mitunter deutlich.

```
public class Java3DTest extends Frame {
    public Java3DTest() {
        GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas = new Canvas3D( config );
        this.add( canvas, BorderLayout.CENTER );
        BranchGroup scene = this.createSceneGraph();
        scene.compile();
        SimpleUniverse universe = new SimpleUniverse( canvas );
```



```

        universe.getViewingPlatform().setNominalViewingTransform();
        universe.addBranchGraph( scene );
    }

```

Die Hilfsklasse `SimpleUniverse` erzeugt zunächst eine passende Standard-Grafikkonfiguration, damit Sie sich um die Details nicht kümmern müssen. Damit wird dann ein `Canvas3D`-Objekt erzeugt, eine Ausgabekomponente für Java 3D-Grafik. Als Nächstes wird ein so genannter Szene-Graph zusammengestellt, der die darzustellenden Elemente und deren Transformationen speichert – die Methode `createSceneGraph()` müssen Sie selbst implementieren, sie wird gleich noch ausführlicher vorgestellt. Die Szene wird dann noch kompiliert, damit die Anzeige einigermaßen schnell vonstatten geht. Am Ende wird dann ein `SimpleUniverse`-Objekt erzeugt, das die eigentlichen Ausgaben vornimmt und das dazu sowohl mit dem `Canvas3D`- als auch mit dem Szene-Objekt verknüpft wird. Außerdem wird der Beobachtungspunkt so im Raum verschoben, dass man einen möglichst guten Blick auf die Szene hat.

```

public BranchGroup createSceneGraph() {
    BranchGroup scene = new BranchGroup();
    Transform3D transformation = new Transform3D();
    transformation.rotX( Math.PI / 4.0 );
    TransformGroup rotation =
        new TransformGroup( transformation );
    rotation.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    Alpha zeit = new Alpha( -1, 5000 );
    RotationInterpolator animation =
        new RotationInterpolator( zeit, rotation );
    BoundingSphere bounds = new BoundingSphere();
    animation.setSchedulingBounds( bounds );
    rotation.addChild( new ColorCube( 0.4 ) );
    rotation.addChild( animation );
    scene.addChild( rotation );
    return scene;
}
//...
}

```

In dieser Methode wird der Szene-Graph zusammengebaut, der als Wurzel ein `BranchGroup`-Objekt besitzt. Die Szene setzt sich nicht nur aus den darzustellenden Elementen zusammen, sondern auch aus Transformationen und Animationen. Hier wird zunächst eine 3D-Transformation definiert, die das später zugeordnete Element um 45 Grad ($2\pi/8$) entgegen dem Uhrzeigersinn um die

X-Achse dreht. Dann wird eine zusätzliche `TransformationGroup` angelegt, die sicherstellt, dass die Werte nach der Transformation wieder in der Gruppe gespeichert werden dürfen (`ALLOW_TRANSFORM_WRITE`).

Für die Animation benötigen Sie eine `Alpha`-Zeitfunktion. Hier wird eine Funktion definiert, die in 5.000 Millisekunden vom Anfang bis zum Ende durchläuft und die dann endlos wieder von vorne startet (das legt die `-1` fest). Mit der Zeitfunktion und der Rotationstransformation erzeugen Sie nun ein `RotationInterpolator`-Animationsobjekt, dem Sie durch das `BoundingSphere`-Objekt sagen, innerhalb welchen Bereichs die Animation stattfindet (hier im Standardbereich; in einer Kugel mit Radius 1 um den Ursprung).

Endlich wird auch das darzustellende Element erzeugt – ein `ColorCube`-Objekt mit passender Skalierung, das automatisch unterschiedlich eingefärbte Seiten besitzt. Dieses Würfel-Objekt wird mit den Transformationen in umgekehrter Reihenfolge ineinander verschachtelt und schließlich zu der Szene hinzugefügt.

Wenn Sie das Programm nun übersetzen und ausführen, sollten Sie in etwa Abbildung 7.1 sehen. Das Bemerkenswerte dabei ist, dass die Animation nirgendwo explizit gestartet wird – als Bestandteil der Szene beginnt sie einfach automatisch, sobald die Szene angezeigt wird.

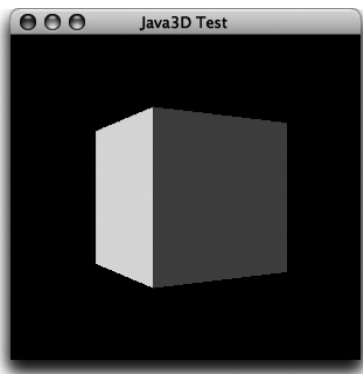


Abbildung 7.1 Drehender Würfel mit Java 3D

7.3 OpenGL/JOGL

Wenn Ihnen Java 3D schon recht mathematisch und aufwändig erschien, wird Ihnen die OpenGL-API erst recht etwas »roh« vorkommen – die Programmierschnittstellen sind sehr grundlegend, und es dauert deutlich länger als beispielsweise bei Java 3D, bis Sie eine komplexe Szene realisiert haben. Nichtsdestotrotz ist OpenGL seit Anfang der 1990er Jahre ein 2D- und 3D-Grafikstandard für viele Plattformen und Programmiersprachen, was vor allem

an der Geschwindigkeit, den guten Optimierungsmöglichkeiten und der vollen Kontrolle über viele Details der Darstellung liegt.

OpenGL ist fester Bestandteil von Mac OS X, Sie finden einige Hilfsprogramme (Profiler und ein Konstruktionsprogramm) im Order `/Developer/Applications/Graphics Tools/` auf Ihrer Festplatte. Auch aus Java heraus können Sie OpenGL nutzen, da diverse Anbindungen entwickelt wurden. Früher wurde vor allem GL4Java (<http://jausoft.com/gl4java/>) eingesetzt, mittlerweile hat sich »JOGL« als Quasi-Standard etabliert. Allgemeine Informationen zu JOGL finden Sie auf der Seite <https://jogl.dev.java.net/>, die Mac OS X-Portierung können Sie als Komplettpaket auch von <http://homepage.mac.com/gziemski/projects/> herunterladen.

Die heruntergeladenen Dateien installieren Sie am besten im Verzeichnis `/Library/Java/Extensions/`. Dort müssen sich mindestens die Bibliotheken `jogl.jar` und `libjogl.jnilib` befinden. Wenn Sie auch die drei Demo-JAR-Archive in dieses Verzeichnis kopieren, können Sie mit dem Befehl

```
java demos.vertexProgRefract.VertexProgRefract
```

ein schönes Beispiel für die Möglichkeiten von OpenGL starten. Die Beschreibungen aller verfügbaren Demoprogramme finden Sie auf der Seite <https://jogl-demos.dev.java.net/>. Aber wie können Sie nun selber eine kleine JOGL-Anwendung schreiben?

```
//CD/examples/ch07/jogl/JoglTest/JoglTest.java
import java.awt.*;
import net.java.games.jogl.*;
public class JoglTest extends Frame implements GLEventListener {
    public JoglTest() {
        super ("JOGL Test");
        GLCapabilities capabilities = new GLCapabilities();
        GLCanvas canvas =
            GLDrawableFactory.getFactory().createGLCanvas(capabilities);
        canvas.addGLEventListener(this);
        this.add( canvas, BorderLayout.CENTER );
    }
}
```

Das JOGL-Paket `net.java.games.jogl` deutet schon darauf hin, dass OpenGL häufig für Spiele verwendet wird, aber auch in Bereichen wie CAD, Simulation und Visualisierung kommt es zum Einsatz. Im Konstruktor erzeugen Sie zunächst ein `GLCapabilities`-Objekt, das die Standardvorgaben für die Grafikkonfiguration liefert. Damit lassen Sie dann ein `GLCanvas`-Objekt generieren, in dem die eigentliche Ausgabe der Grafikelemente stattfindet. Wenn

ein `GLCanvas` angezeigt wird, meldet er die vier Ereignisse `init()`, `display()`, `displayChanged()` und `reshaped()` an einen registrierten Listener, daher wird das entsprechende Interface `GLEventListener` von der Fensterklasse implementiert.

```
public void init(GLDrawable drawable) {
    GL gl = drawable.getGL();
    gl.glClearColor( 1.0F, 1.0F, 1.0F, 1.0F );
    gl.glColor3f( 0.0F, 0.0F, 1.0F );
}
```

Zur Initialisierung setzen Sie die Hintergrundfarbe mit `glClearColor()` auf Weiß (alle Farbkanäle sowie der Alpha-Kanal werden mit `1.0F` auf volle Intensität gebracht) und die Zeichenfarbe mit `glColor3f()` auf Blau. Die Methoden rufen Sie auf dem GL-Kontext auf – dies ist die grundlegende Schnittstelle zu OpenGL mit einer schier unglaublichen Anzahl von Grafikroutinen.

```
public void display(GLDrawable drawable) {
    GL gl = drawable.getGL();
    gl.glClear( GL.GL_COLOR_BUFFER_BIT );
    gl.glBegin( GL.GL_LINE_LOOP );
    gl.glVertex2i( 10, 20 );
    gl.glVertex2i( 80, 10 );
    gl.glVertex2i( 90, 80 );
    gl.glVertex2i( 20, 90 );
    gl.glEnd();
}
```

Wenn JOGL die `display()`-Methode aufruft, müssen Sie die Grafikelemente darstellen. Sie könnten die Ausgabe puffern, hier wird aber einfach bei jedem Aufruf die komplette Grafik neu gezeichnet. Zwischen `glBegin()` und `glEnd()` geben Sie dazu alle nötigen Informationen für das Grafikelement an – in diesem Fall werden Punkte für einen geschlossenen Linienzug (`GL_LINE_LOOP`) festgelegt. Diese Methode erinnert an die `paint()`-Methode und den `Graphics`-Kontext von AWT-Komponenten.

```
public void reshape(GLDrawable drawable,
                    int x, int y, int width, int height) {
    GL gl = drawable.getGL();
    GLU glu = drawable.getGLU();
    gl.glViewport( 0, 0, width, height );
    gl.glMatrixMode( GL.GL_PROJECTION );
```

```

    gl.glLoadIdentity();
    glu.gluOrtho2D( 0.0, 100.0, 0.0, 100.0 );
}

```

Wenn sich die Größe des GLCanvas-Objektes verändert, weil beispielsweise das Fenster breiter oder schmaler gemacht wurde, ruft JOGL die Methode `reshape()` auf. `glViewport()` setzt den Ausgabebereich auf die neue Breite und Höhe. Die Methode `gluOrtho2D()`, die zu den OpenGL-Hilfsroutinen gehört, legt das 2D-Koordinatensystem auf jeder Achse auf den Bereich 0 bis 100 fest.

```

public void displayChanged(GLDrawable drawable,
                           boolean modeChanged,
                           boolean deviceChanged) { }

```

Mit `displayChanged()` kann JOGL signalisieren, dass sich die Konfiguration des Ausgabegeräts verändert hat, beispielsweise die Farbtiefe oder die Bildschirmauflösung. Allerdings ist diese Funktionalität derzeit nicht in JOGL implementiert.

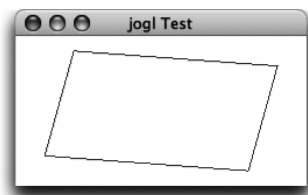


Abbildung 7.2 2D-Ausgabe mit OpenGL bzw. JOGL

Wenn Sie das Programm ausführen, sehen Sie ein Viereck, das sich durch die in `reshape()` definierten Matrix-Transformationen automatisch an die Größe des Fensters anpasst (siehe Abbildung 7.2). Alles in allem haben Sie – verglichen mit Java 3D – bei JOGL mehr Aufwand für weniger »Effekte« getrieben, aber gerade wenn Sie Spiele oder andere zeitkritische grafische Anwendungen schreiben wollen, lohnt sich der Aufwand meistens. Weitere JOGL-Einführungen finden Sie zahlreich im Internet, unter anderem auf den Seiten <http://today.java.net/pub/a/today/2003/09/11/jogl2d.html>, <http://today.java.net/pub/a/today/2004/03/18/jogl-2d-animation.html> und <http://www.genedavissoftware.com/books/jogl/>.

7.4 Java Media Framework (JMF)

Mit dem Java Media Framework aus dem Standard-Erweiterungspaket `javax.media` können Sie Audio-, Video- und andere zeitbasierte Mediendaten in Ihre Java-Applikationen integrieren. Die auf der Seite <http://java.sun.com/products/java-media/jmf/> beschriebenen Eigenschaften lesen sich gut, und die Liste der unterstützten Formate erscheint ausreichend lang. Leider jedoch können bei weitem nicht alle der verfügbaren AVI-, MPEG- und QuickTime-Dateien dekodiert werden, weshalb oft lieber das im folgenden Abschnitt vorgestellte QuickTime for Java verwendet wird. Dennoch bietet JMF einen unschlagbaren Vorteil: Es kann portabel auf jeder J2SE-Implementation eingesetzt werden.

Die aktuelle JMF-Version 2.1.1e können Sie von der Seite <http://java.sun.com/products/java-media/jmf/2.1.1/download.html> herunterladen. Für Mac OS X müssen Sie das »Cross-platform Java«-Archiv verwenden, es steht leider kein »Performance Pack« wie für andere Systeme zur Verfügung. Wenn Sie den ausgepackten Ordner in Ihr Benutzerverzeichnis legen, können Sie mit den Befehlen

```
setenv JMFHOME ~/JMF-2.1.1e
java -classpath $JMFHOME/lib/jmf.jar JMFRegistry
```

testen, ob die JMF-Installation auf Ihrem Rechner läuft. Es erscheint der »JMF Registry Editor« (siehe Abbildung 7.3), mit dem Sie unter anderem die installierten Format-Module verwalten können.

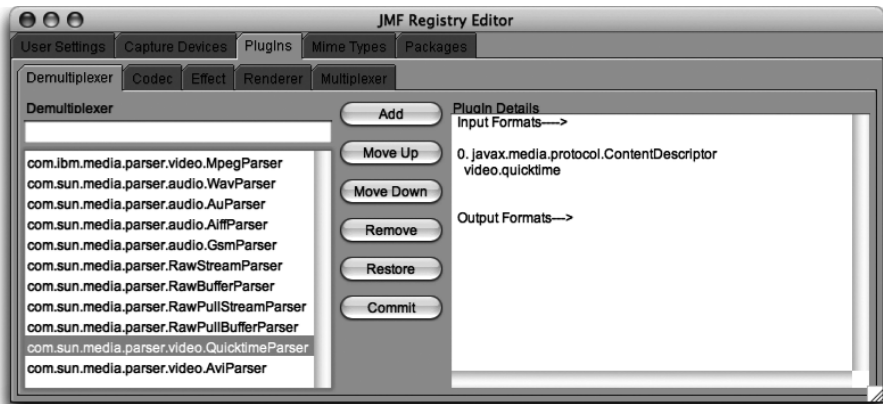


Abbildung 7.3 Registry Editor zur Konfiguration vom JMF

Für die einfachere Verwendung in Ihren Applikationen kopieren Sie aber am besten das Archiv `jmf.jar` in das Verzeichnis `/Library/Java/Extensions/`

(oder Sie packen das Archiv in das jeweilige Programmpaket). Damit können Sie dann das folgende Beispiel starten, ein einfaches Abspielprogramm sowohl für Video- als auch für Audiodaten:

```
//CD/examples/ch07/jmf/SimpleJMFPlayer/SimpleJMFPlayer.java
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import javax.media.*;
public class SimpleJMFPlayer extends Frame
    implements ControllerListener, WindowListener {
    private Player player = null;
    private Component visualComponent = null;
    private Component controlComponent = null;
```

Zunächst benötigen Sie ein `Player`-Objekt, das das Abspielen im Hintergrund steuert. Von diesem Objekt können Sie dann zwei Komponenten erfragen, die in die Benutzungsoberfläche integriert werden: Eine, die den Film selbst anzeigt, und eine für die Elemente, mit denen der Anwender den Film starten und stoppen kann. Die Kommunikation zwischen dem `Player` und der Oberfläche erfolgt über die im Interface `ControllerListener` definierte Methode `controllerUpdate()`.

```
public SimpleJMFPlayer(URL url) {
    try {
        player = Manager.createPlayer( url );
        player.addControllerListener(this);
        player.start();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    this.addWindowListener(this);
}
```

Der Konstruktor lässt am Anfang das nötige `Player`-Objekt erzeugen, wobei die URL der abzuspielenden Daten übergeben wird. Danach wird das Fenster-Objekt als Listener am `Player` registriert und die Wiedergabe gestartet. Hier fehlt zunächst noch die Verknüpfung mit der Oberfläche, diese wird in der folgenden Methode hergestellt.

```
public synchronized void controllerUpdate
    (ControllerEvent event) {
```

```

if (player == null) return;
if (event instanceof RealizeCompleteEvent) {
    visualComponent = player.getVisualComponent();
    controlComponent = player.getControlPanelComponent();
    if (visualComponent != null) {
        this.add( visualComponent, BorderLayout.CENTER );
    }
    if (controlComponent != null) {
        this.add( controlComponent, BorderLayout.SOUTH );
    }
    this.invalidate();
    this.pack();
}
else if (event instanceof EndOfMediaEvent) {
    player.setMediaTime( new Time(0) );
}
}

```

Für die verschiedenen Ereignisse (Fertig zur Anzeige, Ende der Wiedergabe usw.) ruft JMF die Methode `controllerUpdate()` auf und übergibt als Parameter geeignete `ControllerEvent`-Unterklassen. Am interessantesten ist der `RealizeCompleteEvent`, mit dem Sie aufgefordert werden, die Abspielkomponenten in die Oberfläche zu integrieren. Dazu fragen Sie die beiden Komponenten mit `getVisualComponent()` und `getControlPanelComponent()` ab, fügen sie zum `Frame` hinzu und lassen das Layout neu validieren – fertig.

Der zweite gezeigte `EndOfMediaEvent` tritt beim Ende der Wiedergabe auf. Hier wird in diesem Fall einfach nur der Film ganz »zurückgespult«, indem die Wiedergabezeit auf Null gesetzt wird.

```

public void stop() {
    if (player != null) {
        player.stop();
        player.deallocate();
    }
}

```

Wenn die Wiedergabe von außen abbrechbar sein soll, können Sie diese `stop()`-Methode zur Verfügung stellen. Dabei sollten Sie nicht nur das `Player`-Objekt stoppen, sondern mit `deallocate()` auch nicht mehr benötigten Speicher sofort freigeben.

Bevor das Programm beendet wird, rufen Sie `close()` auf, wodurch das `Player`-Objekt interne Aufräumarbeiten durchführen kann:

```
public void windowClosing(WindowEvent e) {
    if (player != null) {
        this.stop();
        player.close();
    }
    this.setVisible(false);
    this.dispose();
    System.exit(0);
}
//...
}
```

Nun können Sie das Abspielprogramm starten, und in dem Fenster wird nicht nur der Film an sich angezeigt, sondern auch die Steuerungselemente (siehe Abbildung 7.4) – beides hatten Sie ja in der Methode `controllerUpdate()` zum `Frame`-Layout hinzugefügt.



Abbildung 7.4 JMF spielt einen QuickTime-Film ab.

Um das Problem der schlecht unterstützten Medienformate zu umgehen, beschreibt der Artikel <http://www.onjava.com/pub/a/onjava/2002/12/23/jmf.html> die interessante Möglichkeit, QuickTime als JMF-Plugin für die Kodierungen und Dekodierungen zu verwenden. Wenn Sie allerdings für Ihre Filme die Standard-JMF-API nur zusammen mit einer systemabhängigen Bibliothek nutzen können, können Sie auch gleich QuickTime verwenden.

7.5 QuickTime for Java (QTJava)

QuickTime (QT) ist wohl eine der ältesten und ausgereiftesten Multimedia-bibliotheken, die es gibt. Seit Anfang der 1990er Jahre steht die Software für Mac OS und Windows zur Verfügung¹, und bei Mac OS X ist sie fester Bestand-

¹ Siehe <http://david.egbert.name/work/newmedia/quicktime/history/>

teil seit der ersten Version dieses Betriebssystems. Am bekanntesten ist QuickTime sicherlich dafür, Filmdateien abzuspielen (beispielsweise im QuickTime-eigenen *.mov-Format), aber es eignet sich genauso gut dazu, Musikdateien anzuhören, Bilddateien zu betrachten und zwischen einer Vielzahl von Formaten zu konvertieren, Audio- und Videodaten als Echtzeit-Datenstrom zu empfangen sowie 3D- und Virtual-Reality-Modelle zu bearbeiten. Mittlerweile ist QuickTime bei Version 6.5 angekommen, das aktuellste Installationspaket können Sie von der Seite <http://www.apple.com/quicktime/download/> herunterladen. Die Abspielsoftware QuickTime Player finden Sie im Verzeichnis /Programme/, die Konfiguration nehmen Sie über **Systemeinstellungen · QuickTime** vor.

Seit 1998 (damals war QuickTime 3.0 aktuell) gibt es eine Java-Schnittstelle zu dieser Technologie. Apple stellt mit »QuickTime for Java«, kurz »QTJava« oder »QTJ«, eine objektorientierte Zugriffsschicht auf die darunter liegende, prozedurale Programmierschnittstelle zur Verfügung. QTJava ist also kein reines Java und damit eigentlich nicht plattformunabhängig. Wenn Sie aber als Zielplattformen Mac OS (X) und Windows voraussetzen können, ist QuickTime die bessere Wahl gegenüber dem Java Media Framework, denn die Funktionalität und die Anzahl der unterstützten Formate sind deutlich höher. Linux-Anwender bleiben bei QuickTime derzeit leider komplett außen vor.

Mit QuickTime 6.4 hat Apple eine Veränderung an den QTJava-APIs vorgenommen, um die Schnittstellen zu vereinfachen und an Java 1.4 anzupassen – vieles, was QTJava vor einigen Jahren noch selbst mitbringen musste, ist mittlerweile Bestandteil der Standard-Klassenbibliothek. Insofern ist eine solche Schlankheitskur eigentlich zu begrüßen. Leider jedoch wurden die betreffenden Klassen und Methoden nicht nur einfach als veraltet (»deprecated«) markiert, einige wichtige alte Routinen verweigern nun mit Java 1.4 komplett ihren Dienst. Dies betrifft vor allem die Klassen aus dem Paket `quicktime.app.display`, die durch andere Klassen im Paket `quicktime.app.view` ersetzt wurden. Nahezu alle Java-Quelltexte müssen daran angepasst werden, wenn die Anwendungen auch mit einer aktuellen QuickTime-Version und Java 1.4 laufen sollen. Dass Sie QuickTime 6.4 benötigen, damit Java 1.4 unterstützt wird, betrifft übrigens nur Mac OS X – die Windows-Variante von QTJava kommt bereits seit QuickTime 6.0 mit dem JDK 1.4 klar.

Die Versionsnummern von QuickTime und von QTJava werden nicht synchron hochgezählt, auch wenn QTJava oft zusammen mit einer neuen QuickTime-Version aktualisiert wird. Zu QuickTime 6.4 und 6.5 gehört beispielsweise QTJava 6.1, zu QuickTime 6.3 die QTJava-Version 6.0. Die nötigen Java-Bibliotheken werden normalerweise vom QuickTime-Installationsprogramm auf die

Festplatte geschrieben. Einzige Ausnahme ist Mac OS X 10.2 – dort müssen Sie bei QuickTime 6.4 die Software-Aktualisierung bemühen, um QTJava nachträglich zu installieren.

Apples Portalseite für QTJava ist <http://developer.apple.com/quicktime/qtjava/>. Passen Sie aber auf, welche der Informationen Sie von dort verwenden, denn teilweise wurden die Texte noch nicht an MacOS X angepasst. Am besten laden Sie sich zunächst die Javadoc-Dokumentation von der Seite <http://developer.apple.com/quicktime/qtjava/javadocs.html> herunter. Sie finden dort zwei Versionen, QTJava 6.0 und QTJava 6.1. Da noch viele Quelltexte für die ältere QTJava-Version programmiert sind, wird im folgenden Beispiel ein einfacher Film-Abspieler für QTJava 6.0 entwickelt. Beachten Sie aber, dass Sie das Programm unter Mac OS X dann zwingend mit Java 1.3 ausführen müssen, sonst erhalten Sie Laufzeit-Fehlermeldungen.

7.5.1 QTJava 6.0 (QuickTime bis Version 6.3)

```
//CD/examples/ch07/qtjava/QTJava60Player/QTJava60Player.java
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import quicktime.*;
import quicktime.app.display.*;
import quicktime.app.QTFactory;
import quicktime.app.image.ImageDrawer;
```

Neben den typischen `import`-Anweisungen für eine AWT-Applikation mit Datei- und Netzwerkzugriffen wird vor allem das Paket `quicktime` mit einigen Unterpaketen eingebunden.

```
public class QTJava60Player extends Frame implements WindowListener {
    private Drawable myQTContent = null;
    private QTCanvas myQTCanvas = null;
```

Jedes QTJava60Player-Fenster hält zwei Referenzen. Das `Drawable`-Objekt verwaltet die darzustellenden Daten, beispielsweise einen Film. Der `QTCanvas` ist eine ganz gewöhnliche AWT-Komponente, mit der das `Drawable`-Objekt dann innerhalb der Benutzungsoberfläche angezeigt wird.

```
public QTJava60Player(String url) {
    try {
        QTSession.open();
```

```

myQTCanvas =
    new QTCanvas( QTCanvas.kInitialSize, 0.5F, 0.5F );
this.add ( myQTCanvas, BorderLayout.CENTER );
myQTContent = QTFactory.makeDrawable( url );
}

```

Bevor Sie irgendwelche QTJava-Objekte anlegen, müssen Sie zur Initialisierung von QuickTime ganz zu Anfang `QTSession.open()` aufrufen. Danach wird die `QTCanvas`-Komponente erzeugt, die dann in den `Frame` eingefügt wird. Mit `kInitialSize` wird die anfängliche Größe festgelegt, die beiden 0.5-Werte zentrieren den Film horizontal und vertikal innerhalb der Komponente. Schließlich wird mit `QTFactory.makeDrawable()` aus einer beliebigen URL ein darstellbares Multimedia-Objekt erzeugt. Was hier noch fehlt, ist die Verknüpfung zwischen dem `QTCanvas` und dem `Drawable`-Objekt.

```

catch (QTEException e) {
    if (QTSession.isInitialized()) {
        myQTContent = ImageDrawer.getQTLogo();
    }
    else {
        throw new RuntimeException( e.getMessage() );
    }
}

```

Aber zunächst prüfen Sie, ob ein Fehler aufgetreten ist. Wenn die `QTSession` initialisiert ist, wurde eventuell die Filmdatei nicht gefunden oder der Inhalt ist defekt. In diesem Fall wird einfach das QuickTime-Logo angezeigt, das sich mit einer `ImageDrawer`-Methode anfragen lässt. Ansonsten wird das Programm mit einer Fehlermeldung beendet.

```

if (myQTCanvas != null) {
    try {
        myQTCanvas.setClient( myQTContent, true );
    }
    catch (QTEException e) {
        throw new RuntimeException( e.getMessage() );
    }
}
this.addWindowListener(this);
}

```

Konnte QuickTime initialisiert und das `QTCanvas`-Objekt erzeugt werden, sagen Sie dem `QTCanvas`-Objekt nun mittels `setClient()`, welche Multi-

mediadaten in der Komponente angezeigt werden sollen. Der `true`-Parameter bewirkt, dass das Layout der AWT-Komponente mit den gesetzten Client-Daten neu berechnet wird.

Damit kann Ihr Frame bereits einen QuickTime-Film abspielen! Es bedarf keiner speziellen Methoden-Aufrufe zum Starten der Wiedergabe – sobald der `QTCanvas` angezeigt wird, werden auch automatisch verwaltete Steuerungselemente dargestellt, mit denen der Anwender das Abspielen kontrollieren kann. Was nun noch folgt, ist Code zum korrekten Beenden der Wiedergabe.

```
public void stop() {
    if (myQTCanvas != null) {
        myQTCanvas.removeClient();
        myQTCanvas = null;
    }
}
```

Die Methode `stop()` kann von außen aufgerufen werden, wenn die Wiedergabe abgebrochen werden soll – beispielsweise weil der Anwender einen entsprechenden Knopf angeklickt hat oder weil ein Applet-Dokument geschlossen wird. Dazu wird einfach die Verbindung zwischen dem `QTCanvas` und den Client-Daten mit der Methode `removeClient()` aufgelöst.

```
public void windowClosing(WindowEvent e) {
    this.stop();
    QTSession.close();
    this.setVisible(false);
    this.dispose();
    System.exit(0);
}
//...
}
```

Spätestens wenn Ihre Applikation beendet wird, sollten Sie die `QTSession` schließen, damit QuickTime Aufräumarbeiten durchführen kann. Anschließend können Sie dann – wie hier geschehen – das Fenster schließen und das Programm beenden.

Damit Sie das Programm kompilieren können, müssen Sie noch die `QTJava`-Klassen zum Suchpfad hinzufügen. In Xcode tragen Sie dazu das Archiv `/System/Library/Java/Extensions/QTJava.zip` in den Target-Einstellungen unter **Search Paths · Java Classes** ein (siehe Abbildung 7.5). Zum Ausführen müssen Sie dieses ZIP-Archiv später dann nicht mehr extra angeben, da es sich in einem der Standard-Erweiterungsverzeichnisse befindet.

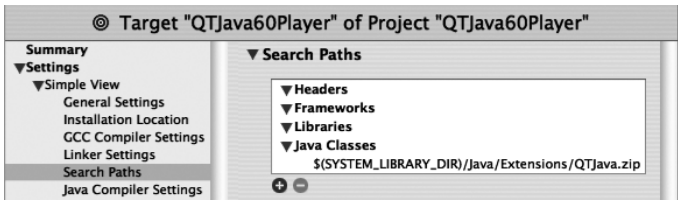


Abbildung 7.5 QTJava-Klassen zum Suchpfad hinzufügen

Da Sie hier mit veralteten Schnittstellen arbeiten, sollten Sie in den Xcode-Target-Einstellungen unter »Java Compiler Settings« den Eintrag »Show usage of deprecated API« aktivieren. So zeigt Ihnen Xcode nach dem Übersetzen neben dem Quelltext, welche Stellen kritisch für die Aufwärtskompatibilität sind. Wenn Sie `javac` direkt aufrufen, können Sie dafür die Option `-deprecation` setzen.

Sie werden sehen, dass Sie das Interface `quicktime.app.display.Drawable`, die Klasse `quicktime.app.QTFactory` und vor allem die Klasse `quicktime.app.display.QTCanvas` bei neuen Projekten nicht mehr einsetzen sollten. Geeigneter Ersatz wird Ihnen im folgenden Abschnitt vorgestellt.

Vor dem Ausführen müssen Sie noch sicherstellen, dass die richtige Java-Version gestartet wird. Dazu ändern Sie beim Executable »java« den Programm-pfad unter »Path to Executable« auf `/System/Library/Frameworks/JavaVM.framework/Versions/1.3.1/Commands/java` (siehe Abbildung 7.6).

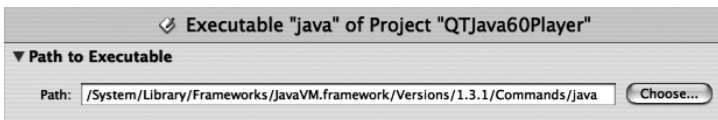


Abbildung 7.6 QTJava 6.0 mit Java 1.3 ausführen

Wenn Sie das Programm nun übersetzen und starten, öffnet sich ein Fenster, in dem neben dem Medien-Ausgabebereich automatisch auch eine Steuerungs-leiste eingeblendet wird (siehe Abbildung 7.7). Mit dem Dreieck-Knopf können Sie die Wiedergabe starten (und auch wieder anhalten) – um die korrekte Behandlung der entsprechenden Ereignisse kümmert sich QTJava.



Abbildung 7.7 Filme mit QTJava 6.0 abspielen

7.5.2 QTJava 6.1 (QuickTime ab Version 6.4)

Um ein solches Abspielprogramm mit dem aktuellen QTJava 6.1 zu realisieren, müssen Sie vor allem den Konstruktor der Klasse anpassen – hier wird ja das Film-Objekt erzeugt und in die Oberfläche eingebunden. Wenn Sie Java 1.4 verwenden, sind diese Änderungen zwingend. Und da die vorgestellten Klassen auch mit Java 1.3 funktionieren, sollten Neuentwicklungen immer wie folgt aufgebaut sein:

```
//CD/examples/ch07/qtjava/QTJava61Player/QTJava61Player.java
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import quicktime.*;
import quicktime.app.view.*;
import quicktime.std.*;
import quicktime.std.movies.*;
import quicktime.std.movies.media.DataRef;
```

Es werden die typischen Pakete eingebunden, dazu das QTJava 6.1-Paket `quicktime.app.view` sowie das ältere (aber immer noch gültige) Paket `quicktime.std` für diverse Konstanten.

```
public QTJava61Player(String url) {
    try {
        QTSession.open();
        DataRef ref = new DataRef( url );
        Movie movie =
            Movie.fromDataRef( ref,
```

```

        StdQTConstants4.newMovieAsyncOK |
        StdQTConstants.newMovieActive );
MovieController ctrl =
    new MovieController( movie );
QTComponent component =
    QTFactory.makeQTComponent( ctrl );
this.add ( component.asComponent(),
          BorderLayout.CENTER );
}
catch (QTEException e) { /* ... */ }
this.addWindowListener(this);
}

```

Auch mit dem aktuellen QTJava muss zunächst eine `QTSession` geöffnet werden, um QuickTime zu initialisieren. Anschließend wird aus der übergebenen URL ein `DataRef`-Objekt erzeugt, mit dem in QuickTime die unterschiedlichsten Datenquellen referenziert werden.

Achtung: `DataRef` kommt mit den aus einem `File`-Objekt erzeugten URL-Zeichenketten nicht zurecht. Um Dateien zu referenzieren, müssen Sie die `file://`-URLs für den `DataRef`-Konstruktor von Hand zusammenbauen.

Mit der erhaltenen Daten-Referenz können Sie nun ein `Movie`-Objekt generieren lassen. Dazu übergeben Sie die Referenz sowie ein paar von Apple empfohlene Konstanten an `Movie.fromDataRef()`. Das `Movie`-Objekt alleine bietet zu wenig Funktionalität zum Abspielen, daher verpacken Sie es in einem `MovieController`. Aus diesem Steuerungsobjekt können Sie dann schließlich mit `QTFactory.makeQTComponent()` eine Komponente für die Benutzungsoberfläche erzeugen lassen.

Beachten Sie, dass es sich hier um die Klasse `quicktime.app.view.QTFactory` handelt und nicht um die gleichnamige, aber veraltete Klasse aus dem Paket `quicktime.app`!

Diese Komponente fügen Sie dann in die Oberfläche ein. Da `makeQTComponent()` eine `quicktime.app.view.QTComponent`-Referenz zurückgibt, müssen Sie das Objekt dafür entweder in eine `java.awt.Component` umwandeln (casten) oder – besser – eine passende Referenz mit `asComponent()` erfragen.

Wenn Sie sich die `quicktime.std.movies.MovieController`-Referenz als Objektvariable merken, können Sie das Abspielen außerhalb des Konstruktors beeinflussen. Beispielsweise können Sie der Methode `play()` einen Parameter für die Abspielgeschwindigkeit übergeben – mit `1.0F` wird der Film in normaler Geschwindigkeit angezeigt, mit `0.0F` wird der Film angehalten.

In der `windowClosing()`-Listener-Methode bleibt alles beim Alten: Unmittelbar vor dem Programmende wird die `QTSession` geschlossen:

```
public void windowClosing(WindowEvent e) {
    QTSession.close();
    this.setVisible(false);
    this.dispose();
    System.exit(0);
}
// ...
}
```

Die Programmierung mit QTJava 6.1 erfordert also minimal mehr Aufwand als mit Version 6.0. Dafür funktioniert die Lösung sowohl mit Java 1.3 als auch mit Java 1.4 – und für die flexiblen Steuerungsmöglichkeiten mit dem `MovieController` müssten Sie bei QTJava 6.0 sowieso denselben Aufwand betreiben.

Wenn Sie sich weitere Beispiele zu QTJava ansehen möchten, finden Sie auf <http://developer.apple.com/samplecode/Java/idxQuickTime-date.html> eine Liste mit Archiven zu diversen Themen. Aber Achtung: Die meisten davon laufen nur mit QTJava 6.0, Apple hat sie leider noch nicht an QTJava 6.1 angepasst. Von der Seite <http://developer.apple.com/sdk/qtjavasdk.html> können Sie ein QTJava-Entwicklungspaket (SDK) herunterladen, das alle diese Beispiele und die API-Dokumentation enthält – allerdings nur für Windows. Um die Situation für QTJava-Programmierer zu verbessern, hat sich gerade die Netzgemeinschaft »OpenQTJ« gegründet, die auf der Seite <https://open-qtj.dev.java.net/> aktualisierte Beispiele, Anleitungen und Hilfestellungen bei Problemen anbietet.

QuickTime bietet auch vielfältige Im- und Exportmöglichkeiten. Beispielsweise können Sie folgenden QTJava-Quelltext verwenden, um eine Bilddatei im Apple-eigenen PICT-Format in das JPEG-Format umzuwandeln:

```
import quicktime.io.QTFile;
import quicktime.qd.Pict;
```

```

import quicktime.std.image.GraphicsExporter;
import quicktime.util.QTUtils;
//...
Pict p = Pict.fromFile( new java.io.File("mein.pict") );
GraphicsExporter export =
    new GraphicsExporter( QTUtils.toOSType("JPEG") );
export.setInputPicture( p );
QTFile out = new QTFile( "mein.jpg" );
export.setOutputFile( out );
export.doExport();

```

Wenn Sie das PICT-Bild innerhalb des Java-Programms weiterverarbeiten möchten, können Sie vom QuickTime-`GraphicsImporter` ein Image-Objekt erzeugen lassen:

```

import java.awt.*;
import quicktime.app.view.*;
import quicktime.io.QTFile;
import quicktime.qd.QDRect;
import quicktime.std.image.GraphicsImporter;
// ...
GraphicsImporter import =
    new GraphicsImporter( QTUtils.toOSType("PICT") );
import.setDataFile( new QTFile("mein.pict") );
QDRect rect = import.getNaturalBounds();
GraphicsImporterDrawer drawer =
    new GraphicsImporterDrawer( import );
QTImageProducer producer =
    new QTImageProducer( drawer,
        new Dimension( rect.getWidth(), rect.getHeight() ) );
Image img = Toolkit.getDefaultToolkit().createImage( producer );

```

7.6 Sound/Musik

Zum Abspielen von Musik, beispielsweise im MP3-Format, setzen Sie am besten QuickTime for Java ein, wie es im vorangegangenen Abschnitt beschrieben wurde. Ob ein Film oder eine Sounddatei abgespielt wird, ändert an der QTJava-Programmstruktur nichts. Wenn Sie auf MP3-Dateien verzichten, können Sie alternativ auch das Java Media Framework verwenden.

Speziell zur Sound-Verarbeitung hat Sun die Java Sound API (JavaSound) entworfen, siehe <http://java.sun.com/products/java-media/sound/>. Diese portablen

Klassen stehen seit Java 1.3 im Paket `javax.sound` zur Verfügung – beachten Sie aber, dass die Audio-Eingaberoutinen beim Mac OS X-Java erst ab Java 1.4 implementiert sind. Und auch in Java 1.4 kommen für die Eingabe nur bestimmte Werte in Frage (44,1 kHz Sample-Frequenz, PCM-Kodierung, mono oder stereo, 8- oder 16-Bit-Samples). Liegt die Audio-Eingabedatei in einem anderen Format vor, müssen Sie sie vor der Verwendung passend umwandeln.

Was mit JavaSound unter Mac OS X leider überhaupt nicht funktioniert, ist die Ansteuerung von MIDI-Geräten. Sollten Sie dies benötigen, müssen Sie das Mac OS X-spezifische CoreAudio-Framework einsetzen, für das Apple Java-Hüllklassen bereitstellt. Die Klassen sind im Archiv `/System/Library/Java/Extensions/CoreAudio.jar` gespeichert und gehören zum Paket `com.apple.audio` (bzw. zu diversen Unterpaketen davon). Eine Übersicht über CoreAudio finden Sie auf den Seiten <http://developer.apple.com/audio/coreaudio.html> und <http://developer.apple.com/documentation/MusicAudio/Reference/CoreAudio/>, die API-Dokumentation der Java-Klassen liegt auf Ihrer Festplatte im Verzeichnis `/Developer/Documentation/CoreAudio/Java/`. Allerdings ist die Java-Dokumentation mehr als dürftig, und die Klassen wurden längere Zeit nicht weiterentwickelt. Insofern sollten Sie nach Möglichkeit auch hier QTJava einsetzen, das zudem wenigstens zwischen Windows und Mac OS X portabel ist.

7.7 Drucken

Die schlechte Nachricht zuerst: Die Java 1.4-Druck-API im Paket `javax.print` ist derzeit unter Mac OS X nicht nutzbar – Apple hat sie einfach noch nicht vollständig implementiert. Die gute Nachricht: Alle früheren Druck-APIs in den Paketen `java.awt` und `java.awt.print` sind voll funktional und können auch mit Java 1.4 genutzt werden. Falls Sie dennoch eine `javax.print`-ähnliche Schnittstelle benötigen, empfiehlt Apple, bis auf weiteres auf »J2PrinterWorks« von <http://www.wildcrest.com/> zurückzugreifen.

Damit Sie einen Eindruck bekommen, wie Mac OS X die vorhandenen Druck-Schnittstellen umsetzt, folgen zwei kurze Beispiele. Zunächst wird die `java.awt`-Druck-API vorgestellt, die mit Java 1.1 eingeführt und mit Java 1.3 erweitert wurde:

```
//CD/examples/ch07/print/Print11/Print11.java
import java.awt.*;
public class Print11 extends Frame {
    public void print() {
        JobAttributes jobAttr = new JobAttributes();
        PageAttributes pageAttr = new PageAttributes();
```

```

jobAttr.setDialog( JobAttributes.DialogType.NATIVE );
pageAttr.setColor( PageAttributes.ColorType.MONOCHROME );

PrintJob job = this.getToolkit().getPrintJob( this,
                                             "Java 1.1/1.3 DruckTest",
                                             jobAttr, pageAttr );

if (job == null) return;

```

Zentrale Klasse ist hier `java.awt.PrintJob`. Sie repräsentiert einen Druckauftrag, den Sie mit `JobAttributes`- und `PageAttributes`-Objekten konfigurieren können. In diesem Beispiel wird mit `DialogType.NATIVE` festgelegt, dass Java den System-Druckdialog anzeigen soll (dieser Aufruf ist eigentlich unnötig, da dies die Standardeinstellung ist). `ColorType.MONOCHROME` bestimmt entsprechend eine schwarz-weiße Ausgabe.

Das `PrintJob`-Objekt legen Sie nicht selbst an, sondern Sie lassen es von `Toolkit.getPrintJob()` erzeugen. Ein `Toolkit`-Objekt stellt die tatsächliche AWT-Implementierung des Systems dar, und die meisten Methoden darin rufen Sie nie direkt auf. Einige wenige nützliche Hilfsroutinen – beispielsweise zur Ermittlung der Bildschirmgröße oder eben zum Erzeugen eines Druckauftrags – benötigt man aber relativ häufig. Das passende `Toolkit`-Objekt erfragen Sie entweder mit `getToolkit()` von einer AWT-Komponente oder aber mit `Toolkit.getDefaultToolkit()`. Hier wird ersteres verwendet, da die Klasse von `Frame` erbt und damit selbst eine Komponente ist – außerdem benötigen Sie die `Frame`-Referenz für `getPrintJob()`, weil Sie den Druckauftrag damit dem `Frame` zuordnen.

Durch den Aufruf von `getPrintJob()` wird auch der Druckdialog angezeigt. Wenn der Benutzer darin auf »Abbrechen« klickt, erhalten Sie als Rückgabe die `null`-Referenz und können Ihrerseits die Druckausgabe beenden.

Der Methodenname `print()` ist übrigens beliebig wählbar und durch keine Schnittstelle fest vorgegeben.

```

Graphics g = job.getGraphics();
Dimension pageSize = job.getPageDimension();
g.drawOval( 0, 0, pageSize.width, pageSize.height );
g.drawString( "Breite " + pageSize.width + " Pixel, Höhe "
             + pageSize.height + " Pixel.",
             pageSize.width/2, pageSize.height/2 );
g.dispose();
job.end();
}

```

```
// ...  
}
```

Hat der Anwender »Drucken« angeklickt, können Sie vom `PrintJob`-Objekt einen `Graphics`-Kontext erfragen und darin ganz normal Grafik und Text ausgeben. Sobald Sie fertig sind, müssen Sie den `Graphics`-Kontext mit `dispose()` freigeben und den Druckauftrag mit `end()` abschließen – sonst wird nichts gedruckt!

Wenn Sie das Programm nun laufen lassen, zeigt das System zuerst einen Dialog zur Konfiguration des Papierformats an (siehe Abbildung 7.8). Ob dieser Dialog wirklich erscheint, hängt davon ab, ob das System aus dem übergebenen `PageAttributes`-Objekt ausreichend Informationen extrahieren kann oder nicht.

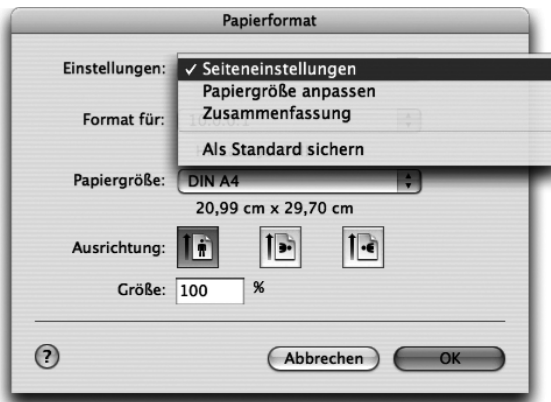


Abbildung 7.8 Konfiguration des Papierformats

Anschließend wird der eigentliche Druckdialog dargestellt, in dem Sie die Mac-typische Vielzahl von Einstellmöglichkeiten vorfinden (siehe Abbildung 7.9). Unten links im Dialog sehen Sie auch die Knöpfe, mit denen die Druckausgabe automatisch in eine PDF-Datei umgelenkt werden kann.

Neben diesem systemspezifischen Druckdialog bietet Java auch einen Dialog, der auf allen Systemen gleich aussieht (siehe Abbildung 7.10). Sie können diese Form auswählen, indem Sie bei `JobAttributes.setDialog()` den Wert `JobAttributes.DialogType.COMMON` übergeben. Überlegen Sie sich aber gut, ob Ihnen der Java-eigene Druckdialog irgendwelche Vorteile bringt, denn der Anwender wird ihn als Fremdkörper in einer ansonsten an das System angepassten Applikation empfinden.



Abbildung 7.9 Konfiguration des Druckauftrags

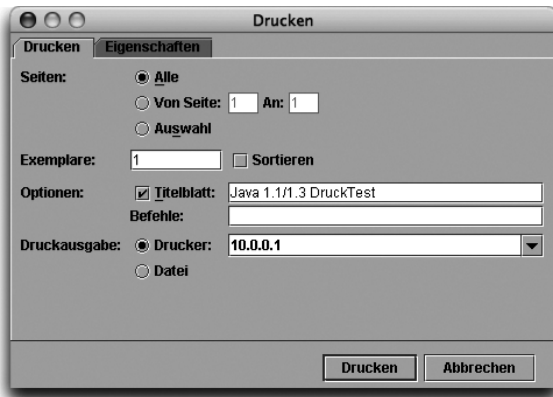


Abbildung 7.10 Javas systemübergreifender Druckdialog

Der große Nachteil der `java.awt`-Druckschnittstelle ist, dass Java 2D von ihr nicht unterstützt wird. Daher wurde mit Java 1.2 eine neue Druck-API im Paket `java.awt.print` eingeführt, die neben der Java 2D-Fähigkeit auch gute Konfigurationsmöglichkeiten für das Papierformat besitzt. Zudem benötigen Sie hierbei keine `Frame`-Referenz wie beim `java.awt.PrintJob`, daher ist die Ansteuerung beispielsweise auch aus einer `main()`-Methode heraus möglich:

```
//CD/examples/ch07/print/Print12/Print12.java
import java.awt.print.*;
// ...
```

```

PrinterJob job = PrinterJob.getPrinterJob();

PageFormat seitenformat = job.defaultPage();
seitenformat.setOrientation( PageFormat.PORTRAIT );

Printable daten = new Druckdaten();
job.setPrintable( daten, seitenformat );

if ( job.printDialog() ) {
    try {
        job.print();
    }
    catch (PrinterException e) {
        e.printStackTrace();
    }
}

```

Diesmal wird kein `java.awt.PrintJob`-Objekt verwendet, sondern ein `java.awt.print.PrinterJob`-Objekt. Dieses erzeugen Sie wiederum nicht direkt, sondern Sie ermitteln ein passendes Objekt mit `getPrinterJob()`. Mit `defaultPage()` erfragen Sie danach ein Standard-`PageFormat`-Objekt für das Papierformat, das dann noch konfiguriert werden kann – hier wird mit `setOrientation()` das Koordinatensystem eingestellt, das seinen Ursprung links oben hat.

Die Besonderheit der `java.awt.print`-API kommt jetzt: Die eigentliche Methode für die Druckausgabe wird in einer Klasse realisiert, die das `Printable`-Interface implementiert. Bei diesem Beispiel ist dies die Klasse `Druckdaten`, die Sie weiter unten aufgelistet finden. Dieses `Printable`-Objekt wird zusammen mit dem `PageFormat`-Objekt mittels `setPrintable()` an den Druckauftrag übergeben.

Nun können Sie die `print()`-Methode des Druckauftrags zum sofortigen Ausdrucken aufrufen. Oder aber Sie rufen vorher noch `printDialog()` auf, damit der Druckdialog angezeigt wird und der Anwender die voreingestellten Werte noch verändern kann. `printDialog()` gibt `false` zurück, wenn der Benutzer »Abbrechen« angeklickt hat.

Die Klasse mit der eigentlichen Ausgabe-Methode ist wie folgt implementiert:

```

//CD/examples/ch07/print/Print12/Druckdaten.java
import java.awt.*;
import java.awt.print.*;
public class Druckdaten implements Printable {

```

```

public int print( Graphics graphics,
                 PageFormat pageFormat, int pageIndex ) {
    if (pageIndex > 0) return NO_SUCH_PAGE;

```

Durch das Printable-Interface muss die Klasse die Methode print() implementieren, der ein Graphics-Kontext, ein Seitenformat-Objekt und die aktuelle Seitenzahl übergeben wird. Java ruft die print()-Methode einfach immer wieder mit aufsteigender Seitenzahl auf – bis das Ende erreicht ist und print() den Wert Printable.NO_SUCH_PAGE zurückgibt. Hier kann nur eine einzige Seite mit dem Index 0 gedruckt werden.

```

    Graphics2D g2d = (Graphics2D)graphics;
    g2d.translate( pageFormat.getImageableX(),
                  pageFormat.getImageableY() );
    int breite = (int)pageFormat.getImageableWidth();
    int hoehe = (int)pageFormat.getImageableHeight();

    g2d.drawOval( 0, 0, breite, hoehe );
    g2d.drawString( "Breite " + breite + " Pixel, Höhe " + hoehe
                    + " Pixel.", breite/2, hoehe/2 );
    return PAGE_EXISTS;
}
}

```

Wie bei Java2D üblich können Sie den Graphics-Kontext in einen Graphics2D-Kontext umwandeln. Anschließend wird die Ausgabe mit translate() in den sichtbaren Bereich verschoben, dessen Startpunkt mit getImageableX() und getImageableY() aus dem Seitenformat ermittelt wird. Ebenso werden die Breite und Höhe des Ausgabebereichs abgefragt, wodurch die Seitenränder korrekt beachtet werden. Nach erfolgreicher Ausgabe einer Seite muss print() schließlich noch den Wert Printable.PAGE_EXISTS zurückgeben.

7.8 Literatur & Links

► Java 2D

- J. B. Knudsen, »Java 2D Graphics«, O'Reilly 1999
Dieses Buch ist zwar schon etwas älter, bietet aber eine hervorragende Programmieranleitung für 2D-Grafik mit Java.
- <http://java.sun.com/products/java-media/2D/>
Auf dieser Seite listet Sun alle relevanten Dokumentationen zu Java 2D und verwandten APIs auf.

► **Java 3D**

- D. Selman, »Java 3D Programming«, Manning Publications 2002
Dieses Buch deckt viele weiterführende Informationen zu Java 3D ab, die von der Standarddokumentation nicht oder nur unzureichend behandelt werden.

► **OpenGL**

- Shreiner/Woo/Neider, »OpenGL Programming Guide«, 4th ed., Addison-Wesley 2003
Auch wenn in diesem Buch C und nicht Java als Programmiersprache eingesetzt wird, bietet es einen guten Überblick über die wesentlichen Ideen der OpenGL-Programmierung. Und Java-APIs wie JOGL sind eh recht nahe an den C-Routinen.
- <http://www.opengl.org/>
Die OpenGL-Homepage enthält unter anderem eine Programmierführung und das Referenzhandbuch.

► **QuickTime**

- Vogt/Kastenholz, »QuickTime 6«, Galileo Press 2002
Dieses Buch ist keine Programmieranleitung für QTJava, sondern eine ausführliche Anleitung für die Anwendung der QuickTime-Technologie – aber das Verständnis dafür ist unerlässlich, wenn Sie QuickTime ernsthaft programmieren wollen.

8 Werkzeuge

8.1	Ant und Maven	385
8.2	JUnit	398
8.3	Decompiler	403
8.4	Obfuscators	404
8.5	Bytecode Viewer (Disassembler)	406
8.6	Profiler	408
8.7	JavaBrowser	410
8.8	Jikes	412
8.9	Groovy	413
8.10	UML-Modellierung	416
8.11	<oXygen/> XML-Editor	417
8.12	Literatur & Links	418

- 1 **Grundlagen**
- 2 **Entwicklungsumgebungen**
- 3 **Grafische Benutzungsoberflächen (GUI)**
- 4 **Ausführbare Programme**
- 5 **Portable Programmierung**
- 6 **Mac OS X-spezifische Programmierung**
- 7 **Grafik und Multimedia**
- 8 **Werkzeuge**
- 9 **Datenbanken und JDBC**
- 10 **Servlets und JavaServer Pages (JSP)**
- 11 **J2EE und Enterprise JavaBeans (EJB)**
- 12 **J2ME und MIDP**
- A **Kurzeinführung in die Programmiersprache Java**
- B **Java auf Mac OS 8/9/Classic**
- C **Java 1.5 »Tiger«**
- D **System-Properties**
- E **VM-Optionen**
- F **Xcode- und Project Builder-Einstellungen**
- G **Mac OS X- und Java-Versionen**
- H **Glossar**
- I **Die Buch-CD**

8 Werkzeuge

»Sieht so aus, als hätten die unser Denkmal heute Nacht schon ohne uns enthüllt – hol den Vorschlaghammer!« (Wir sind Helden)

Bei der Software-Entwicklung werden neben den Möglichkeiten der Entwicklungsumgebung oft viele nützliche Werkzeuge eingesetzt, die Ihnen das Leben als Programmierer einfacher machen. In diesem Kapitel wird Ihnen eine Auswahl solcher Werkzeuge vorgestellt – die Liste ist sicherlich nicht komplett, und oft gibt es zu einem Bereich noch zahlreiche weitere Lösungen, aber die wichtigsten und bekanntesten Programme sind dabei. Am besten fassen Sie die Beschreibungen als Hinweise auf, wo sich eine weitere Recherche für Sie lohnen könnte.

Die Anwendungen sind zum Teil bereits im System vorhanden, zum Teil stehen sie frei im Netz zur Verfügung oder müssen kommerziell erworben werden. Bei den kommerziellen Werkzeugen ist sichergestellt, dass eine kostenlose Testversion vorhanden ist.

Die meisten der vorgestellten Programme sind übrigens selbst größtenteils oder komplett in Java realisiert. Falls Java für Sie relativ neu ist, sind diese Anwendungen auch hervorragende Beispiele, welche Möglichkeiten Ihnen die Java-Technologie bietet – beispielsweise hinsichtlich der Gestaltung von komplexen Benutzeroberflächen.

8.1 Ant und Maven

Ant ist mittlerweile *das* Standardwerkzeug, um Java-Quelltexte zu übersetzen und die erzeugten Klassen zu geeigneten Programmarchiven zusammenzufassen (d.h. zu den bereits bekannten JAR-Archiven oder zu Web- bzw. Enterprise-Archiven WAR und EAR). Dazu steht neben dem eigentlichen Compiler-Aufruf eine Vielzahl von Befehlen zur Verfügung, unter anderem zum Erzeugen der Javadoc-Dokumentation, zum Packen von ZIP-Archiven und zum Kopieren von Dateien und Verzeichnissen. Ant ist damit ganz speziell für Java-Projekte ein Nachfolger von `make`. Der große Vorteil dieser Lösung ist, dass die Erzeugung eines Programms (der Build-Prozess) unabhängig von der Entwicklungsumgebung erfolgt – Ihr Projekt ist damit also nicht mehr zwingend an die IDE gebunden, mit der Sie die Entwicklung begonnen haben. Und die Bindung an Ant fällt relativ leicht, denn es handelt sich hierbei um ein kostenloses Open Source-Werkzeug!

Bei Apples Entwicklerwerkzeugen wird Ant optional installiert, wenn Sie bei den »Xcode Tools« die angepasste Installation gewählt und den Eintrag »Java Application Servers Development« zusätzlich markiert haben. Nachträglich können Sie die Software installieren, wenn Sie auf der Xcode-CD den Installer `ApplicationsServerDev.pkg` im Verzeichnis `Packages` aufrufen. Sie finden Ant dann im Verzeichnis `/Developer/Java/J2EE/Ant/` auf Ihrem Startvolumen. Alternativ können Sie aber auch ganz einfach die neueste Version von der offiziellen Ant-Homepage <http://ant.apache.org/> herunterladen und in ein beliebiges Verzeichnis auspacken.

Falls Sie Ant im Terminal verwenden möchten, müssen Sie das `bin`-Verzeichnis zur `PATH`-Umgebungsvariablen hinzufügen. Zusätzlich können Sie noch die Variable `ANT_HOME` setzen, aber bei neueren Ant-Versionen ist dies normalerweise nicht mehr nötig. Wenn Sie das Ant-Archiv also in Ihrem Benutzerverzeichnis ausgepackt haben, sieht das in etwa wie folgt aus:

```
setenv PATH ~/apache-ant-1.6.1/bin:$PATH
setenv ANT_HOME ~/apache-ant-1.6.1
```

Nun können Sie in der Shell das `ant`-Kommando aufrufen. Wenn Sie die Pfade korrekt gesetzt haben, sollten Sie folgende Ausgabe sehen:

```
[straylight:~] much% ant -version
Apache Ant version 1.6.1 compiled on February 12 2004
```

Als Nächstes müssen Sie eine Konfigurationsdatei schreiben, die festlegt, wie Ihr Projekt übersetzt werden soll. Diese Datei wird auch als **Buildfile** bezeichnet und heißt meistens `build.xml`. An der Dateinamenserweiterung sehen Sie es schon: Als Dateiformat wird XML verwendet. Speichern Sie diese Datei im Projektverzeichnis ab. Das folgende Listing kann Ihnen dabei als Grundlage für eigene Buildfiles dienen:

```
<!-- CD/examples/ch08/ant/build.xml -->
<project name="Ant-Test" default="run" basedir=".">
  <target name="compile">
    <echo message="Java compilieren..."/>
    <javac srcdir="."/>
  </target>

  <target name="run" depends="compile">
    <echo message="Java ausführen..."/>
    <java classname="HalloWelt">
      <classpath>
```

```

        <pathelement path="\${basedir}"/>
    </classpath>
</java>
</target>
</project>

```

Listing 8.1 Ant-Buildfile build.xml

Das XML-Wurzelement ist bei Ant-Buildfiles `project`, dem Sie optional einen Namen geben können. Mit `basedir` müssen Sie ein Arbeitsverzeichnis festlegen. Oft ist dies einfach nur der Punkt, also das aktuelle Verzeichnis.

Innerhalb des Projekts befinden sich beliebig viele `target`-Elemente. Ein Target stellt einen Arbeitsschritt im Build-Prozess dar. Eines davon können und sollten Sie zum Standard-Target machen, indem Sie es beim `default`-Attribut im `project`-Element eintragen.

Typisch ist ein `compile`-Target. Hier wird zunächst mit der `echo`-Anweisung (einem so genannten »Task«) eine Statusmeldung ausgegeben, danach wird der `javac`-Task gestartet. Die Java-Quelltexte werden dabei im aktuellen Verzeichnis gesucht, aber Sie können beim `srcdir`-Attribut natürlich ein beliebiges anderes Verzeichnis vorgeben.

Bevor ein Programm ausgeführt werden kann, muss es übersetzt worden sein. Diese Abhängigkeit ist daher beim `run`-Target im `depends`-Attribut vermerkt – um die korrekte Ausführungsreihenfolge der Targets kümmert sich Ant ohne weiteres Zutun. Ein Target kann auch von mehreren anderen abhängen, die dann mit Komma getrennt eingetragen werden. Das Ausführen selbst geschieht mit dem `java`-Task, dem die Klasse mit der `main()`-Methode übergeben wird. Außerdem wird an dieser Stelle – als untergeordnetes Element – der Klassenpfad zusammengebaut, der in diesem Buildfile nur aus einem einzigen `pathelement` besteht. Dazu wird die vordefinierte Ant-Property `basedir` mit der `{}`-Notation ausgewertet; der Wert dieser Property wurde im `project`-Element gesetzt.

Der Aufruf im Terminal ist nun denkbar einfach: Führen Sie im Projektverzeichnis – also dort, wo sich die Datei `build.xml` befindet – das `ant`-Kommando aus:

```

[straylight:examples/ch08/ant] much% ant
Buildfile: build.xml
compile:
    [echo] Java kompilieren...
    [javac] Compiling 1 source file

```

```

run:
    [echo] Java ausführen...
    [java] Hallo Ant-Welt!
BUILD SUCCESSFUL
Total time: 6 seconds

```

Weil beim Ant-Aufruf keine weiteren Parameter angegeben wurden, wird das Standard-Target ausgeführt, das im `project`-Element mit dem `default`-Attribut festgelegt wurde, hier also `run`. Da dieses Target von `compile` abhängt, wird letzteres zuerst aufgerufen.

Wenn Sie als Parameter die Option `-verbose` übergeben, erhalten Sie ausführlichere Angaben darüber, wie Ant den Build-Prozess durchführt. Mit `-buildfile` können Sie einen anderen Namen für das Buildfile angeben, falls Sie nicht `build.xml` verwenden möchten. Am Ende der Parameterliste schließlich können Sie einen oder mehrere Target-Namen angeben, die ausgeführt werden sollen, zum Beispiel `ant compile`.

8.1.1 Ant in Xcode

Ant kann in Xcode integriert werden und den Xcode-eigenen Build-Mechanismus ersetzen. Damit ist es möglich, dass Sie ein Ant-Buildfile mit verschiedenen Entwicklungsumgebungen einsetzen – auch auf unterschiedlichen Betriebssystemen!

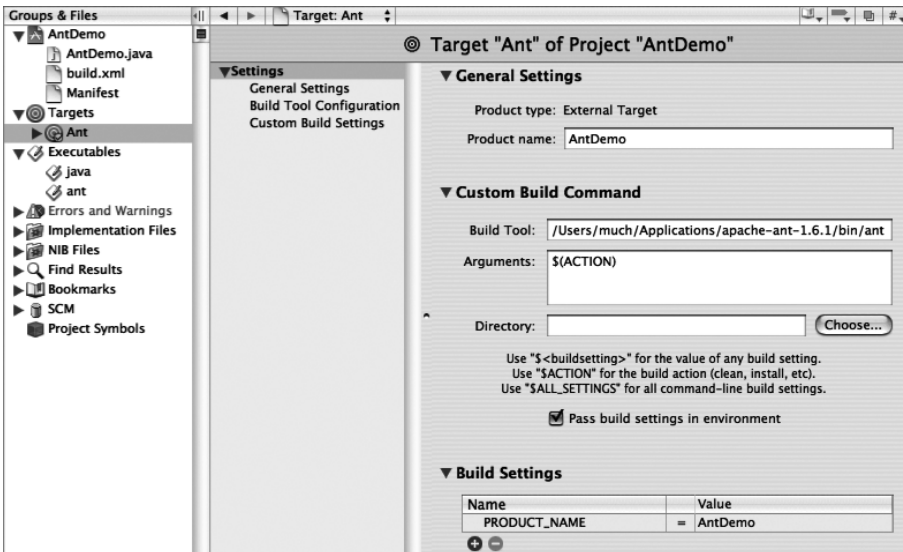


Abbildung 8.1 Ant-Target in Xcode konfigurieren

Das Beispielprojekt ist hier als »Java Tool« erzeugt worden, aber Sie können natürlich auch jeden anderen Java-Projekttyp verwenden. Führen Sie nun einen **Ctrl**-Klick auf die Zeile »Targets« in der linken Spalte »Groups & Files« aus. Im Kontext-Popup-Menü wählen Sie dann **Add · NewTarget** und im daraufhin erscheinenden Dialog **Special Targets · External Target** an. Als Namen für das Target können Sie beispielsweise »Ant« eingeben.

Selektieren Sie nun das neu erzeugte Target und konfigurieren Sie das »Custom Build Command« (siehe Abbildung 8.1). Unter »Build Tool« tragen Sie den absoluten Pfad des ant-Befehls ein. Bei »Arguments« lassen Sie `$(ACTION)` stehen, dann übergibt Xcode an das Ant-Buildfile den Parameter `clean` bzw. `build` – je nachdem, ob das Target zurückgesetzt oder kompiliert werden soll. Bei den »Build Settings« ist nur der Eintrag `PRODUCT_NAME` nötig. Als Wert tragen Sie den gewünschten Namen für Ihre Anwendung ein. Damit dieser Name auch im Ant-Buildfile ausgewertet werden kann, muss das Kästchen »Pass build settings in environment« aktiv sein.

Das Xcode-Target, das beim Anlegen des Projekts automatisch erzeugt wird (in diesem Beispiel »AntDemo«), können Sie nun einfach löschen: Markieren Sie den Eintrag und drücken Sie **←**. Dadurch wird das neue Target auch gleich zum aktiven Target. Wenn Ihr Projekt mehrere Targets besitzt, können Sie dies ansonsten ganz links oben im Xcode-Fenster mit dem Target-Popup umschalten.

Nun fügen Sie ein Ant-Buildfile zu Ihrem Xcode-Projekt hinzu. Rufen Sie dazu **File · New File...** auf, wählen Sie im erscheinenden Assistenten »Empty File in Project« und legen Sie die Datei `build.xml` im Projektverzeichnis an. Für ein gutes Zusammenspiel mit Xcode könnte das Buildfile in etwa wie folgt aufgebaut sein:

```
<!-- CD/examples/ch08/ant/AntDemo/build.xml -->
<project default="build" basedir=".">
```

Als Standard-Target ist für dieses Projekt `build` definiert (und nicht etwa `run`), weil dies auch in Xcode der Standardfall ist und Xcode den gleichnamigen Parameter beim Aufruf des Ant-Buildfiles normalerweise weglässt.

```
  <property environment="env"/>
  <property name="product" value="${env.PRODUCT_NAME}"/>
```

Mit der `environment`-Property werden alle verfügbaren Umgebungsvariablen eingelesen und dem Ant-Buildfile unter der Bezeichnung `env` zur Verfügung gestellt. Der Programmname wird dann anhand der Variablen `PRODUCT_NAME` ermittelt – damit ist das Buildfile an die weiter oben beschriebene Xcode-Kon-

figuration gebunden. Wenn Sie das Buildfile mit einer anderen Entwicklungsumgebung einsetzen, müssen Sie dort dann entweder auch die entsprechende Variable setzen, oder Sie definieren die `product`-Property hier im Buildfile einfach mit einer festen Zeichenkette. Bei der weiteren Ausführung kann auf eine Property dann mit der schon bekannten `{}`-Notation und ihrem Namen zugegriffen werden, hier also `{product}`.

```
<property name="src"      location="."/>
<property name="build"    location="build"/>
<property name="classes"  location="${build}/classes"/>
<property name="jar"      location="${build}/${product}.jar"/>
```

Nun werden einige Verzeichnisse festgelegt. Die Quelltexte (und andere Dateien wie z.B. das JAR-Manifest) müssen direkt im Projektverzeichnis liegen. Das Ergebnis landet im Unterverzeichnis `build`, wie dies bei Xcode-Projekten üblich ist. Die Klassen werden dabei zunächst in das Unterverzeichnis `classes` übersetzt, danach wird daraus ein JAR-Archiv erzeugt.

```
<target name="clean">
  <delete dir="${build}" includeEmptyDirs="true"/>
  <delete dir="${classes}" includeEmptyDirs="true"/>
  <delete file="${jar}"/>
</target>
```

Das `clean`-Target löscht ganz einfach das `build`-Verzeichnis. Damit ist der Build-Prozess zurückgesetzt, und alle Dateien müssen neu übersetzt werden. Die zwei weiteren Löschanweisungen sind eigentlich überflüssig und nur zur Sicherheit vorhanden, falls Sie weiter oben bei den Properties andere Pfade vorgeben. Der Target-Name wird von Xcode als Parameter an das Ant-Buildfile übergeben, wenn in Xcode der Menüpunkt **Build · Clean** angewählt wird.

```
<target name="init">
  <mkdir dir="${build}"/>
  <mkdir dir="${classes}"/>
</target>
```

Das Target `init` wird nur indirekt von anderen Targets aufgerufen und erzeugt alle nötigen Verzeichnisse für den Build-Prozess.

```
<target name="compile" depends="init">
  <javac srcdir="${src}" destdir="${classes}"/>
  <jar jarfile="${jar}" basedir="${classes}"
      manifest="Manifest"/>
</target>
```

Wie schon beim ersten Buildfile, das Sie im vorhergehenden Abschnitt gesehen haben, kümmert sich das `compile`-Target um den Aufruf des Java-Compilers. Hier wird daraus zusätzlich noch ein JAR-Archiv erzeugt, das mit einer passenden Manifest-Datei doppelklickbar gemacht werden kann.

```
<target name="build" depends="compile">
</target>
```

Das `build`-Target, das über das Xcode-Build-Kommando gestartet wird, ist leer – es greift einfach auf das `compile`-Target zurück. Hier können Sie später weitere Anweisungen für die Programmerstellung einfügen, wie dies beispielsweise im Programmpaket-Buildfile in Abschnitt 0 der Fall ist.

```
<target name="run">
  <java jar="{jar}" fork="true"/>
</target>
</project>
```

Zum Schluss finden Sie wieder das `run`-Target zum Ausführen des Java-Programms. Diesmal wird aber keine einzelne Klasse gestartet, sondern das erzeugte JAR-Archiv anhand der Konfiguration im Manifest. In diesem Fall darf die Java-Anwendung nicht mehr in derselben Laufzeitumgebung wie Ant ausgeführt werden, daher muss mit dem `fork`-Attribut explizit eine neue Laufzeitumgebung angefordert werden.

Beachten Sie, dass dieses Target nicht vom `compile`-Target abhängt! Damit können Sie eine alte Programmversion starten, auch wenn zwischenzeitlich schon Änderungen am Quelltext vorgenommen wurden.

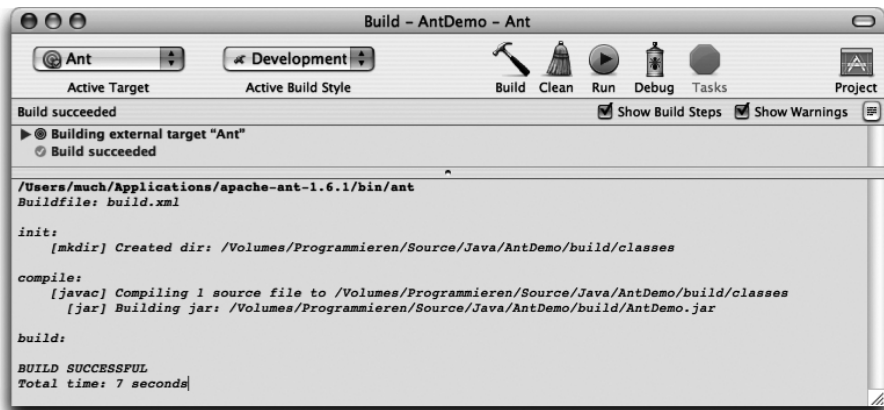


Abbildung 8.2 Ant-Build mit Xcode ausführen

Nun können Sie den Ant-Build-Vorgang in Xcode aufrufen. Öffnen Sie den Dialog **Build · Show Detailed Build Results** und klicken Sie dort das »Build«-Symbol an (siehe Abbildung 8.2). Sie sehen, dass Ant ohne Parameter gestartet wird, das Buildfile `build.xml` aus dem Projektverzeichnis verwendet und darin das Standard-Target `build` ausführt.

Was noch fehlt, ist das Ausführen des gerade übersetzten Programmcodes durch Ant. Xcode erzeugt für jedes Java-Projekt ein »java«-Executable, das den Code aber direkt mit dem `java`-Kommando startet – das funktioniert zwar auch in diesem Fall prima, aber damit werden eventuelle Besonderheiten des Ant-run-Targets ignoriert. Rufen Sie daher durch `[Ctrl]`-Klick auf »Executables« in der »Groups & Files«-Liste den Dialog **Add · New Custom Executable** auf. Darin vergeben Sie als Executable-Namen beispielsweise »ant« und tragen als Pfad `/bin/sh` ein – Xcode kann nämlich das `ant`-Skript nicht selbst ausführen, sondern muss dafür eine Shell bemühen.

Wählen Sie nun das neue Executable an. Auf der Konfigurationsseite können Sie dann bei »Arguments« den absoluten Pfad für das `ant`-Kommando eintragen, gefolgt vom Ant-Targetnamen `run` (siehe Abbildung 8.3).

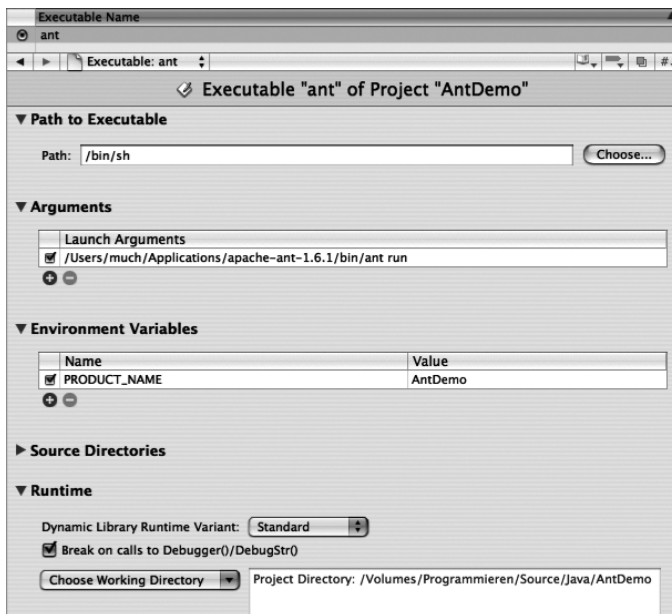


Abbildung 8.3 Xcode-Executable für Ant konfigurieren

Bei »Environment Variables« definieren Sie wieder die Variable `PRODUCT_NAME`, damit das Buildfile auch beim Ausführen den korrekten Programmnamen kennt. Schließlich müssen Sie noch bei »Runtime« im Pop-up-Menü »Choose

Working Directory« das »Project Directory« auswählen, damit Ant das Buildfile findet. Fertig! Jetzt können Sie im »Show Detailed Build Results«-Dialog das »Run«-Symbol anklicken: Ant wird aufgerufen und startet Ihr Java-Programm.

8.1.2 Ant in Xcode 1.5

In der Version 1.5 bringen die Xcode Tools spezielle Ant-Typen für Java-Projekte mit (siehe Abbildung 8.4), außerdem wird Ant 1.6.1 nun im Pfad `/Developer/Java/Ant/` installiert. Die Grundlagen aus dem vorangegangenen Abschnitt sind weiterhin gültig, und zum Erzeugen von Programmpaketen benötigen Sie auch mit Xcode 1.5 eigene Buildfiles (siehe Abschnitt 8.1.4).



Abbildung 8.4 Ant-Projekttypen in Xcode 1.5

Folgende Ant-Projekttypen stehen zur Verfügung:

► **Ant-based Empty Project**

Dieser Typ legt ein komplett leeres Projekt an, das ein »Custom Build Command« besitzt, wie es im vorangegangenen Abschnitt beschrieben wurde. Das Buildfile müssen Sie selbst schreiben.

► **Ant-based Java Library**

Hiermit erzeugte Projekte bringen ein passendes `build.xml` zum Erzeugen eines JAR-Archivs mit.

► **Ant-based Application Jar**

Dieser Typ ist ähnlich wie der voranstehende Library-Typ. Allerdings wird zusätzlich noch ein Manifest zum JAR-Archiv hinzugefügt, damit dieses per Doppelklick startbar ist.

8.1.3 Ant in Eclipse

In vielen anderen Entwicklungsumgebungen ist Ant fest integriert, beispielsweise in Eclipse. Dort können Sie zu dem Projekt über das Menü **File · New · File** eine Datei `build.xml` hinzufügen – Eclipse erkennt die Datei automatisch als Ant-Buildfile und zeigt als Dateisymbol eine kleine Ameise (»Ant«) an.

Wenn Sie das Buildfile geschrieben (oder kopiert und angepasst) haben, können Sie es dann über den Menüpunkt **Run · External Tools · Run As · Ant Build** ausführen lassen. Dies ist auch über das Kontext-Popup-Menü an den diversen Stellen möglich, wo das Buildfile angezeigt wird (z.B. in der »Package Explorer«- oder »Outline«-View).

8.1.4 Programmpakete mit Ant erzeugen

```
<!-- CD/examples/ch08/ant/AntBundle/build.xml -->
<project default="build" basedir=".">
```

Ein Ant-Buildfile kann auch dazu eingesetzt werden, um direkt nach dem Übersetzen des Java-Codes daraus ein Mac OS X-Programmpaket (Bundle) zu erzeugen, wie es in Kapitel 4, *Ausführbare Programme*, beschrieben wurde. Das hier vorgestellte Buildfile ist speziell an Xcode angepasst, aber mit kleinen Änderungen kann es auch zusammen mit anderen Entwicklungsumgebungen verwendet werden – damit können Sie dann Mac OS X-Programmpakete auch auf anderen Betriebssystemen generieren. Die Grundstruktur des Ant-Projekts ist dieselbe wie bei dem Buildfile, das Sie weiter oben schon gesehen haben. Im Folgenden sind die wichtigen Änderungen und Erweiterungen beschrieben.

```
  <property name="bundle"    location="${build}/${product}.app"/>
  <property name="contents"  location="${bundle}/Contents"/>
  <property name="macos"     location="${contents}/MacOS"/>
  <property name="stub"
    location="${macos}/JavaApplicationStub"/>
```

Zusätzlich zu den Properties für die Quelltexte und das JAR-Archiv werden zunächst Properties für die Bundle-Struktur festgelegt. `bundle` bezeichnet den Namen des obersten Programmpaket-Verzeichnisses, `stub` verweist auf den nativen Programmstarter in einem der Unterverzeichnisse.

Die Targets `clean`, `init` und `compile` funktionieren im Wesentlichen wie gehabt. Das `build`-Target enthält nun aber die Anweisungen, mit denen das Programmpaket zusammengestellt wird. Die Abfolge stimmt dabei mit den Schritten aus dem Abschnitt »Programmpakete von Hand erzeugen« in Kapitel 4, *Ausführbare Programme*, überein:

```

<target name="build" depends="compile">
  <property name="resources" location="\${contents}/Resources"/>
  <property name="java"      location="\${resources}/Java"/>
  <mkdir dir="\${macos}"/>
  <mkdir dir="\${java}"/>

```

Hier werden noch ein paar lokale Properties für Verzeichnisse definiert, die nur innerhalb dieses Targets benötigt werden. Anschließend werden damit alle erforderlichen Verzeichnisse für das Programmpaket angelegt.

```

  <copy todir="\${macos}"
        file="/System/Library/Frameworks/JavaVM.framework/
              Versions/Current/Resources/MacOS/
              JavaApplicationStub"/>
  <copy todir="\${contents}" file="Info.plist">
    <filterset>
      <filter token="PRODUCT" value="\${product}"/>
    </filterset>
  </copy>
  <echo file="\${contents}/PkgInfo" append="false"
        message="APPL????"/>
  <copy todir="\${resources}" file="\${product}.icns"/>
  <copy todir="\${java}" file="\${jar}"/>

```

Nun werden alle Dateien in die passenden Verzeichnisse kopiert. Der Programmstarter `JavaApplicationStub` wird dabei aus einem System-Verzeichnis genommen. Wenn Sie dieses Ant-Buildfile auf einem anderen Betriebssystem einsetzen, müssen Sie sich die Datei natürlich von woanders besorgen, beispielsweise aus dem Programmpaket `CD/utility/Minimal.app`.

Die Konfigurationsdatei `Info.plist` wird an dieser Stelle nicht nur kopiert, durch das `filterset` werden auch alle Vorkommen von `@PRODUCT@` im XML-Quelltext durch den Programmnamen – hier im Beispiel »AntBundle« – ersetzt. Betroffen von der Textersetzung sind die Schlüssel `CFBundleName`, `ClassPath` und `MainClass`. Falls Sie hier andere (feste) Zeichenketten wünschen, müssen Sie `Info.plist` in der Entwicklungsumgebung manuell anpassen. Die weiteren Stellen, die Sie typischerweise ändern sollten, sind mit der Zeichenkette »TODO« markiert. Als Erweiterung des Ant-Buildfiles könnten Sie auch für diese Zeichenketten geeignete Filter und Properties definieren.

Das Programmsymbol wird vom Buildfile immer in einer Datei mit dem Programmnamen und der Dateinamenserweiterung `.icns` erwartet, hier also in `AntBundle.icns`. Sie können den Namen beliebig ändern, müssen dann aber

das Buildfile und die Datei `Info.plist` (beim Schlüssel `CFBundleIconFile`) anpassen. Wenn Sie für die Entwicklung auf anderen Betriebssystemen das Java-Standard-Programmsymbol verwenden möchten, können Sie die Datei `GenericJavaApp.icns` aus dem Programmpaket `Minimal.app` von der Buch-CD einsetzen. Alternativ benutzen Sie einfach irgendeine Bild-Datei in einem der gebräuchlichen Formate (z.B. GIF, JPEG oder TIFF).

Zum Schluss wird dann mit dem JAR-Archiv der eigentliche Java-Programmcode in das Programmpaket kopiert. Falls Ihre Anwendung aus mehreren Archiven besteht, müssen Sie diese Zeile im Buildfile entsprechend anpassen.

```
<chmod file="${stub}" perm="755"/>
</target>
```

Schließlich muss noch der Programmstarter mit dem `chmod`-Task ausführbar gemacht werden, da die Informationen über das x-Bit beim Kopieren der Datei mit dem `copy`-Task verloren gehen. Damit ist das Mac OS X-Programmpaket nun fertig!

```
<target name="run">
  <exec executable="${stub}" dir="${build}"/>
</target>
```

Beim `run`-Target wird nun nicht mehr direkt `java` gestartet. Stattdessen wird mit dem `exec`-Task der native Programmstarter aufgerufen, der anhand der Bundle-Konfiguration die Laufzeitumgebung erzeugt und den Java-Code ausführt.

Eine andere Möglichkeit zum Erzeugen von Programmpaketen mit Ant stellt der »Jar Bundler Ant Task« dar. Wenn Sie sich das Archiv von der Seite <http://www.loomcom.com/jarbundler/> herunterladen und installieren, können Sie ein Programmpaket dann mit einem einzigen Aufruf des `jarbundler`-Tasks generieren lassen. Dieser Task stellt auch automatisch die Datei `Info.plist` zusammen, was Sie mit diversen Attributen im Ant-Buildfile beeinflussen können.

8.1.5 Maven

Maven ist eine Art Nachfolger für Ant und baut darauf (sowie auf weiteren Komponenten wie der Skriptsprache Jelly) auf. Während Sie bei Ant im Buildfile genau beschreiben müssen, wie der Buildprozess Schritt für Schritt abläuft und welche Programme (Tasks) dabei aufgerufen werden, sagen Sie bei Maven nur noch, was das Ziel ist und welche Ressourcen dafür zur Verfügung stehen –

passende Maven-Plugins wissen dann automatisch, welche Werkzeuge in welcher Reihenfolge aufgerufen werden müssen. Entsprechend verwendet Maven kein Buildfile `build.xml`, sondern eine Projekt-Konfigurationsdatei `project.xml`:

```
<!-- CD/examples/ch08/maven/project.xml -->
<project>
  <build>
    <sourceDirectory>.</sourceDirectory>
  </build>
</project>
```

Listing 8.2 Einfache Maven-Projektdatei

Sie legen hier also nur fest, wo sich Ihre Quelltexte befinden, nicht aber, was damit passieren soll. Der Aufruf erfolgt dann mit `maven java:compile` – das `java`-Plugin soll alle in `project.xml` definierten Quelltexte kompilieren (`compile` ist ein so genanntes »Goal«).

Sie können Maven kostenlos von <http://maven.apache.org/> herunterladen und das ausgepackte Archiv irgendwo installieren (im folgenden Beispiel geschieht dies im Benutzerverzeichnis). Zur Installation müssen Sie wie bei Ant einige Umgebungsvariablen setzen und eine einmalige Initialisierung durchführen:

```
setenv PATH ~/maven-1.0/bin:$PATH
setenv MAVEN_HOME ~/maven-1.0
chmod a+rx $MAVEN_HOME/bin/*
$MAVEN_HOME/bin/install_repo.sh $HOME/.maven/repository
```

Wenn alles korrekt ist, sollten Sie beim Aufruf mit der Option `--version` (oder `-v`) folgende Ausgabe sehen:

```
[straylight:~] much% maven --version

  _ _ _
 |  \ /  |  _ _ _Apache_  _ _
 |  | \ | /  _ ` \ V / -_) ' \ ~ intelligent projects ~
 | _ |  _ \ _ _ | \ / \ _ _ | _ | _ | v. 1.0
```

Eine Übersicht über alle vorinstallierten Plugins und Goals erhalten Sie mit `maven -g`. Der tiefere Einstieg in Maven ist auf <http://maven.apache.org/start/> ausführlicher beschrieben. Die Integration in Xcode erfolgt analog zu Ant.

8.2 JUnit

Spätestens nachdem Sie Ihre Software fertig gestellt haben, müssen Sie die Anwendung funktional und integrativ testen. Häufig werden dafür Tests geschrieben, die in der `main()`-Methode die nötigen Aufrufe durchführen und die Ergebnisse mit `System.out.println()` ausgeben. Leider lässt sich dies schlecht automatisieren und in den Software-Prozess integrieren – die Tests müssen also von Hand gestartet und ausgewertet werden, was schon bei mittelgroßen Projekten kaum realistisch ist. Daher wurde »JUnit« für so genannte Unit-Tests entwickelt, das zum einen mit einer grafischen Oberfläche benutzt und zum anderen als Kommandozeilentool in ein Projekt integriert werden kann. Die Unit-Tests werden beim Extreme Programming (XP) benötigt, das den »Test First«-Ansatz verfolgt (zuerst den Test schreiben, erst danach den eigentlichen Programmcode), sie können aber auch bei anderen Entwicklungsprozessen eingesetzt werden. JUnit ist kostenlos und Open Source.

Bei JUnit werden die Tests eines Testfalls in Unterklassen von `junit.framework.TestCase` programmiert. Dabei werden alle Methoden, die mit »test« beginnen, die `public` sind und die keine Parameter erwarten, automatisch als Testroutinen ausgeführt. Mehrere Testfälle können zu einer `junit.framework.TestSuite` zusammengefasst werden. Ein (unvollständiger) Testfall für die Standardklasse `ArrayList` zeigt Ihnen die wichtigsten Konzepte von JUnit:

```
//CD/examples/ch08/junit/EinKleinerJUnitTest.java
import junit.framework.*;
import java.util.*;

public class EinKleinerJUnitTest extends TestCase {
    private ArrayList liste;

    public EinKleinerJUnitTest(String name) {
        super(name);
    }

    protected void setUp() {
        liste = new ArrayList();
    }

    protected void tearDown() {
        liste.clear();
    }
}
```

```

public void testLeereListe() {
    assertTrue( liste.isEmpty() );
}

public void testUnerwarteteException() {
    Object o = liste.get(0);
}

public void testErwarteteException() {
    try {
        Object o = liste.get(0);
        fail("Sollte eine IndexOutOfBoundsException werfen");
    }
    catch(IndexOutOfBoundsException e) { }
}

public static Test suite() {
    return new TestSuite( EinKleinerJUnitTest.class );
}

public static void main(String args[]) {
    junit.textui.TestRunner.run( suite() );
}
}

```

Listing 8.3 Ein kleiner JUnit-Testfall

Damit JUnit Testobjekte erzeugen kann, muss die Klasse einen Konstruktor ohne oder mit einem `String`-Parameter besitzen. Innerhalb der von Ihnen programmierten Test-Methoden können Sie dann mit den von `TestCase` geerbten `assertXXX()`-Methoden bestimmte Sachverhalte prüfen, die bei Misserfolg einen Testfehler erzeugen. Hier wird in `testLeereListe()` mit `assertTrue()` sichergestellt, dass eine `ArrayList` direkt nach der Erzeugung wirklich leer ist.

Nicht abgefangene Exceptions führen zu Fehlern beim Ausführen des Tests. Beispielsweise wird in der Methode `testUnerwarteteException()` das Auslesen vom nicht vorhandenen Listenelement mit dem Index 0 eine `IndexOutOfBoundsException` hervorrufen. Wenn Sie genau wissen, dass eine bestimmte Anweisung eine Exception wirft, müssen Sie den Test anders formulieren: Fangen Sie die Exception selber ab, und lassen Sie den Test nur in dem

Fall, dass die Exception wider Erwarten *nicht* auftritt, mit `fail()` fehlschlagen. Im Beispiel finden Sie dies in `testErwarteteException()`.

Allgemeine Initialisierungs- und Aufräumarbeiten, die bei jedem Test benötigt werden, können Sie in die Methoden `setUp()` und `tearDown()` verlagern, die von JUnit automatisch vor bzw. nach jeder Test-Methode aufgerufen werden. Seiteneffekte zwischen den einzelnen Tests gibt es dadurch nicht, denn für jeden Test wird ein eigenes Testobjekt erzeugt.

Schließlich wird von der statischen Methode `suite()` eine `TestSuite` erzeugt, die von einem »TestRunner« ausgeführt und ausgewertet werden kann. JUnit bringt einige solcher TestRunner mit, und in `main()` wird derjenige mit der Textoberfläche gestartet. Damit kann der Test dann bequem im Terminal gestartet oder in ein Projekt integriert werden.

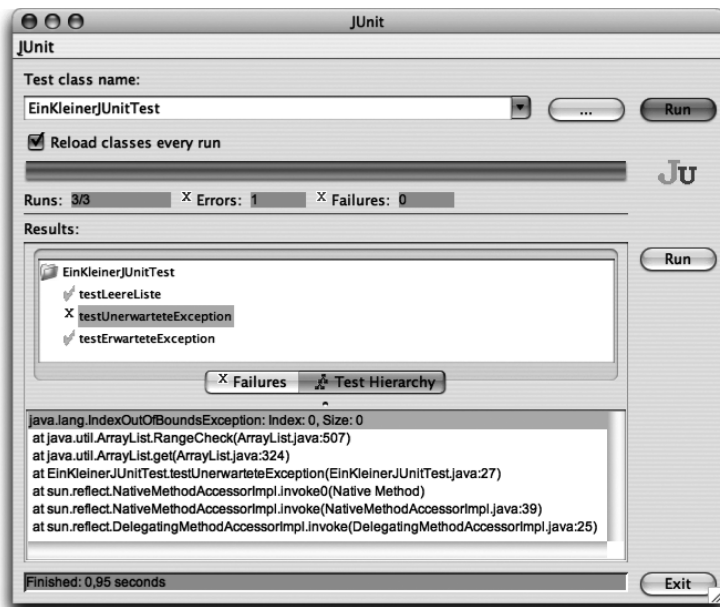


Abbildung 8.5 JUnit 3.8.1

Die Installation von JUnit ist denkbar einfach. Laden Sie sich das aktuelle Archiv von <http://www.junit.org/> herunter, packen Sie es aus und legen Sie den Archivinhalt irgendwo ab – hier beispielsweise im Verzeichnis `~/junit3.8.1/`. Der Aufruf des grafischen TestRunners (siehe Abbildung 8.5) erfolgt dann mit folgendem Kommando:

```
java -cp ./Users/much/junit3.8.1/junit.jar
      junit.swingui.TestRunner EinKleinerJUnitTest
```

Wenn Sie den letzten Parameter, die Klasse mit der `suite()`-Methode, nicht angeben, können Sie die Klasse nachträglich im TestRunner-Dialog auswählen.

Damit Sie das JUnit-Archiv nicht bei jedem Übersetzen und Ausführen von Testfällen angeben müssen, können Sie es auf den systemweiten Klassenpfad setzen:

```
setenv CLASSPATH ./Users/much/junit3.8.1/junit.jar
```

Einfacher und besser ist es aber, wenn Sie das Archiv `junit.jar` in eines der Java-Erweiterungsverzeichnisse kopieren, beispielsweise nach `/Library/Java/Extensions/`. In beiden Fällen können Sie den Test dann wie folgt starten:

```
java EinKleinerJUnitTest
```

Das JUnit-Archiv wird hierbei automatisch gefunden, und die `main()`-Methode startet den TestRunner. Mit einem solchen Aufruf ist JUnit dann auch in Xcode integrierbar.

8.2.1 JUnit in Xcode

Die folgende Beschreibung setzt voraus, dass `junit.jar` in einem der `Extensions`-Verzeichnisse installiert ist. Ansonsten müssen Sie das Archiv bei jedem Projekt in den Target-Einstellungen unter **Settings · Simple View · Search Paths · Java Classes** hinzufügen.

Neben dem Programmcode schreiben Sie die Klassen für die Testfälle, die Sie am besten in eine eigene Gruppe einordnen (siehe Abbildung 8.6). Die Gruppe erzeugen Sie mit einem `[Ctrl]`-Klick in den Quelltexte-Bereich von »Groups & Files«. Im Kontext-Popup-Menü können Sie dann **Add · New Group** anwählen. Das Beispiel `CD/examples/ch08/junit/FesteZeichenkette/` besitzt nur einen einzigen Testfall, mehrere können Sie im Testcode mit der `TestSuite`-Methode `addTestSuite()` zusammenfassen.

Die Testfälle werden bei der normalen Programmausführung nicht berücksichtigt, da das Projekt-Executable »java« nicht die `main()`-Methode des Tests ausführt, sondern diejenige in der Hauptklasse der Anwendung. Daher erzeugen Sie nun ein spezielles Executable zum Testen. Führen Sie dazu einen `[Ctrl]`-Klick auf »Executables« in der »Groups & Files«-Liste aus und rufen Sie im Kontext-Menü **Add · New Custom Executable** auf. Im erscheinenden Dialog tragen Sie als Namen »JUnit« und als Pfad `/usr/bin/java` ein.

Wählen Sie nun in der »Executables«-Liste das gerade erzeugte »JUnit« aus. Rechts in den Einstellungen legen Sie bei »Launch Arguments« mit der Option

-classpath das Programmarchiv fest. Dahinter geben Sie die Klasse an, die in der `main()`-Methode den `TestRunner` ausführt. Schließlich müssen Sie noch sicherstellen, dass der Radio-Button oben in der »Executable Name«-Liste aktiv ist – fertig. Nun können Sie den JUnit-Test mit **Build · Build and Run** starten.

Anstelle der Textoberfläche können Sie natürlich auch jedes Mal die grafische JUnit-Oberfläche starten lassen. Dazu tauschen Sie im `Test-main()` einfach den `junit.textui.TestRunner` gegen den `junit.swingui.TestRunner` aus.

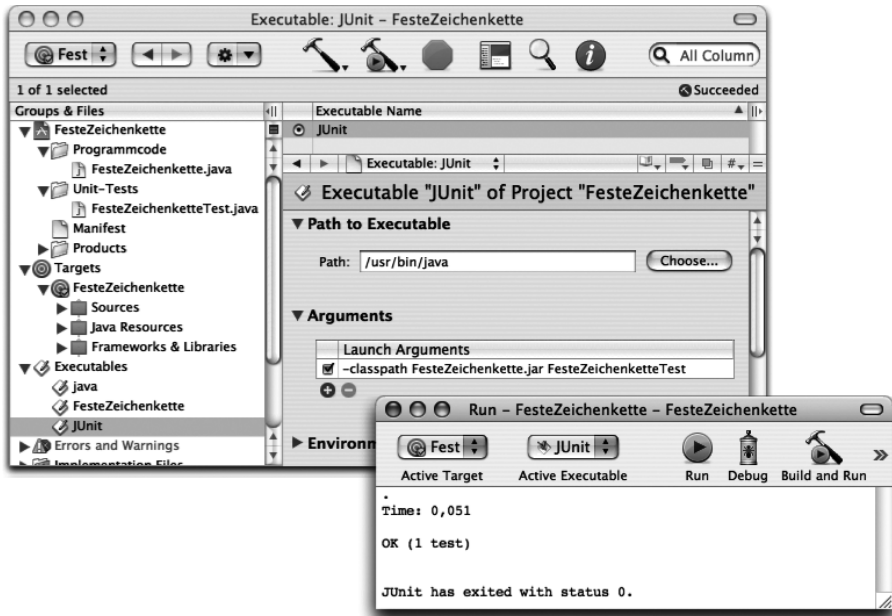


Abbildung 8.6 JUnit-Testfälle in einem Xcode-Projekt

8.2.2 JUnit in Eclipse

Andere Entwicklungsumgebungen haben JUnit fest oder als Plugin integriert, so z.B. Eclipse. Der `TestRunner` läuft dann innerhalb der Entwicklungsumgebung, und Testfälle können teilweise automatisch für bestehende Klassen generiert werden.

In Eclipse rufen Sie unter **File · New · Other...** einen Assistenten (Wizard) auf, von dem Sie sich unter **Java · JUnit** einen `TestCase` oder eine `TestSuite` erzeugen lassen können. Wenn die Testdatei in einer der Views markiert ist, können Sie den Test mit **Run · Run As · JUnit Test** ausführen lassen. Dies geschieht in einer eigenen JUnit-View, die Sie auch direkt über **Window · Show View · Other... · Java · JUnit** aufrufen können.

8.3 Decompiler

Decompiler dienen dazu, aus übersetztem Bytecode wieder den ursprünglichen Quelltext zu generieren. Dies kann sehr nützlich sein, falls Sie zu einer Klasse den Quelltext nicht besitzen, aber einen Fehler darin finden und beseitigen müssen. Das »Zurückübersetzen« gestaltet sich dabei recht einfach, und je nachdem mit welchen Debug-Informationen die Klassen kompiliert wurden, können sogar die Kommentare restauriert werden.

Am bekanntesten ist vermutlich der Decompiler `jad`, den Sie einfach als Befehl im Terminal aufrufen und dem Sie (neben diversen Optionen, die die Quelltextausgabe beeinflussen) die auszuwertende Klassendatei übergeben. Damit Sie von überall Zugriff auf das Programm haben, ohne irgendwelche Umgebungsvariablen kopieren zu müssen, verschieben Sie die Datei `jad` einfach in ein geeignetes Verzeichnis:

```
sudo mv jad /usr/local/bin
```

Es gibt diverse grafische Oberflächen für `jad`. Auf der Homepage des Decompilers finden Sie eine Übersicht darüber, viele sind aber nur für Windows verfügbar.

Ein anderer Decompiler ist »JCavaj«, der eine grafische Oberfläche gleich mitbringt (siehe Abbildung 8.7; auch wenn dort beim inneren Fenster das Windows-Look & Feel zu sehen ist, läuft die Anwendung unter Mac OS X). Die Verwendung von JCavaj ist denkbar einfach – es handelt sich um ein JAR-Archiv, das Sie per Doppelklick starten können. Nachdem Sie dann im »Settings«-Menü den Klassenpfad um die gewünschten Klassen bzw. Archive erweitert haben, können Sie die zu dekompilierende Klasse links in der Liste auswählen. Beachten Sie, dass der zurückübersetzte Quelltext fast mit dem Original übereinstimmt (abgesehen von der Textformatierung) – lokale Variablen haben nicht mehr ihre ursprünglichen Namen, sondern automatisch erzeugte wie `string_0_` usw.

- ▶ `jad` 1.5.8 – kostenlos (für nichtkommerziellen Einsatz)
<http://www.kpdus.com/jad.html>
- ▶ JCavaj 1.0 – kostenlos
<http://www.sureshotsoftware.com/jcavaj/>

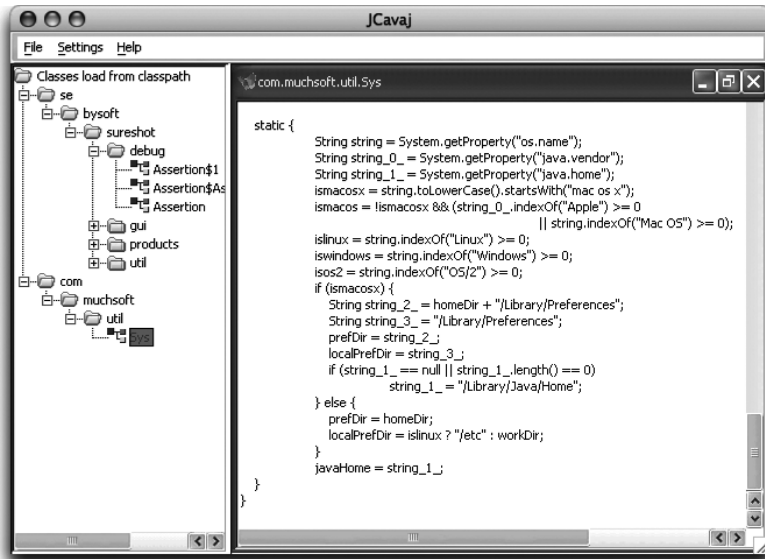


Abbildung 8.7 Jcavaj 1.00

8.4 Obfuscators

Das Erzeugen von Java-Quelltext aus dem Bytecode der Klassendatei mithilfe eines Decompilers ist nicht immer im Sinne des Erfinders bzw. Programmierers – häufig enthält der Code gewisse Algorithmen, die nicht öffentlich bekannt sein sollen. Zunächst einmal gibt es keinen perfekten Schutz, um das Dekompilieren zu verhindern – alles, was Sie an Code zum Kunden ausliefern, kann irgendwie »geknackt« werden. Wollen Sie dies unter allen Umständen verhindern, müssen Sie eine Server-basierte Lösung (z.B. mit Servlets und JSP) einsetzen, die überhaupt keinen Java-Code nach außen schickt.

Eine andere Möglichkeit ist der Einsatz eines Hardware-Kopierschutzes (Dongle). Beispielsweise erlaubt das Produkt »Key4J« (<http://www.wibu.de/key4j/>) die Verschlüsselung von Java-Klassen mithilfe eines USB-Kopierschutzsteckers. Die Java-Anbindung, die mit einer JNI-Bibliothek realisiert ist, steht neben Windows und Linux auch für Mac OS X zur Verfügung.

Das Dekompilieren kann aber durch den Einsatz eines so genannten »Obfuscators« (»Vernebler« oder »Verwirrer«) immerhin erschwert werden. Dabei gibt es zwei Strategien: Ein Teil der verfügbaren Decompiler generiert aus den zu schützenden Klassen illegalen Bytecode, der zwar noch von den meisten Java Virtual Machines ausgeführt, aber nicht mehr von den Decompilern korrekt

entschlüsselt werden kann. Der Nachteil ist offensichtlich – der erzeugte Code läuft nicht mehr überall. Außerdem könnte eine neue Generation von Decompilern durchaus in der Lage sein, den illegalen Bytecode zu entschlüsseln. Daher besteht die bessere Strategie darin, nicht die Dekompilierung zu verhindern, sondern den Quelltext so unverständlich wie möglich zu machen. Dazu erhalten die Klassen, Methoden und Variablen sehr allgemeine Namen wie beispielsweise `a`, `b`, `c`, `aa`, `aaa` – ab einer gewissen Größe ist ein Quelltext mit solchen Bezeichnungen nicht mehr lesbar. Damit öffentliche Schnittstellen erhalten bleiben, können Sie üblicherweise angeben, welche Klassen und Methoden von der Umbenennung ausgenommen sind.

Ein angenehmer Nebeneffekt (manchmal sogar der einzige, für den man einen Obfuscator einsetzt) ist die Optimierung des Codes – die Größe der Klassen und Archive nimmt zum Teil drastisch ab!

Ein kostenloser und dennoch sehr leistungsfähiger Obfuscator ist »ProGuard«. Er wird in der Kommandozeile (oder als Skript in einem Projekt) mit

```
java -jar proguard.jar @meineKonfiguration.pro
```

aufgerufen, wobei der übergebene Parameter eine Konfigurationsdatei mit allen Archiven, Bibliotheken, Umbenennungsausnahmen usw. bezeichnet. Eine grafische Oberfläche können Sie mit dem doppelklickbaren Archiv `lib/proguardui.jar` aufrufen. Damit können Sie dann zum einen die Konfigurationsdateien komfortabel erstellen und speichern – achten Sie aber darauf, dass Sie das automatisch eingetragene Archiv `rt.jar`, das es bei Apples Java-Implementierung nicht gibt, durch `classes.jar` von der gewünschten Java-Version ersetzen (siehe Abbildung 8.8). Zum anderen können Sie die gerade konfigurierte Umwandlung direkt ausführen lassen – was unter Mac OS X allerdings recht gemächlich vonstatten geht, da das Einlesen der Klassenbibliothek einige Zeit benötigt.

Wenn Ihnen die ProGuard-eigenen Konfigurationsdateien nicht zusagen, können Sie ProGuard auch direkt mit Ant verwenden. Das Archiv `proguard.jar` stellt dafür einen speziellen Ant-Task bereit, der auf der ProGuard-Homepage (unter »Manual«) beschrieben ist.

- ▶ ProGuard 2.1 – kostenlos
<http://proguard.sourceforge.net/>

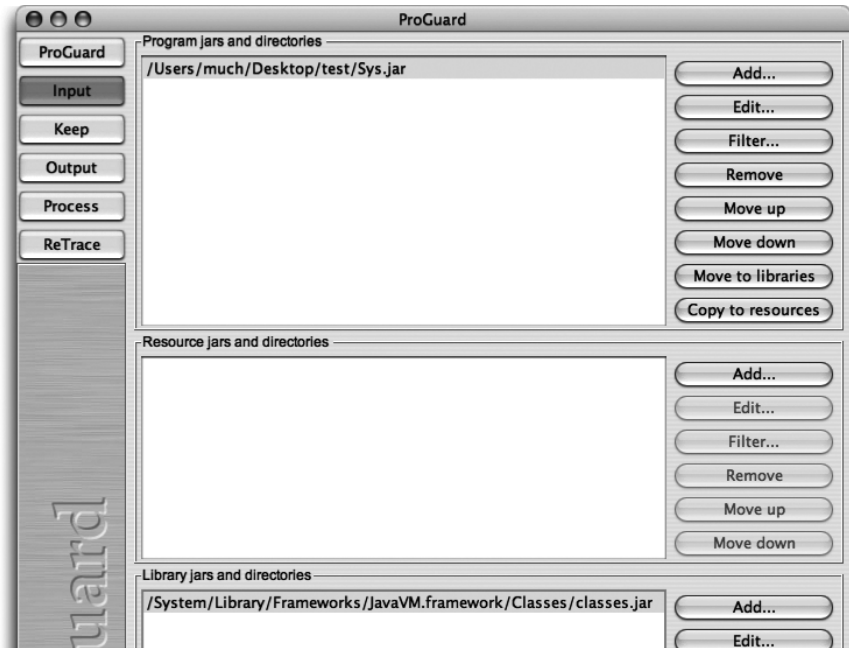


Abbildung 8.8 Standard-Klassenbibliothek in ProGuard konfigurieren

8.5 Bytecode Viewer (Disassembler)

Wenn Sie sich den erzeugten Bytecode einer Klasse ansehen möchten, um beispielsweise Optimierungen abzuschätzen oder um Fehler zu suchen, können Sie den Java-Disassembler `javap` mit der Option `-c` verwenden:

```
[straylight:~] much% javap -c HalloWelt
Compiled from "HalloWelt.java"
public class HalloWelt extends java.lang.Object{
public static void main(java.lang.String[]);
    Code:
        0:  getstatic      #12;
        3:  ldc           #15;
        5:  invokevirtual #20;
        8:  return
public HalloWelt();
    Code:
        0:  aload_0
        1:  invokespecial #26;
        4:  return
}
```

Der Artikel <http://java.sun.com/developer/TechTips/2000/tt0829.html> beschreibt, welche Informationen Sie aus solchen Bytecode-Listings extrahieren können.

Da es recht mühsam ist, den Bytecode im Terminal zu analysieren, gibt es Programme, die Ihnen Klassen und Java-Archive innerhalb einer grafischen Oberfläche disassemblieren. Der vielleicht beste Vertreter dieser Kategorie ist »jclasslib« (siehe Abbildung 8.9) – und die Software ist zudem komplett kostenlos!

Ältere Versionen von jclasslib starten nicht, wenn Sie Java 1.4.2 installiert haben – Sie erhalten dann immer die Fehlermeldung, dass Java 1.4.1 benötigt wird. Dieses Problem tritt auch bei anderer Software auf, wenn im Programmcode bestimmte Verzeichnisse fest kodiert sind. Als kurzfristige Abhilfe können Sie solchen Programmen eine alte Java-Version vorgaukeln, indem Sie im Terminal einen symbolischen Link mit dem alten Namen auf die aktuelle Java-Version anlegen:

```
cd /System/Library/Frameworks/JavaVM.framework/Versions/
sudo ln -s 1.4.2 1.4.1
```

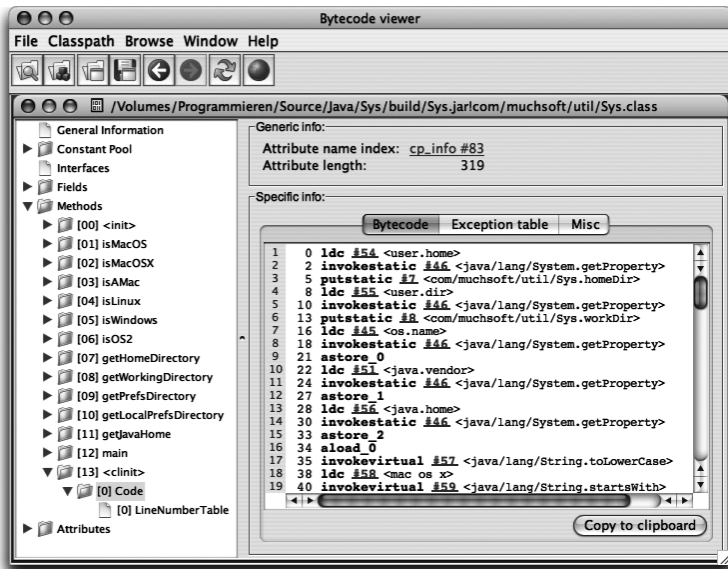


Abbildung 8.9 jclasslib bytecode viewer 2.0

► jclasslib 2.0 – kostenlos

<http://www.ej-technologies.com/products/jclasslib/overview.html>

8.6 Profiler

Die Analyse des statischen Bytecodes kann helfen, um Programmierfehler aufzuspüren. Häufiger wird jedoch das dynamische Laufzeitverhalten untersucht, um Performanzengpässe im realen Einsatz erkennen zu können. Dazu kann man die `java`-Optionen `-Xprof` und `-Xrunhprof` verwenden, aber die manuelle Auswertung der Ergebnisse ist sehr mühsam. Aus diesem Grund kommen beim ernsthaften Profiling eigentlich immer spezialisierte Werkzeuge zum Einsatz, die so genannten »Profiler«. Mit einem Profiler messen Sie typischerweise den Speicherverbrauch der laufenden Anwendung (siehe Abbildung 8.10) und können sich die aktuellen Objekte und Threads (sowie deren CPU-Zeitverbrauch) anzeigen lassen.

Für Mac OS X eignet sich der »JProfiler«, der gut ins System integriert ist. JProfiler bietet gute Filtermöglichkeiten und – für die spätere Auswertung – einen Export der Ergebnisse im HTML-Format. Die doppelklickbare Mac OS X-Applikation `JProfiler` finden Sie nach der Installation übrigens im `bin`-Verzeichnis des Programms.

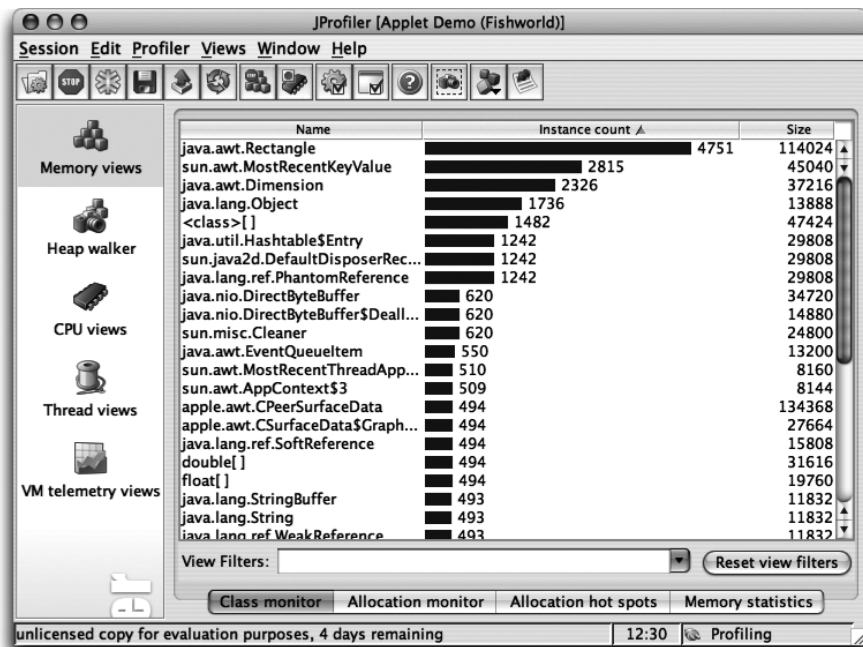


Abbildung 8.10 JProfiler 3.0.1

► JProfiler 3.0 – ab ca. 475 €

<http://www.ej-technologies.com/products/jprofiler/overview.html>

Eine kostenlose Alternative bietet Apple mit »Shark«, das ab der Version 4.0 Profiling auch für Java-Anwendungen unterstützt. Das Programm ist Bestandteil der CHUD-Tools (»Computer Hardware Understanding Developer Tools«), die Sie kostenfrei von [ftp://ftp.apple.com/developer/Tool_Chest/Testing_-_Debugging/Performance_tools/](http://ftp.apple.com/developer/Tool_Chest/Testing_-_Debugging/Performance_tools/) herunterladen können.

Nach der Installation liegt das Programm im Verzeichnis `/Developer/Applications/Performance Tools/` und die Dokumentation in `/Developer/Documentation/CHUD/`. Damit eine ausführliche Auswertung möglich ist, sollten Sie Ihren Java-Code mit der Option `-g` übersetzen, damit der Bytecode die nötigen Debug-Informationen enthält. Anschließend starten Sie das Java-Programm mit den Optionen `-XrunShark` und `-Xnoclassgc` (letzteres aus Sicherheitsgründen, um Abstürzen von Shark vorzubeugen):

```
[straylight:~] much% javac -g PerfTest.java
[straylight:~] much% java -XrunShark -Xnoclassgc PerfTest
2004-07-13 16:23:45.327 java[603] Shark for Java is enabled...
```

Nun können Sie Shark starten. Wählen Sie im Steuerungsfenster eines der »Java ...«-Profile (in Abbildung 8.11 »Java Method Trace«) und den Java-Prozess aus, den Sie untersuchen möchten.

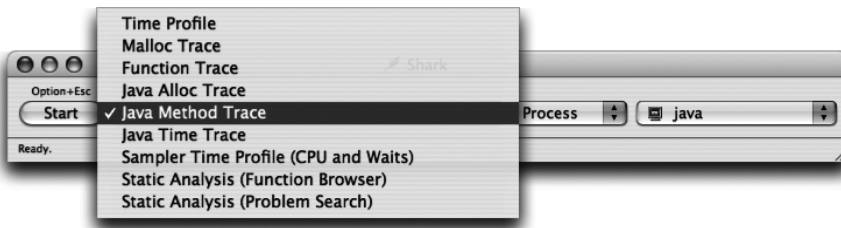


Abbildung 8.11 Konfiguration für ein Shark-Profilung

Mit »Start« beginnen Sie dann das Profiling (und mit demselben Knopf beenden Sie es auch wieder). Die ermittelten Werte kann Shark als Liste oder als Grafik anzeigen. Wenn Sie unter **Preferences** · **Search Paths** · **Source Files** die Verzeichnisse für Ihre Java-Quelltexte eingetragen haben, gelangen Sie innerhalb der Profile-Ergebnisse mit einem Doppelklick auf einen Methodennamen in den Quelltext (siehe Abbildung 8.12). Dort sind die kritischen Stellen farblich hervorgehoben. Wählen Sie unten im Fenster den »Edit«-Knopf an, ruft Shark die entsprechende Stelle zur Bearbeitung in Xcode auf.

Eine genauere Beschreibung von Shark finden Sie bei Apple auf der Seite http://developer.apple.com/tools/shark_optimize.html.

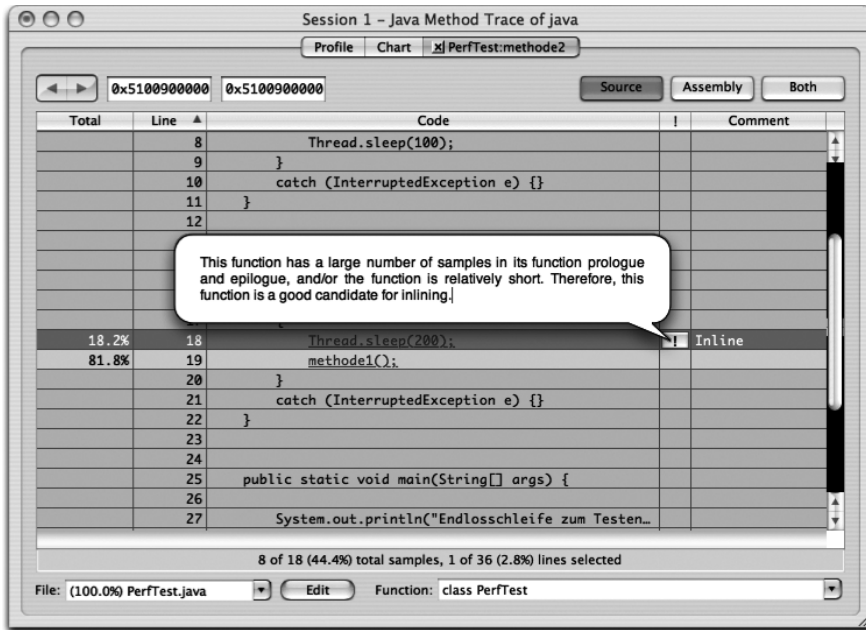


Abbildung 8.12 Quelltextansicht mit Profiling-Ergebnissen

8.7 JavaBrowser

Der JavaBrowser im Verzeichnis /Developer/Applications/Java Tools/ dient zur komfortablen Anzeige der Java-API-Dokumentation. Das Programm durchsucht JAR- und ZIP-Archive sowie einzelne Klassen und listet die enthaltenen Pakete, Klassen und Methoden (mit vollständiger Signatur) auf. Mit den Schaltflächen links unten im Fenster können Sie von der Anzeige der Signaturen auf die Anzeige des Quelltextes bzw. auf die javadoc-Dokumentation (sofern jeweils vorhanden) umschalten (siehe Abbildung 8.13).

Über das Tools-Menü rufen Sie den »Class Finder« auf, der Ihnen anhand eines Suchbegriffs alle in Frage kommenden Klassen, Methode und Attribute auflistet. Mit einem Klick auf einen Listeneintrag erhalten Sie im Hauptfenster eine ausführlichere Beschreibung.

Zunächst ist im JavaBrowser nur die Standard-Klassenbibliothek eingetragen, eigene Bibliotheken können Sie aber sehr einfach hinzufügen – tragen Sie in den entsprechenden Listen in **JavaBrowser · Preferences** einfach Ihre Archive und Dokumentationsverzeichnisse ein (siehe Abbildung 8.14). Wenn Sie nun mit **File · New Default Browser** ein neues Fenster öffnen, können Sie auf Ihren eigenen Code und Quelltext zugreifen.

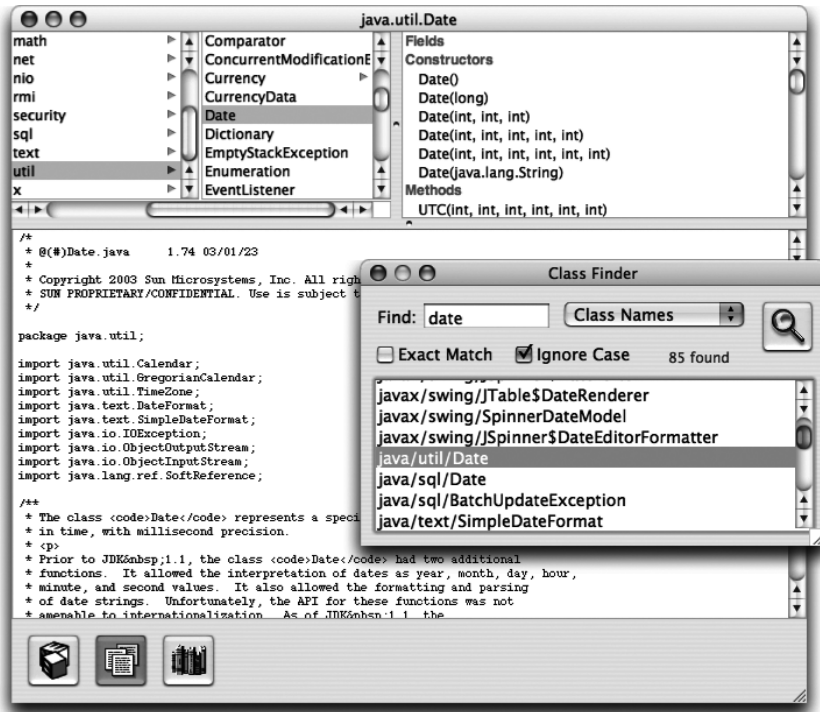


Abbildung 8.13 Der JavaBrowser zur Suche in der API-Dokumentation

Damit der JavaBrowser die javadoc-Dokumentation findet, tragen Sie einfach das Verzeichnis ein, in dem sich die von javadoc generierte Datei `allclasses-noframe.html` befindet. Beim Quelltext kann dies etwas aufwändiger sein, je nachdem, wie Sie Ihre Quelltext-Dateien verwalten: Damit die `*.java`-Dateien gefunden werden, müssen sie sich in einer Verzeichnishierarchie befinden, die den Paketnamen der Klassen entspricht (d.h. wie bei einzelnen Klassendateien). Als Suchpfad geben Sie dann das Verzeichnis außerhalb der Verzeichnishierarchie an (also das Verzeichnis für das anonyme Paket). Natürlich können Sie auch ein Java-Archiv mit den Quelltexten verwenden, aber auch dann muss im Archiv eine passende Verzeichnishierarchie vorhanden sein.

Der JavaBrowser zeigt Ihnen auch die Quelltexte der Standard-Klassenbibliothek an. Dazu greift er auf das Archiv `/Library/Java/Home/src.jar` zurück, in dem sich die von Sun freigegebenen Java-Quelltexte befinden.

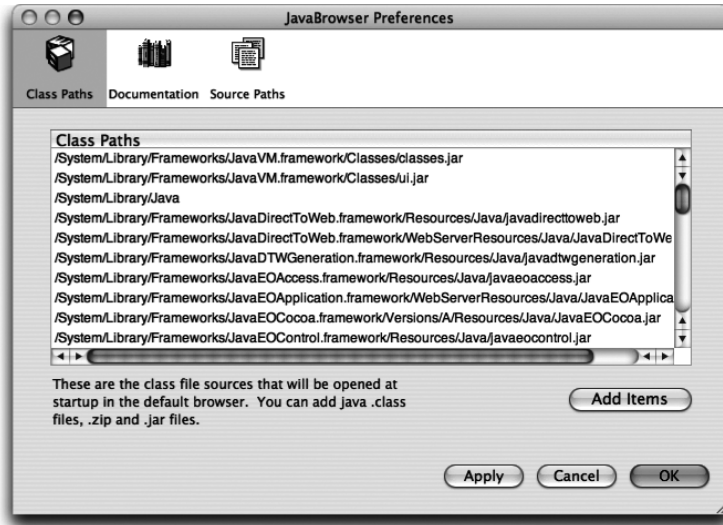


Abbildung 8.14 Eigene Archive zum JavaBrowser hinzufügen

8.8 Jikes

Jikes ist ein alternativer Java-Compiler, der von IBM entwickelt wird. Er bietet folgende Vorteile gegenüber Suns Compiler:

- ▶ Jikes hält sich genauer an die »Java Language Specification« (JLS) als `javac` (!). Ob man diesen Vorteil wirklich nutzen kann, ist eine andere Sache: Der meiste Java-Code verlässt sich vermutlich auf das Verhalten des Sun-Compilers.
- ▶ Er ist deutlich performanter, was gerade bei größeren Projekten auffällt.
- ▶ Es gibt eine Abhängigkeitsanalyse bei der Übersetzung. Dadurch wird inkrementelles Übersetzen möglich, und nebenbei erhalten Sie eine »Makefile«-Funktionalität.
- ▶ Als Programmierer werden Sie durch mehr Warnungen und ausführlichere Fehlermeldungen besser unterstützt.
- ▶ Jikes ist Open Source.

Der Compiler ist bei Mac OS X bereits fertig vorinstalliert, wenn auch nur in der Version 1.18 (aktuell ist Version 1.20). Sie können sich aber jederzeit die neueste Version von <http://ibm.com/developerworks/opensource/jikes> herunterladen – dort finden Sie dann auch Hinweise, wie Sie Jikes unter Mac OS X mit `gcc` selber kompilieren (wenn Sie Apples Entwicklerwerkzeuge installiert haben, läuft das im Wesentlichen darauf hinaus, dass Sie im passenden Verzeichnis `make` aufrufen).

Im Terminal wird der Compiler mit `jikes` aufgerufen – wie `javac`, und er ist auch weitestgehend aufrufkompatibel bezüglich der Optionen (lassen Sie sich ruhig einmal alle Optionen anzeigen, Jikes bietet ein paar zusätzliche Möglichkeiten, die Suns Compiler nicht kennt). Allerdings kennt `jikes` die Java-Klassenbibliothek nicht von sich aus, Sie müssen den passenden Pfad daher beim Aufruf mit der `classpath`-Option spezifizieren:

```
jikes -classpath /System/Library/Frameworks/JavaVM.framework/Classes/classes.jar:. HalloWelt.java
```

Wenn Sie die Klassenbibliothek nicht bei jedem Aufruf mitgeben möchten, können Sie auch die Umgebungsvariable `CLASSPATH` oder `JIKESPATH` setzen. Dies ist auf der Seite <http://www-124.ibm.com/developerworks/oss/jikes/faq/user-index.shtml> genauer beschrieben. In Xcode ist die Verwendung von Jikes deutlich einfacher: Sie müssen nur den verwendeten Compiler umstellen (siehe Abbildung 8.15) – um die Konfiguration des Klassenpfades kümmert sich dann Xcode automatisch.



Abbildung 8.15 Jikes in Xcode als Compiler festlegen

8.9 Groovy

Java gehört zu den streng typisierten Programmiersprachen. Neben der Objektorientierung gehört dies zu den klaren Vorteilen, wenn es um die Entwicklung großer Software-Projekte geht. Wenn man aber nur »mal eben schnell« ein kleines Problem lösen möchte, kann der Aufwand mit Java unnötig hoch sein. Daher gibt es neben den streng typisierten Sprachen noch zahlreiche lose typisierte Skriptsprachen, die oft auch auf ganz bestimmte Probleme spezialisiert sind, beispielsweise Textmanipulation. Meistens kommt dann aber eines zu kurz: die Integration mit Java-Software.

Aus diesem Grund wird derzeit die Skriptsprache »Groovy« entwickelt, die Java nicht ersetzen, sondern ergänzen soll. Groovy übernimmt viele Eigenschaften aus Sprachen wie Python, Ruby, PHP, JavaScript und Smalltalk, darunter dynamische Typisierung, Überladen von Operatoren, Closures (Funktionsabschlüsse), reguläre Ausdrücke, einfache Listebearbeitung und vieles mehr. Für besondere Anwendungsgebiete wie Webseitengenerierung (»Groovlets«) existieren spezielle Bibliotheken. Die Syntax der Sprache ist dabei an Java angelehnt, und natürlich wird auch das ursprüngliche Ziel, die Integration mit Java, erreicht: Sie haben vollen Zugriff auf die Standard-Klassenbibliothek, und die Groovy-Skripte werden vor der Ausführung in ganz normalen Java-Bytecode kompiliert!

Da Groovy komplett in Java programmiert ist, ist auch die Installation kein Problem. Laden Sie sich einfach von <http://groovy.codehaus.org/> das aktuelle Binärarchiv herunter, hier im Beispiel `groovy-1.0-beta-6.zip`. Packen Sie das Archiv aus und kopieren Sie den Inhalt in ein beliebiges Verzeichnis, beispielsweise nach `~/groovy/`. Bevor Sie Groovy nutzen können, müssen Sie noch ein paar Umgebungsvariablen setzen. Wenn Sie `tcsh` als Shell verwenden, ergänzen Sie am Ende der Datei `~/.login` folgende Zeilen (gegebenenfalls müssen Sie die Datei erst noch neu anlegen):

```
setenv PATH ~/groovy/bin:$PATH
setenv JAVA_HOME /Library/Java/Home
setenv GROOVY_HOME ~/groovy
```

Danach machen Sie die Start-Skripte mit `chmod a+rx ~/groovy/bin/*` ausführbar. Im Terminal können Sie nun die Groovy-Shell `groovysh` zum interaktiven Eingeben und Ausführen von Skriptcode aufrufen:

```
[straylight:~] much% groovysh
Lets get Groovy!
=====
Version: 1.0-beta-6 JVM: 1.4.2-34
Type 'exit' to terminate the shell
Type 'help' for command help
1> liste = [ 'a', 'b', 'c' ]
2> print liste[1..2]
3> go
[b, c]
1> liste.each { print(it) }
2> go
abc
```

Alternativ können Sie mit `groovyConsole` eine grafische Konsole aufrufen (siehe Abbildung 8.16). Im unteren Bereich der Konsole können Sie beliebig viele Anweisungen eingeben und dann mit **Actions** • **Run** ausführen lassen.

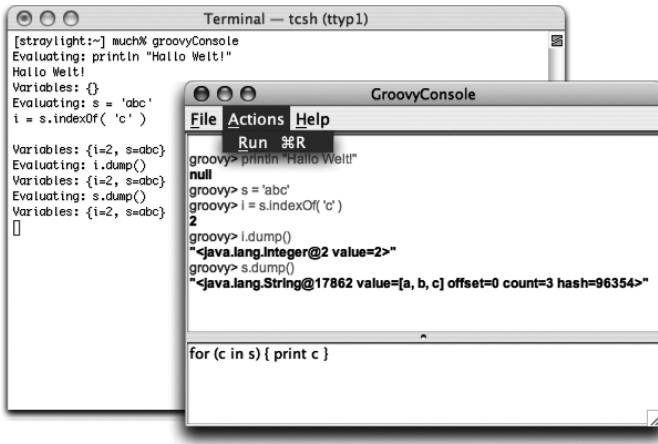


Abbildung 8.16 Die Groovy-Konsole

Schließlich haben Sie noch die Möglichkeit, die Anweisungen in einer Skript-Datei mit der Dateinamenserweiterung `.groovy` abzuspeichern. Im Terminal können Sie diese Skripte dann einfach an das Kommando `groovy` übergeben und ausführen lassen.

Interessant ist, dass Sie diese Skript-Dateien nicht nur direkt ausführen, sondern auch in Bytecode, d.h. in normale Java-Klassendateien, übersetzen lassen können! Dazu verwenden Sie den Groovy-Compiler, den Sie wie folgt aufrufen:

```
groovyc HalloWelt.groovy
```

Als Ergebnis erhalten Sie – wie bei `javac` – eine Datei `HalloWelt.class`. Wenn sich im Skript Anweisungen außerhalb von Funktionen und Klassen befinden, wird eine `main()`-Methode mit diesem Code erzeugt; die Klasse kann dann also mit `java` ausgeführt werden (siehe `CD/examples/ch08/groovy/HalloWelt.groovy`). Da Groovy aber zusätzlich zur Java-Standard-Klassenbibliothek weitere Bibliotheken anbietet, müssen Sie diese bei der Ausführung mit einbinden:

```
java -classpath ./Users/much/groovy/lib/groovy-1.0-beta-6.jar:/Users/much/groovy/lib/asm-1.4.1.jar HalloWelt
```

Über spezielle Java-Klassen können Sie Groovy auch direkt aus Ihrer Java-Anwendung heraus aufrufen. Auf der Seite <http://groovy.codehaus.org/Embedding+Groovy> finden Sie hierzu weitere Informationen.

Eine gute Einführung in die Groovy-Programmierung erhalten Sie auf der Seite <http://www.ociweb.com/jnb/jnbFeb2004.html> (allerdings wie so oft nur auf Englisch).

8.10 UML-Modellierung

Bevor Sie mit dem Programmieren beginnen, sollten Sie Ihr Projekt planen. Bei größeren Projekten ist dafür die Definition eines kompletten Software-Prozesses vorgesehen – inklusive Analyse, Entwurf und Qualitätssicherung. Unabhängig von der Größe des Projekts hat sich seit einigen Jahren zur Planung und zur begleitenden Dokumentation die grafische Modellierungssprache UML (»Unified Modelling Language«) durchgesetzt. Für einfache Anwendungen reicht oft der Entwurf eines Klassendiagramms (siehe Abbildung 8.17), bei komplexerer Software werden Sie vermutlich auch andere UML-Diagrammtypen (Sequenzdiagramm, Kollaborationsdiagramm, Aktivitätsdiagramm usw.) einsetzen.

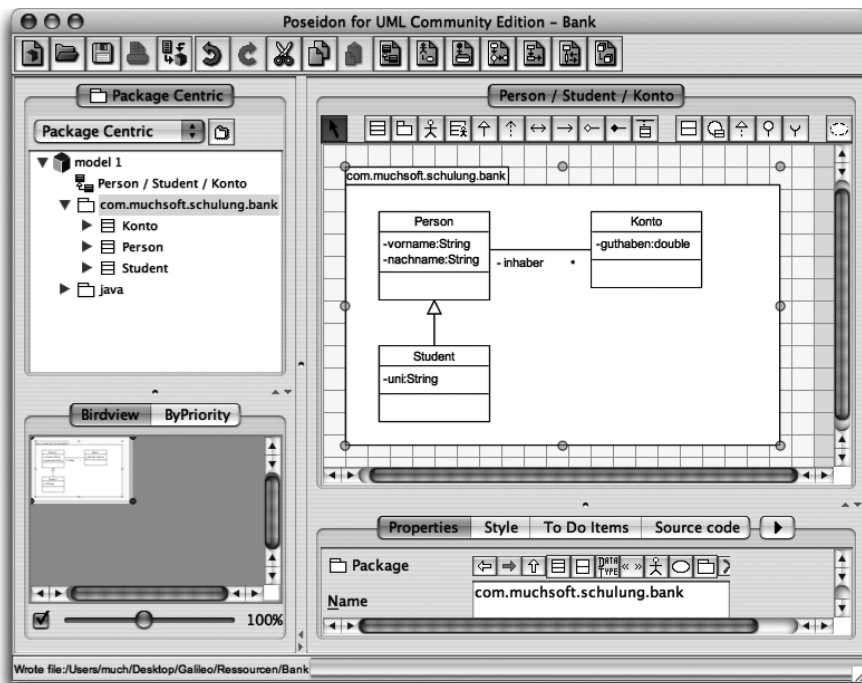


Abbildung 8.17 Poseidon CE 2.5

Obwohl UML prinzipiell komplett von Hand auf einem Blatt Papier anwendbar ist, werden häufig Software-Werkzeuge eingesetzt, die das Erstellen und Verwalten der Diagramme deutlich vereinfachen. Ein bekannter Vertreter dieser

Kategorie ist »Poseidon«, das auch als Mac OS X-Version erhältlich ist. Poseidon baut auf der Open Source-Anwendung »ArgoUML« auf und erweitert dessen Funktionalität.

Poseidon ist in diversen Editionen erhältlich – am besten testen Sie zunächst die kostenlose und auch für kommerzielle Zwecke einsetzbare »Community Edition«, die bereits UML 2.0 unterstützt und den Export der Diagramme in diversen Grafikformaten sowie die Generierung von Java-Code anhand der Diagramme erlaubt. Ab der »Standard Edition« ist dann das Drucken und der Import bestehender Java-Klassen möglich, die »Professional Edition« bietet zusätzlich komplettes Java Round Trip Engineering und den Import von Rational Rose-Dateien.

- ▶ ArgoUML 0.14 – kostenlos
<http://argouml.tigris.org/>
- ▶ Poseidon 2.5 – »Community Edition« kostenlos
<http://www.gentleware.com/>
- ▶ MagicDraw UML 7.8 – »Community Edition« kostenlos
<http://www.magicdraw.com/>
- ▶ VisualParadigm for UML 3.2 – »Community Edition« kostenlos
<http://www.visual-paradigm.com/vpuml.php>

8.11 <oXygen/> XML-Editor

Bei vielen Bereichen der Java-Entwicklung werden Sie XML-Dateien erzeugen oder bearbeiten müssen – sei es, um Tomcat-Webapplikationen zu konfigurieren oder um die Mac OS X-spezifischen »Information Property Lists« mit einem systemübergreifenden Tool zu verändern. Natürlich können Sie all dies mit einem einfachen Texteditor erledigen, denn XML ist normalerweise ein reines Textformat. Mit einem speziellen XML-Editor, der Ihre Eingaben so weit wie möglich auf Korrektheit prüft, haben Sie es aber einfacher.

Es gibt zahlreiche XML-Editoren in allen Preis- und Leistungsstufen. <oXygen/> ist zwar nicht kostenlos, bietet dafür aber viele Funktionen, die Sie bei der täglichen Entwicklungsarbeit unterstützen (Validation, Transformationen mit XSLT und XSL-FO, einen XSLT-Debugger u.a.). Sie können den Editor entweder als Plugin in Eclipse nutzen, oder Sie verwenden <oXygen/> als ganz normale Mac OS X-Applikation. Die Installation ist absolut problemlos, die Einbindung ins System ist gut gelungen.

- ▶ <oXygen/> 4.2 – ca. 96 US-\$
<http://www.oxygenxml.com/>

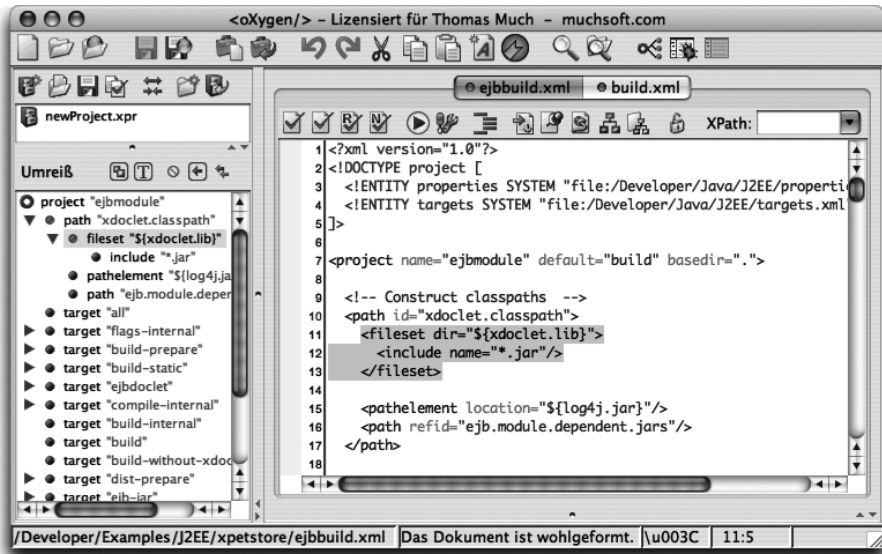


Abbildung 8:18 XML-Bearbeitung und -Validierung mit <oxygen/>

8.12 Literatur & Links

► Ant

- S. Edlich, »Ant kurz & gut«, O'Reilly 2002
Diese Kurzreferenz bietet zunächst einen schnellen Überblick über die Konzepte von Ant und listet anschließend alle eingebauten Tasks mit kurzen Beispielen auf.
- <http://ant.apache.org/manual/>
Das englische Online-Handbuch zu Ant enthält ebenfalls eine vollständige Übersicht aller eingebauten Tasks.

► JUnit

- J. Link, »Unit Tests mit Java«, dpunkt.verlag 2002
Dieses Buch gibt einen umfassenden Überblick über diverse Test-Techniken – mit besonderem Augenmerk natürlich auf Java, die vom Extreme Programming benötigten Unit Tests und damit speziell auf JUnit.

► UML

- Grässle/Baumann, »UML projektorientiert«, 3., aktualisierte Aufl., Galileo Computing 2004
Beschreibt die UML-Modellierung unter den verschiedenen Aspekten eines Geschäftssystems, der Programmierung und des Einsatzes. Die Vorgehensweise ist dabei angenehm praxisnah.

- ▶ M. Hitz/G. Kappel, »UML@Work«, 2. Aufl., dpunkt.verlag 2003
Führt zunächst alle UML-Diagrammtypen ein und setzt diese dann für Analyse und Entwurf ein. Dieses Buch geht etwas theoretischer auf die Hintergründe des Software-Entwurfs mit UML ein.
- ▶ <http://www.uml.org/>
Hier finden Sie alle Spezifikationen rund um UML. Zum Lernen ist aber eines der angegebenen Bücher deutlich besser.

▶ **Java Bytecode**

- ▶ T. Lindholm/F. Yellin, »The Java Virtual Machine Specification«, 2nd ed., Addison-Wesley 1999
Wenn Sie sich für Java-Disassembler oder Optimierungen auf Bytecode-Ebene interessieren, bietet Ihnen dieses Buch einen kompletten Überblick über die Instruktionen und die Funktionsweise der Java Virtual Machine (JVM).
- ▶ <http://java.sun.com/docs/books/vmspec/>
Auf dieser Seite können Sie sich obiges Buch online ansehen oder als HTML-Version herunterladen.

9 Datenbanken und JDBC

9.1	SQL	424
9.2	JDBC	426
9.3	Datenbanken	431
9.4	ODBC	441
9.5	Literatur & Links	442

- 1 **Grundlagen**
- 2 **Entwicklungsumgebungen**
- 3 **Grafische Benutzungsoberflächen (GUI)**
- 4 **Ausführbare Programme**
- 5 **Portable Programmierung**
- 6 **Mac OS X-spezielle Programmierung**
- 7 **Grafik und Multimedia**
- 8 **Werkzeuge**

9 **Datenbanken und JDBC**

- 10 **Servlets und JavaServer Pages (JSP)**
- 11 **J2EE und Enterprise JavaBeans (EJB)**
- 12 **J2ME und MIDP**
- A **Kurzeinführung in die Programmiersprache Java**
- B **Java auf Mac OS 8/9/Classic**
- C **Java 1.5 »Tiger«**
- D **System-Properties**
- E **VM-Optionen**
- F **Xcode- und Project Builder-Einstellungen**
- G **Mac OS X- und Java-Versionen**
- H **Glossar**
- I **Die Buch-CD**

9 Datenbanken und JDBC

»Interpol und Deutsche Bank, FBI und Scotland Yard, Flensburg und das BKA haben unsere Daten da.« (Kraftwerk)

Wenn Ihre Applikationen Daten speichern müssen, beispielsweise Kundenadressen, können Sie diese natürlich in irgendwelchen Textdateien ablegen oder als Binärstrukturen in einer Datei sichern. Sobald Sie es aber mit mehr als einer Hand voll Daten zu tun haben, wird diese selbst gestrickte Verarbeitung schnell zu aufwändig – Sie müssen die Sicherheit der Daten auch unter großer Last gewährleisten, von ausreichender Performanz ganz abgesehen. Hier sind Datenbanken die bessere Lösung, die sich all dieser Probleme annehmen. Viele Datenbanken sind zudem kostenlos verfügbar und dank einer passenden Java-Schnittstelle einfach einzusetzen.

In diesem Kapitel erhalten Datenbank-Einsteiger zunächst eine kurze Einführung in die Datenbanksprache SQL, danach geht es dann um die Verwendung von Datenbanken in Java mittels der JDBC-Schnittstelle. JDBC ist Javas Standard-API für SQL-Datenbankzugriffe. Es gibt zwar auch andere Lösungen wie SQL/J oder JDO (»Java Data Objects«), diese sind aber nicht so verbreitet. JDBC kann auch auf Client-Systemen verwendet werden, kommt aber häufiger auf Server-Systemen innerhalb von Webapplikationen zum Einsatz, wie sie in den folgenden beiden Kapiteln beschrieben werden. Abschließend werden die wichtigsten für Mac OS X verfügbaren Datenbanksysteme vorgestellt. Die Installationen von MySQL und PostgreSQL werden ausführlich besprochen, da es sich bei diesen beiden Produkten um zwei beliebte, kostenlose Datenbanken handelt.

Es werden hier nur so genannte relationale Datenbanken (RelDB) behandelt und keine objektorientierten Datenbanken (OODB). Dies erscheint zunächst vielleicht etwas unerwartet, denn schließlich ist Java ja eine objektorientierte Programmiersprache. In der Praxis ist dies aber der normale Anwendungsfall – relationale Datenbanken sind der Standard, entsprechend sind SQL und JDBC nur für relationale Datenbanken ausgelegt. JDBC stellt damit eine Schnittstelle zwischen der relationalen und der objektorientierten Welt dar.

Für Einsteiger sind noch einige Begriffsklärungen notwendig. Die zu speichernden Daten sind bei relationalen Datenbanken in Tabellen organisiert, die im folgenden Abschnitt über SQL genauer vorgestellt werden. Alle nötigen Tabellen werden zu einer Datenbasis (»Database«, DB) zusammengefasst. Verwaltet wird die Datenbasis (und somit die Tabellen) von einem Datenbank-Ver-

waltungs-System (»Database Management System«, DBMS) oder kürzer Datenbank-System. Da sehr häufig nur mit einer Datenbasis pro Datenbank-System gearbeitet wird, werden die Begriffe Datenbasis, Datenbank, DB und Datenbank-System meist synonym verwendet.

9.1 SQL

SQL (»Structured Query Language«) ist die Standard-Abfrage- und Manipulationssprache für relationale Datenbanken. Egal, wie die Datenbank intern funktioniert, mit SQL stehen Ihnen einheitliche Befehle zur Verfügung, damit Sie nicht jedes Datenbank-System komplett neu erlernen müssen.¹ Zusammenhängende SQL-Befehle werden üblicherweise Anweisung (»Statement«) oder Abfrage (»Query«) genannt.

Im Folgenden wird davon ausgegangen, dass eine Verbindung zur Datenbank besteht, d.h., dass Sie beim Datenbank-System angemeldet sind und eine Datenbank ausgewählt haben. Dies ist teilweise abhängig vom verwendeten Datenbank-System und wird daher ausführlich weiter unten bei der Installation der jeweiligen Produkte beschrieben. JDBC vereinheitlicht dieses Vorgehen, so dass Sie aus Sicht von Java dann wieder kaum Unterschiede bemerken.

Innerhalb der Datenbank arbeiten Sie dann nur noch mit Tabellen, in denen die Daten wie bei einer Tabellenkalkulation organisiert sind. Pro Zeile wird ein Datensatz gespeichert, und jede Spalte hat einen Titel, der den Namen des jeweiligen Datensatz-Feldes angibt. Bei einer Finanzanwendung eines Kreditinstituts könnte es beispielsweise eine Tabelle mit dem Namen »KontoTabelle« geben, in der der Kontoinhaber und der Kontostand vermerkt sind:

ID	Nachname	Vorname	Kontostand
1	Much	Thomas	-50,00
2	Eichel	Hans	1.000,00
3	Duck	Dagobert	123.456.789,10

Tabelle 9.1 Tabelle »KontoTabelle«

Neben den offensichtlichen Feldern »Nachname«, »Vorname« und »Kontostand« gibt es auch die Spalte »ID«. Dies ist eine eindeutige Kennzeichnung für

¹ Zumindest im Idealfall. Natürlich gibt es auch bei SQL Unterschiede bei den diversen Datenbank-Systemen – und sogar eigene Abfragesprachen. Dieses Kapitel konzentriert sich aber auf die Gemeinsamkeiten.

jeden Datensatz, denn es könnte ja durchaus verschiedene Personen geben, die denselben Namen tragen und alleine damit nicht unterscheidbar wären. Die ID-Spalte, die auch andere Namen tragen kann, wird auch als Schlüssel (oder »Primary Key«, Primärschlüssel) bezeichnet. Auf dieser Tabelle können Sie nun SQL-Abfragen durchführen:

```
SELECT *  
FROM KontoTabelle
```

Diese Abfrage zeigt Ihnen einfach alle Felder (angegeben durch das Sternchen) und alle Datensätze der Tabelle an.

```
SELECT ID, Nachname, Vorname  
FROM KontoTabelle  
WHERE Kontostand < 0
```

Hiermit werden Ihnen die bei SELECT angegebenen Felder der Datensätze angezeigt, deren Kontostand negativ ist. Es gibt zahlreiche weitere Optionen, mit denen Sie die Suche verfeinern können. Und natürlich können Sie die Datenbasis auch verändern, wofür UPDATE- und INSERT-Anweisungen zur Verfügung stehen – letzteres ist im Abschnitt 0 bei JDBC beschrieben.

Mächtig werden Datenbanken aber erst durch die Verknüpfungen mehrerer Tabellen, die so genannten Relationen (Beziehungen)². Betrachten Sie folgende zusätzliche Tabelle mit dem Namen »TelefonTabelle«:

ID	Festnetz	Mobil	KontoID
1	0123-567890	0117-9876543	2
2	0987-1234567	0127-12345	1
3	0999-999999		2

Tabelle 9.2 Tabelle »TelefonTabelle«

Hier gibt es wieder ein ID-Feld zur eindeutigen Kennzeichnung der Datensätze. Allerdings gilt diese ID nur innerhalb dieser Tabelle und ist unabhängig von der ID in der Konto-Tabelle. Die Beziehung zwischen beiden Tabellen wird erst durch das Feld »KontoID« hergestellt – hier ist ein ID-Wert aus der Konto-Tabelle eingetragen, wodurch der Telefon-Datensatz dem Konto-Datensatz zugeordnet wird. Während ID wieder der Schlüssel der Telefon-Tabelle ist, nennt man KontoID einen Fremdschlüssel – ein Wert, der in einer fremden

² In der Datenbank-Theorie werden auch die Tabellen selbst Relationen genannt, da sie Beziehungen zwischen Daten herstellen.

Tabelle Schlüssel-Wert ist. Eine verknüpfte Abfrage sieht nun beispielsweise wie folgt aus:

```
SELECT Nachname, Vorname, Festnetz
FROM KontoTabelle k, TelefonTabelle t
WHERE k.ID = t.KontoID
```

Bei `SELECT` geben Sie wieder alle Felder an, die angezeigt werden sollen – wobei Sie nun aus den Feldern aller beteiligten Tabellen auswählen können. Bei `FROM` nennen Sie alle zu verknüpfenden Tabellen und geben ihnen Variablennamen. Mit diesen Variablennamen können Sie nun bei `WHERE` gezielt auf die Felder der einzelnen Tabellen zugreifen und die Werte vergleichen. Als Ausgabe erhalten Sie dann folgende Ergebnistabelle:

```
+-----+-----+-----+
| Nachname | Vorname | Festnetz |
+-----+-----+-----+
| Eichel   | Hans    | 0123-567890 |
| Much     | Thomas  | 0987-1234567 |
| Eichel   | Hans    | 0999-999999 |
+-----+-----+-----+
```

Beachten Sie, dass für Hans Eichel zwei Festnetz-Nummern gefunden werden, für Dagobert Duck dagegen gar keine – genau so, wie es bei den KontoID-Feldern in der Telefon-Tabelle eingetragen ist.

Die gezeigten Beispiele sind bewusst sehr einfach gehalten. Im praktischen Einsatz würde man vermutlich noch eine dritte Tabelle verwenden, in der nur die Personendaten (ohne Kontostand) erfasst sind. Der Vorgang des Aufteilens der Daten auf eine optimale Tabellenanzahl und -struktur wird Normalisierung genannt und ist die hohe (wenn auch erlernbare) Kunst des Datenbankentwurfs. Für den einfachen Datenbankeinsatz kennen Sie nun aber alle nötigen Grundlagen.

9.2 JDBC

Die JDBC-API ist die Java-Standardschnittstelle für den SQL-basierten Datenbankzugriff. Mit JDBC können Sie eine Verbindung zu einer Datenbank aufbauen, SQL-Anweisungen an die Datenbank senden und die Ergebnisse auswerten. Die normalen JDBC-Klassen und (vor allem) -Interfaces finden Sie im Paket `java.sql`, für J2EE und Server-Anwendungen gibt es noch ein paar Erweiterungen im Paket `javax.sql`.

Während JDBC dem Programmierer eine einheitliche Schnittstelle für alle Datenbanken bietet, ist der interne Zugriff auf die Datenbank abhängig vom

jeweiligen Produkt. Daher müssen Sie für jede Datenbank einen passenden JDBC-Treiber installieren, der meistens vom Datenbankhersteller geliefert wird. Auf der Seite <http://servlet.java.sun.com/products/jdbc/drivers> können Sie nach einem bestimmten Treiber bzw. einer bestimmten Datenbank suchen. Bei allen in Abschnitt 0 vorgestellten Datenbank-Systemen sind die Treiber-Webadressen direkt angegeben. Zur Installation eines Treibers müssen Sie das jeweilige ZIP- oder JAR-Archiv nur auf den Klassenpfad setzen – am einfachsten kopieren Sie das Archiv nach `/Library/Java/Extensions/`.

9.2.1 Datensätze lesen

Bevor Sie mit einer Datenbank arbeiten können, müssen Sie eine Verbindung zum Datenbank-System aufbauen und dazu einen geeigneten Treiber laden. Der folgende Code gilt daher unabhängig davon, ob Sie Datensätze lesen oder schreiben wollen:

```
//CD/examples/ch09/JDBCLeSen/JDBCLeSen.java
import java.sql.*;
//...
try {
    Class.forName("com.mysql.jdbc.Driver").newInstance();
}
catch (Exception e) {
    System.err.println("Treiber konnte nicht geladen werden");
    return;
}
```

Mit `Class.forName()` wird versucht, den Treiber dynamisch zu laden. So kann das Programm passend reagieren, falls der Treiber nicht gefunden wurde – beispielsweise könnten Sie dann auf einen anderen Treiber und eine andere Datenbank zurückgreifen. Der als Zeichenkette übergebene Klassenname des Treibers ist vom jeweiligen Treiberhersteller dokumentiert. Das Beispielprogramm verwendet also MySQL, dessen Installation weiter unten beschrieben ist.³

Der Aufruf von `newInstance()` ist bei aktuellen JDBC-Treibern nicht mehr nötig und nur für ältere (nicht ganz korrekt implementierte) Treiber vorhanden.

```
String benutzer = "";
String password = "";
```

³ Beim JDBC-Zugriff auf MySQL finden Sie häufig noch den alten Treibernamen `org.gjt.mm.mysql.Driver`, der aus Kompatibilitätsgründen immer noch genutzt werden kann.

```

Connection con = null;
Statement stmt = null;
ResultSet rs = null;

try {
    con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/test",
        benutzer, passwort);
    con.setReadOnly(true);
    stmt = con.createStatement();

```

Nun wird die Verbindung zum Datenbank-System aufgebaut. Dazu fragen Sie vom `DriverManager` mit `getConnection()` eine `Connection`-Referenz ab, wofür Sie eine spezielle JDBC-Adresse übergeben. Der `jdbc:-`Bestandteil ist festgelegt, danach folgt ein treiberspezifischer Name (hier `mysql`). Danach folgt – wie bei einer URL – die Rechneradresse mit Portangabe. Die oben angegebenen Werte entsprechen den Standardwerten von MySQL, daher könnten Sie hier beide Angaben auch komplett weglassen. Am Ende der Zeichenkette steht der Name der zu verwendenden Datenbank. Bei diesem Beispiel wird mit der allgemein nutzbaren `test`-Datenbank von MySQL gearbeitet, daher werden als Benutzername und Passwort nur leere Zeichenketten übergeben.

Wenn Sie Datensätze nur lesen wollen, ist es dann eine gute Idee, `setReadOnly(true)` aufzurufen, wodurch die Datenbank performanter arbeiten kann. Danach lassen Sie sich mit `createStatement()` ein `Statement`-Objekt erzeugen, mit dem Sie die eigentliche SQL-Anfrage an die Datenbank schicken können.

```

    rs = stmt.executeQuery(
        "SELECT Nachname, Vorname, Kontostand "
        + "FROM KontoTabelle");
    final int NACHNAME = 1;
    final int VORNAME = 2;
    final int KONTOSTAND = 3;

    while ( rs.next() ) {
        System.out.print( rs.getString(NACHNAME) + "\t" );
        System.out.print( rs.getString(VORNAME) + "\t" );
        System.out.println( rs.getDouble(KONTOSTAND));
    }
    rs.close();

```


`executeQuery()` überträgt die SQL-Anweisungen zur Datenbank und erhält als Ergebnis ein `ResultSet`-Objekt, eine Ergebnistabelle – die natürlich auch leer sein kann. Die Felder dieser Ergebnistabelle sind von 1 aufsteigend durchnummeriert, passend zu den Feldnamen, die Sie bei `SELECT` angegeben haben.

Durch das `ResultSet` können Sie sich nun mit `next()` von einem Ergebnisdatensatz zum nächsten durcharbeiten – bis keine weiteren Datensätze mehr vorhanden sind und die Methode `false` zurückliefert. Innerhalb eines Datensatzes fragen Sie dann mit `getString()`, `getDouble()` usw. das Feld mit der angegebenen Nummer ab. Die Abfragemethode sollte dabei ungefähr zum Feld-Datentyp passen, es finden ansonsten aber recht großzügige, automatische Umwandlungen statt.

Wichtig ist, dass es pro `Statement` zu einer Zeit nur ein gültiges `ResultSet` gibt. Sobald Sie über dasselbe `Statement`-Objekt eine neue Abfrage verschickt haben, ist das alte `ResultSet` ungültig – unabhängig davon, ob Sie noch eine Referenz darauf halten. Wenn Sie mehrere `ResultSet`s gleichzeitig benötigen, lassen Sie sich einfach mehrere `Statements` erzeugen.

Zum Schluss ist es wie immer eine gute Idee, die angeforderten Ressourcen explizit freizugeben, damit die Datenbank so performant wie möglich arbeiten kann. Hier wird das `ResultSet` geschlossen, und im folgenden `finally`-Block sind das `Statement` und die `Connection` dran – idealerweise in genau der umgekehrten Reihenfolge, in der die Ressourcen belegt wurden:

```
    }
    catch (SQLException e) {
        e.printStackTrace();
    }
    finally {
        if (stmt != null) {
            try {
                stmt.close();
            }
            catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (con != null) {
            try {
                con.close();
            }
        }
    }
}
```

```

        catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

Beachten Sie, dass die beiden Objekte in unabhängigen try-catch-Blöcken freigegeben werden, denn wenn bei `Statement.close()` ein Fehler auftritt, muss `Connection.close()` unbedingt trotzdem noch aufgerufen werden. Ansonsten würde Ihr Programm bis zu seiner Beendigung eine der oft zahlenmäßig begrenzten Datenbank-Verbindungen belegen, was schnell dazu führt, dass keine weiteren Anwender mit der Datenbank arbeiten können.

9.2.2 Datensätze schreiben

Beim Schreiben von Datensätzen sind das Laden des Treibers und das Freigeben der Ressourcen identisch mit dem vorherigen Beispiel, daher sehen Sie im Folgenden nur die für den Schreibzugriff relevanten Stellen:

```

//CD/examples/ch09/JDBCSchreiben/JDBCSchreiben.java
//...
con = DriverManager.getConnection(
    "jdbc:mysql:///test", benutzer, passwort);
stmt = con.createStatement();
con.setAutoCommit(false);

try {
    con.setReadOnly(false);
    int anz = stmt.executeUpdate(
        "INSERT INTO KontoTabelle "
        + "(Nachname, Vorname, Kontostand) "
        + "VALUES ('Müller', 'Lieschen', 500.00)");
    // weitere zusammengehörige Aktualisierungsanweisungen...
    con.commit();
    System.out.println("Festgeschriebener Datensätze: " + anz);
}
catch (Exception e) {
    con.rollback();
    con.setReadOnly(true);
}

```

Nach dem Verbindungsaufbau mit `getConnection()` – bei dem hier übrigens die Rechneradresse des SQL-Servers fehlt und somit der MySQL-Standardwert `localhost:3306` angenommen wird – wird zunächst wieder ein `Statement`-Objekt für die SQL-Anweisungen erzeugt.

Als Besonderheit wird bei diesem Beispiel zusätzlich die Auto-Commit-Funktionalität ausgeschaltet. JDBC schreibt normalerweise jede einzelne Änderung sofort in der Datenbank fest. Typischerweise möchten Sie aber als Datenbank-Programmierer darüber die Kontrolle haben, damit Sie mehrere zusammengehörige Anweisungen an die Datenbank schicken können, die dann entweder komplett oder gar nicht im Datenbestand festgeschrieben werden. Diese Anweisungsfolge stellt dann eine so genannte **Transaktion** dar. Bei aktiviertem Auto-Commit ist jede einzelne Anweisung eine eigene Transaktion.

Um Änderungen an der Datenbasis vorzunehmen, rufen Sie `executeUpdate()` auf und übergeben die gewünschte SQL-Anweisung. Hier wird mit `INSERT INTO` ein Datensatz in die angegebene Tabelle eingefügt. Dabei geben Sie zunächst alle Namen der Felder an, die Sie ausfüllen möchten, und anschließend hinter `VALUES` die eigentlichen Werte. Achten Sie auf die runden Klammern und darauf, dass Sie Zeichenketten-Werte in einfache Anführungszeichen einschließen.

Nachdem Sie alle Änderungen vorgenommen haben, rufen Sie `commit()` auf, um die Daten festzuschreiben. Wenn hierbei – oder vorher schon während Ihrer Aktualisierungen – ein Fehler auftritt, führen Sie im `catch`-Block `rollback()` aus, wodurch das Datenbank-System alle noch nicht festgeschriebenen Änderungen wieder rückgängig macht.

9.3 Datenbanken

Nachdem Sie nun die plattformübergreifende Verwendung von Datenbanken mittels JDBC kennen, finden Sie im Folgenden eine Liste der wichtigsten für Mac OS X verfügbaren Datenbank-Systeme. Für MySQL und PostgreSQL wird auch die Installation unter Mac OS X ausführlich beschrieben, bei allen weiteren Datenbanken sind zumindest die Download-Adressen für die Datenbank selbst und für den JDBC-Treiber angegeben.

Datenbank-Systeme, die nicht direkt unter Mac OS X betrieben werden können, lassen sich mit einem geeigneten JDBC-Treiber aber trotzdem ansprechen, wenn der Datenbank-Server auf einem anderen System läuft. Sie benötigen dafür einen netzwerkfähigen JDBC-Treiber, einen so genannten Typ-3- oder (besser) Typ-4-Treiber.

9.3.1 MySQL

Die populärste Open Source-Datenbank ist sicherlich MySQL, weil sie relativ kompakt und schnell ist und im Laufe der Jahre Fähigkeiten hinzugewonnen hat, die ihren Einsatz auch in unternehmenskritischen Bereichen interessant machen. Zur Installation laden Sie sich einfach von der Seite http://dev.mysql.com/downloads/mysql/4.1.html#Mac_OS_X je nach Mac OS X-Version das passende »Installer package« in der »Standard«-Version herunter. Die hier verwendete Version 4.1.5 befindet sich zwar noch in der Gamma-Phase (und gilt daher noch nicht als öffentlich verfügbar), ist aber für neue Projekte die ideale Ausgangsplattform. Als stabile Version steht noch MySQL 4.0 zur Verfügung, und wer etwas risikofreudiger ist, kann eine Alpha-Version 5.0 verwenden.

Installieren

Der offizielle Installer ist mittlerweile die einfachste Möglichkeit zur Installation von MySQL unter Mac OS X – daher wird diese Vorgehensweise im Folgenden beschrieben. Wenn Sie MySQL dagegen selbst kompilieren möchten, finden Sie auf der Seite <http://developer.apple.com/internet/opensource/osdb.html> alle nötigen Informationen.

Wenn Sie MacOS X Server einsetzen, müssen Sie sich um die Installation nicht kümmern – MySQL ist bei MacOS X Server vorinstalliert. Zur Verwaltung der Datenbank steht Ihnen dort die Anwendung `/Programme/Dienstprogramme/MySQL Manager` zur Verfügung.

Bevor Sie MySQL installieren, sollte Sie mit dem Werkzeug `/Programme/Dienstprogramme/NetInfo Manager` sicherstellen, dass es einen Benutzer mit dem Namen »mysql« auf Ihrem System gibt. Sie könnten einen solchen Benutzer zwar auch unter **Systemeinstellungen • Benutzer** anlegen, aber alle damit erzeugten Benutzer haben ein sichtbares Verzeichnis im `/Benutzer-Order`, was für einen System-Dienst wie MySQL überhaupt nicht benötigt wird.

Wählen Sie im NetInfo Manager in der ersten Spalte den (normalerweise) einzigen Eintrag »/« aus und in der Spalte rechts daneben den Eintrag »groups«. Wenn darin bereits ein Eintrag »mysql« vorhanden ist, brauchen Sie im Zweifel nichts weiter unternehmen. Ansonsten duplizieren Sie den Eintrag »www« (in der Spalte »groups«) und passen ihn entsprechend der Abbildung 9.1 an. Die Gruppen-ID »gid« sollte dabei eindeutig sein, d.h. von keiner anderen Gruppe verwendet werden. Sobald Sie nun in der Spalte »/« auf den Eintrag »users« klicken, müssen Sie die Änderungen an der Gruppe »mysql« bestätigen.



Abbildung 9.1 Gruppe »mysql« im NetInfo Manager konfigurieren

Unter »users« gibt es eventuell wiederum bereits den Eintrag »mysql«. Falls nicht, duplizieren Sie wieder den Eintrag »www« (diesmal in der Spalte »users«) und passen ihn so an, wie es Abbildung 9.2 zeigt. Die Werte von »uid« und »gid« müssen dabei gleich sein und dem »gid«-Wert der »mysql«-Gruppe entsprechen.

Damit existiert nun ein Benutzer »mysql« auf Ihrem System, dem Sie mit

```
sudo passwd mysql
```

ein sicheres Passwort zuweisen sollten.

Die Konfiguration der NetInfo-Daten lässt sich auch von der Kommandozeile aus mit dem Befehl `niutil` vornehmen. Passen Sie aber auf, was Sie tun – NetInfo ist für die komplette Benutzerverwaltung von MacOS X zuständig, so dass Sie hier beliebig viel kaputt konfigurieren können. Im schlimmsten Fall können Sie sich anschließend nicht mehr an Ihrem System anmelden ...

Die eigentliche MySQL-Installation ist nun unspektakulär. Doppelklicken Sie das Paket `mysql-standard-...pkg` (der Dateiname enthält noch diverse Versionsangaben) und lassen Sie MySQL auf Ihrem Startvolume einrichten. Nachbearbeitungen von Hand sollten nicht mehr nötig sein, da der Installer alle nötigen Skripte aufruft und die Datei- und Datenbankrechte passend setzt. Die Datenbank finden Sie anschließend im Verzeichnis `/usr/local/mysql/`, was wiederum nur ein Alias für einen Ordner mit der kompletten MySQL-Versionsnummer im Verzeichnis `/usr/local/` ist.



Abbildung 9.2 Benutzer »mysql« konfigurieren

Zusätzlich können Sie ein weiteres Installationspaket `MySQLStartupItem.pkg` ausführen lassen, das ein so genanntes »StartupItem« im Verzeichnis `/Library/StartupItems/MySQLCOM/` anlegt. Dadurch kann MySQL automatisch geladen werden, wenn Sie sich am System anmelden. In diesem Kapitel wird das StartupItem einfach nur genutzt – der genaue Aufbau der Dateien ist im folgenden Kapitel bei den Servlets bzw. bei Tomcat beschrieben.

Von früheren MySQL-Versionen wurde das StartupItem im Verzeichnis `/Library/StartupItem/MySQL/` angelegt, was aber zu Problemen mit MacOS X Server führte, da das System dieses Verzeichnis für das vorinstallierte MySQL nutzt.

Ob MySQL dann beim Start tatsächlich geladen wird, hängt von einem Eintrag in der Datei `/etc/hostconfig` ab. MySQL trägt dort die Zeile `MYSQLCOM=-YES-` ein. Um das automatische Starten des Datenbank-Servers zu unterbinden, ändern Sie einfach mit

```
sudo pico /etc/hostconfig
```

den Wert dieses Eintrags auf `-NO-`.

Der Eintrag `MYSQL` wird von Mac OS X Server für das vorinstallierte MySQL verwendet. Eine MySQL-Neuinstallation deaktiviert diesen Eintrag gegebenenfalls, damit nach einem Rechner-Neustart nur ein Datenbank-Server aufgerufen wird. Gestoppt wird ein laufender Server durch die Installation allerdings nicht, dies müssen Sie manuell veranlassen.

Starten und konfigurieren

Wenn Sie das StartupItem installiert haben, ist der Start des Datenbank-Servers einfach:

```
sudo SystemStarter start MySQL
```

Ansonsten rufen Sie die ausführbare Datei direkt auf. Allerdings sollten Sie dann das Programm nach dem Start mit `Ctrl+Z` unterbrechen und mit dem Kommando `bg` weiter im Hintergrund ausführen lassen:

```
[straylight:~] much% sudo /usr/local/mysql/bin/mysqld_safe
```

Password:

```
Starting mysqld daemon with databases from /usr/local/mysql/data  
^Z
```

```
[straylight:~] much% bg
```

Bei Versionen vor MySQL 4.0 wurde der Datenbank-Server noch mit dem Kommando `safe_mysqld` gestartet – wundern Sie sich also nicht, wenn Sie ältere Dokumentationen lesen.

Damit Sie bequem mit den MySQL-Kommandos arbeiten können, können Sie das `bin`-Verzeichnis zur `PATH`-Umgebungsvariablen hinzufügen:

```
setenv PATH /usr/local/mysql/bin:$PATH
```

Aus Sicherheitsgründen sollten Sie nun unbedingt das Passwort des MySQL-»root«-Benutzers mit dem folgenden Kommando setzen:

```
mysqladmin -u root password neues_Passwort
```

Damit ist der Administrator-Zugang der Datenbank abgesichert. Leider verwendet MySQL hier denselben Namen wie das System für seinen »root«-Benutzer. Während letzterer nur zum Starten der Datenbank benötigt wird, ist der MySQL-»root«-Benutzer für alle Verwaltungsaufgaben innerhalb des Datenbank-Systems zuständig.

Testen, arbeiten, beenden

Nachdem Sie die Datenbank gestartet und abgesichert haben, gelangen Sie mit dem Kommando `mysql` in den interaktiven Befehls-Interpreter:

```
[straylight:~] much% mysql test --user=root --password
Enter password:
Reading table information for completion of table and column nam
es
You can turn off this feature to get a quicker startup with -A
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 27 to server version: 4.1.5-gamma-
standard
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> show databases;
+-----+
| Database |
+-----+
| mysql    |
| test     |
+-----+
2 rows in set (0.00 sec)
mysql> exit
Bye
```

Bei der »mysql«-Eingabeaufforderung können Sie entweder spezielle Interpreter-Kommandos eingeben (hier z.B. `exit`) oder ganz normales SQL ausführen lassen. Wenn Sie `mysql` nicht mit dem »root«-Benutzer ausführen, sehen Sie bei `show databases`; nur die allgemein zugängliche Datenbank `test`, ansonsten haben Sie auch Zugriff auf die Verwaltungstabellen von MySQL.

Das Beenden des Datenbank-Servers nehmen Sie am besten mit dem Kommando

```
mysqladmin shutdown --user=root --password
```

vor, oder – wenn Sie das StartupItem installiert haben – alternativ mit

```
sudo SystemStarter stop MySQL
```

Wenn Sie bei der Installation oder der Konfiguration von MySQL Probleme bekommen, bietet Ihnen die Seite <http://www.entropy.ch/software/MacOSx/mysql/> zahlreiche Antworten auf häufig gestellte Fragen.

Das Bearbeiten der Datenbanken und Tabellen mit dem Programm `mysql` ist zwar möglich, aber nicht unbedingt angenehm. Glücklicherweise gibt es zahl-

reiche grafische Oberflächen für MySQL, von denen zwei gelungene Varianten sogar kostenlos sind: »YourSQL« (<http://www.mludi.net/YourSQL/>, siehe Abbildung 9.3) und »CocoaMySQL« (<http://cocoamysql.sourceforge.net/>).

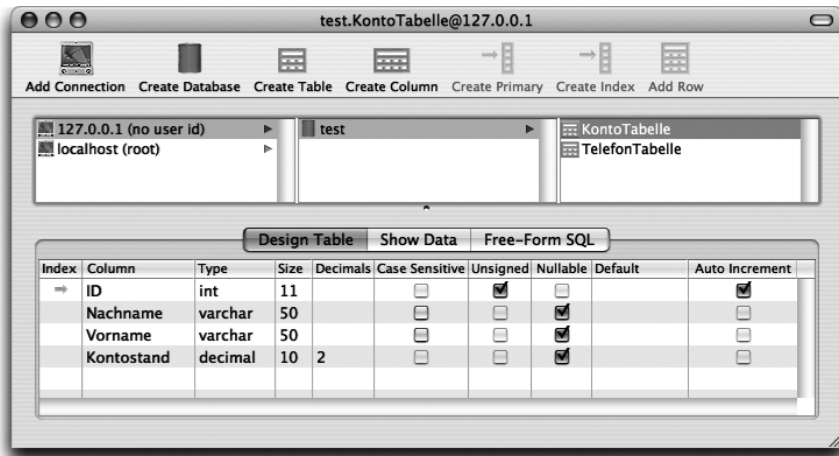


Abbildung 9.3 YourSQL

Wenn Sie die Beispieldaten nicht selbst eingeben möchten, können Sie die Daten auch einfach importieren:

```
mysqlimport test CD/examples/ch09/test.sql
```

Die Daten wurden mit `mysqldump` exportiert und befinden sich auf der Buch-CD im angegebenen Verzeichnis.

JDBC-Treiber

Zur Anbindung von MySQL an Java fehlt nur noch der passende JDBC-Treiber. Ursprünglich wurde der Treiber unabhängig von MySQL entwickelt, die Projektseiten finden Sie noch immer unter der Adresse <http://mmmmysql.sourceforge.net/> im Netz.

Mittlerweile wurde der Treiber vom MySQL-Hersteller aufgekauft und in »Connector/J« umbenannt. Die stabile Version 3.0 erhalten Sie auf der Seite <http://dev.mysql.com/downloads/connector/j/3.0.html> – laden Sie einfach das ZIP-Archiv herunter und setzen Sie es auf den Klassenpfad. Die Treiberversion 3.0 kann ab Java 1.3 eingesetzt werden, die Version 3.1 setzt mindestens Java 1.4 voraus. Der Treiber wird über die Klasse `com.mysql.jdbc.Driver` angesprochen, die Connection-URL beginnt mit `jdbc:mysql:`. Eine ausführliche Dokumentation zum Treiber finden Sie auf der Seite <http://dev.mysql.com/doc/connector/j/en/>.

9.3.2 PostgreSQL

»PostgreSQL« ist ebenfalls eine Open Source-Datenbank, die vielleicht nicht ganz so bekannt wie MySQL, dafür aber aus Datenbank-Sicht technisch besser ist (auch wenn MySQL 5 in diesem Bereich aufholen wird). Die Datenbank wird seit über 15 Jahren weiterentwickelt, läuft auf vielen UNIX-Systemen und ist auch im Hochlast-Betrieb äußerst stabil.

Installation und Test

Wenn Sie möchten, können Sie sich die Quelltexte von <http://www.postgresql.org/> herunterladen und PostgreSQL selber kompilieren – Apple beschreibt im Artikel <http://developer.apple.com/internet/opensource/postgres.html>, wie dies funktioniert. Einfacher ist es jedoch, wenn Sie das fertig übersetzte Installationspaket von der Seite <http://www.entropy.ch/software/macosx/postgresql/> verwenden, wo derzeit das Archiv `pgsql-7.4.3.pkg.tar.gz` angeboten wird. Nach der Installation befindet sich PostgreSQL dann im Verzeichnis `/usr/local/pgsql/`.

Wie bei MySQL legen Sie dann mit dem NetInfo Manager eine Gruppe und einen Benutzer »postgres« an. Als »gid« und »pid« können Sie 401 verwenden, falls diese Nummer bei Ihnen nicht schon anderweitig genutzt wird. Da Sie sich gleich auf diesem Benutzerkonto anmelden werden, müssen Sie als »shell« z.B. `/bin/tcsh` und bei »home« irgendein Verzeichnis (z.B. `/usr/local/pgsql/postgres_home`) eintragen. Legen Sie das »home«-Verzeichnis im Terminal an und geben Sie dem »postgres«-Benutzer ein sicheres Passwort.

```
[straylight:~] much% sudo chown -R postgres /usr/local/pgsql/
```

Mit `chown` übertragen Sie die Rechte am `pgsql`-Verzeichnis auf den »postgres«-Benutzer, der alle weiteren Aktionen mit der Datenbank durchführen wird.

```
[straylight:~] much% su - postgres
Password:
[Straylight:~] postgres% /usr/local/bin/initdb -D /usr/local/pgsql/data
[Straylight:~] postgres% /usr/local/bin/pg_ctl start -D /usr/local/pgsql/data -l postgres.log
```

Nun melden Sie sich mit `su` als Benutzer »postgres« an. Anschließend initialisiert und startet dieser Benutzer die Datenbank.

```
[Straylight:~] postgres% /usr/local/bin/createdb test
CREATE DATABASE
[Straylight:~] postgres% /usr/local/bin/psql test
```

```

Welcome to psql 7.4.3, the PostgreSQL interactive terminal.
Type: \copyright for distribution terms
      \h for help with SQL commands
      \? for help on internal slash commands
      \g or terminate with semicolon to execute query
      \q to quit
test=# SELECT now();
           now
-----
2004-09-30 11:11:02.657851+02
(1 row)
test=# \q

```

Bevor Sie mit PostgreSQL arbeiten können, müssen Sie eine erste Datenbank mit `createdb` anlegen (die hier wieder den Namen `test` erhält). Das Kommando `psql` startet dann den interaktiven PostgreSQL-Befehlsinterpreter. Beendet wird der Datenbank-Server mit dem Verwaltungswerkzeug `pg_ctl`:

```
/usr/local/bin/pg_ctl stop -D /usr/local/pgsql/data
```

Zusätzlich zum eigentlichen PostgreSQL-Installationspaket können Sie auch ein Paket für ein StartupItem herunterladen, das alle nötigen Dateien im Verzeichnis `/Library/StartupItems/PostgreSQL/` anlegt. Ob der Datenbank-Server dann tatsächlich gestartet wird, hängt von der Variable `POSTGRES` in der Datei `/etc/hostconfig` ab.

JDBC-Treiber

Der passende JDBC-Treiber steht auf der PostgreSQL-Homepage unter der Adresse <http://jdbc.postgresql.org/download.html> zur Verfügung. Es gibt einen JDBC2-Treiber für Java 1.2 und 1.3 sowie einen JDBC3-Treiber für Java 1.4. Der jeweilige Treiber wird über die Klasse `org.postgresql.Driver` angesprochen, die Connection-URL beginnt mit `jdbc:postgresql:`.

9.3.3 Firebird

»Firebird« blickt auf eine über 20-jährige Entwicklungsgeschichte zurück und kann getrost im Hochlast-Bereich eingesetzt werden. Dennoch ist diese Datenbank eine Open Source-Entwicklung, denn sie ist aus dem Quelltext der InterBase-Datenbank hervorgegangen, den Borland vor einigen Jahren veröffentlicht hat (<http://www.borland.com/interbase/>).

Firebird 1.5 können Sie fertig für MacOS X kompiliert von <http://www.firebirds.org/> herunterladen, das Archiv ist angenehm klein. Dort finden Sie auch

den JDBC-Treiber »JayBird« in der Version 1.5.1 als separate Archive für Java 1.3 bzw. Java 1.4. Der Treiber wird über die Klasse `org.firebirdsql.jdbc.FBDriver` angesprochen, die `Connection`-URL beginnt mit `jdbc:firebirdsql:`. Der genaue Aufbau der URL ist auf <http://jaybirdwiki.firebirdsql.org/Jay-Bird/JdbcUrls> dokumentiert.

9.3.4 HSQLDB

»HSQLDB« ist eine Weiterentwicklung der »Hypersonic SQL«-Datenbank (<http://hsqldb.sourceforge.net/>). HSQLDB ist Open Source, komplett in Java programmiert, sehr klein und schnell. Die Daten können im Speicher und auf der Festplatte gehalten werden. Zur Installation der Datenbank laden Sie das Archiv von der Seite <http://hsqldb.sourceforge.net/> herunter, binden das JAR-Archiv im Klassenpfad ein und bauen eine Verbindung zur Datenbank auf – falls sie nicht schon läuft, startet sie dann automatisch.

Der JDBC-Treiber ist in das Programmarchiv integriert und wird über die Klasse `org.hsqldb.jdbcDriver` angesprochen. Die `Connection`-URL beginnt mit `jdbc:hsqldb:` und ist auf der Seite <http://hsqldb.sourceforge.net/doc/src/org/hsqldb/jdbc/jdbcConnection.html> ausführlich beschrieben.

9.3.5 Oracle

»Oracle« dürfte eine der bekanntesten Datenbanken überhaupt sein. Wenn Sie riesige Datenmengen zu verwalten haben und viele Benutzer darauf zugreifen müssen, kommt Oracle für Sie in die engere Wahl – allerdings müssen Sie dann auch das passende Kleingeld für die Lizenzgebühren mitbringen. Auf der Seite <http://www.oracle.com/technology/tech/mac0s/> finden Sie einen Link zur Vorabversion von Oracle 10g. Für MacOS X ist sowohl der Datenbank-Server als auch der Client verfügbar, und beides kann kostenlos getestet werden. Die Mächtigkeit und Komplexität des Datenbank-Systems erkennt man schon an der Größe des Programmarchivs – über 570 MByte.

Auf der oben genannten Seite befindet sich auch ein Link zu den JDBC-Treibern – Oracle bietet diverse davon an. Während bei älteren Oracle-Versionen zum Teil noch systemabhängige Treiber genutzt wurden, kommen für Oracle 10g reine Java-Treiber zum Einsatz, die auf allen Plattformen genutzt werden können. Während Oracle die Treiber bisher immer in Archiven mit nicht sehr aussagekräftigen Namen wie `classes12.jar` ausgeliefert hat (für die Java 1.2-Treiber), hat das Java 1.4-Archiv `ojdbc14.jar` nun endlich einen kleinen Bezug zu Oracle. Die Treiber werden über die Klasse `oracle.jdbc.driver.OracleDriver` angesprochen, die `Connection`-URL beginnt mit `jdbc:oracle:`.

9.3.6 Sybase Adaptive Server Enterprise

Eine weitere Datenbank für den höheren Bereich ist »Adaptive Server Enterprise« (ASE) von Sybase. Die Entwicklerversion können Sie kostenlos von <http://www.sybase.com/mac/> herunterladen, für den Einsatz müssen Sie den Datenbank-Server dann kaufen.

Der im Entwicklerpaket enthaltene JDBC-Treiber nennt sich bei Sybase »jConnect« und wird je nach jConnect-Version entweder über die Klasse `com.sybase.jdbc.SybDriver` (4.x) oder über `com.sybase.jdbc2.jdbc.SybDriver` (5.x) angesprochen. Die Connect-URL beginnt in beiden Fällen mit `jdbc:sybase:.`

9.3.7 Berkeley DB Java Edition

Ein absoluter Exot in dieser Liste ist die »Berkeley«-Datenbank, die Sie von der Seite <http://www.sleepycat.com/products/je.shtml> herunterladen können. Die Berkeley-Datenbank wird seit ihrer Java-Umsetzung häufiger erwähnt – und gerade daher sollten Sie wissen, dass es sich dabei zwar um eine absolut brauchbare Datenbank handelt, die allerdings nicht relational ist und damit auch keinen JDBC-Treiber besitzt. Die Programmierung erfolgt also über eine komplett andere API als die übrigen der vorgestellten Datenbank-Systeme! Der Artikel <http://today.java.net/pub/a/today/2004/08/24/sleepy.html> gibt Ihnen einen Überblick über die Anwendung und Programmierung von Berkeley.

9.4 ODBC

ODBC (»Open Database Connectivity«) ist ursprünglich ein Microsoft-Standard zum Datenbankzugriff aus beliebigen Programmiersprachen. Mittlerweile gibt es auch eine Open Source-Implementation dieser Schnittstelle (<http://www.iodbc.org/>), die Apple seit Mac OS X 10.2 ins System integriert hat. Zum Verwalten der ODBC-Datenquellen ist standardmäßig die Anwendung `/Programme/Dienstprogramme/ODBC Administrator` vorhanden.

Obwohl die ODBC-Treiber von Mac OS X gut funktionieren, ist der ODBC Administrator leider nicht besonders gut zu bedienen. Das größte Problem aus Java-Sicht ist aber, dass die so genannte »JDBC-ODBC-Bridge« – ein JDBC-Treiber, der die in ODBC konfigurierten Datenquellen anspricht – unter Mac OS X nicht vorhanden ist. Java-Code, der die Treiber-Klasse `sun.jdbc.odbc.JdbcOdbcDriver` verwendet, wird also mit einer Fehlermeldung abbrechen. Dies ist zwar nicht schön, andererseits gehört der Treiber aber sowieso nicht zu den J2SE-Standardbibliotheken.

Im Netz finden sich einige ODBC-Treiber für Mac OS X, bessere ODBC-Konfigurationsprogramme und auch JDBC-ODBC-Brückentreiber. Letztendlich stellen sie für Mac OS X aber keine ideale Lösung dar, und Sie sollten deren Einsatz nur in Erwägung ziehen, wenn Sie ODBC-Datenquellen anbinden müssen, für die keine reinen Java-Treiber existieren – was selten vorkommt.

9.5 Literatur & Links

► Datenbanken & SQL

- ▶ Connolly/Begg/Strachan, »Datenbanksysteme«, Addison-Wesley 2002
Ein sehr ausführliches Werk zu den Grundlagen von Datenbank-Entwurf, -Implementierung und -Verwaltung.
- ▶ Throll/Bartosch, »Einstieg in SQL«, Galileo Computing 2004
Eine praktische Einführung in die SQL-Programmierung – mit einem Überblick über die wichtigsten Konzepte relationaler Datenbanken.
- ▶ M. Throll, »MySQL 4« 2. Aufl., Galileo Computing 2003
Eine ausführliche Anleitung und ein Nachschlagewerk zur Installation und Administration von MySQL – mit Entscheidungshilfen für den MySQL-Einsatz.

► JDBC

- ▶ D. Bales, »JDBC Pocket Reference«, O'Reilly 2003
Enthält neben alle relevanten APIs und kurzen (aber vollkommen ausreichenden) Beispielen eine Liste der wichtigsten Datenbanken und ihrer JDBC-Treiber.
- ▶ D. Bales, »Java Programming with Oracle JDBC«, O'Reilly 2002
Auch wenn JDBC ein Standard für den Java-Datenbankzugriff ist, bieten diverse Hersteller Erweiterungen für ihre Produkte an. Entsprechend behandelt dieses Buch neben den JDBC-Grundlagen die JDBC-Optimierung mit den speziellen Oracle-Möglichkeiten.
- ▶ G. Reese, »Java Database Best Practices«, O'Reilly 2003
Zeigt alle wichtigen Konzepte der Datenspeicherung unter Java auf, d.h. nicht nur JDBC, sondern auch mögliche Alternativen.
- ▶ <http://java.sun.com/products/jdbc/>
Sun bietet auf dieser Seite alle Spezifikationen und weiterführenden Links zu JDBC.

10 Servlets und JavaServer Pages (JSP)

10.1	Servlets	446
10.2	Tomcat	448
10.3	JavaServer Pages (JSP)	453
10.4	Webapplikationen	455
10.5	Literatur & Links	459

- 1 **Grundlagen**
 - 2 **Entwicklungsumgebungen**
 - 3 **Grafische Benutzungsoberflächen (GUI)**
 - 4 **Ausführbare Programme**
 - 5 **Portable Programmierung**
 - 6 **Mac OS X-spezifische Programmierung**
 - 7 **Grafik und Multimedia**
 - 8 **Werkzeuge**
 - 9 **Datenbanken und JDBC**
- 10 Servlets und JavaServer Pages (JSP)**
- 11 **J2EE und Enterprise JavaBeans (EJB)**
 - 12 **J2ME und MIDP**
- A Kurzeinführung in die Programmiersprache Java**
- B Java auf Mac OS 8/9/Classic**
- C Java 1.5 »Tiger«**
- D System-Properties**
- E VM-Optionen**
- F Xcode- und Project Builder-Einstellungen**
- G Mac OS X- und Java-Versionen**
- H Glossar**
- I Die Buch-CD**

10 Servlets und JavaServer Pages (JSP)

*»Hier in dem Lokal vor zwei Jahren/
hier am gleichen Tisch/
der gleiche Wirt hat serviert«
(Herbert Grönemeyer)*

Webseiten bestehen seit vielen Jahren nicht mehr nur aus statischen HTML-Seiten, sondern aus dynamisch generierten Seiten, deren Inhalt interaktiv beispielsweise aus Datenbanken ermittelt und passend zusammengestellt wird. Dazu stehen auf Web-Servern diverse Technologien zur Verfügung: Früher vor allem CGIs, heutzutage PHP, Active Server Pages (ASP) und – wenn Sie geeignete Java-Software auf dem Server einsetzen – Java Servlets und JavaServer Pages (JSP).

Servlets und JSP sind Java-Klassen, die jeweils für eine oder mehrere Webadressen (URLs) den Seiteninhalt bei Anforderung dynamisch erzeugen. Die Verwaltung der Klassen und die Zuordnung der Anforderungen erfolgt durch einen so genannten Servlet- bzw. JSP-Container. In diesem Kapitel wird dafür das kostenlose und bekannte »Tomcat« verwendet, das entweder alleine oder in einem Web-Server (beispielsweise Apache) integriert eingesetzt werden kann. Ein anderer bekannter Servlet-Container ist »Jetty« (<http://www.mortbay.org/jetty/>).

Der Unterschied zwischen Servlets und JSP besteht in ihrer Programmierung. Während Sie Servlets ganz normal als Java-Unterklasse schreiben, ist eine Java-Server Page eine HTML-Seite mit vereinzelt Java-Code und diversen kleinen Platzhaltern. Entsprechend setzen Sie ein Servlet dann ein, wenn Sie vor allem Steuerungscode schreiben und nur wenig feste Ausgabedaten erzeugen wollen. JSPs kommen zum Einsatz, wenn die Seiten vor allem aus unveränderlichem HTML-Code bestehen, der nur an einzelnen Stellen dynamisch angepasst werden muss.

Mac OS X 10.3 unterstützt IPv6 (das Internet-Protokoll in der neuen Version 6), das von Java 1.4 automatisch als Standard-Netzwerkprotokoll verwendet wird. Sie können die Verwendung des älteren IPv4 erzwingen, indem Sie die System-Property `java.net.preferIPv4Stack` auf »true« setzen – normalerweise sollte dies aber nicht erforderlich sein. Falls nötig, müssen Sie die Property beim Aufruf des Servlet-Containers setzen.

10.1 Servlets

Java Servlets können allgemein zur Erzeugung und Verarbeitung von Daten auf dem Server eingesetzt werden – nicht nur für HTML (sondern z.B. auch für PDF-Dateien und Bilder) und nicht nur auf HTTP-Web-Servern. Daher finden Sie im Paket `javax.servlet` die allgemeine Servlet-Oberklasse `GenericServlet`. Das wichtigste Einsatzgebiet sind aber sicherlich HTML-Seiten, wofür die Oberklasse `javax.servlet.http.HttpServlet` zum Einsatz kommt:

```
//CD/examples/ch10/ServletTest/ServletTest.java
import java.io.*;
import java.util.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletTest extends HttpServlet {

    public void doGet( HttpServletRequest req,
                      HttpServletResponse res )
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<html><head><title>Hallo</title></head><body>");
        out.println("<h1>Hallo Servlet-Welt!</h1>");
        out.println("<p>Zeit auf dem Server: ");
        out.println(
            DateFormat.getDateTimeInstance().format( new Date() )
        );
        out.println("</body></html>");
    }
}
```

Listing 10.1 Ein einfaches Servlet

Das Testservlet überschreibt die Methode `doGet()`, in der bei Servlets normale Anfragen von Web-Browsern verarbeitet werden. Als Parameter wird vom Servlet-Container ein `HttpServletRequest`-Objekt mitgegeben, in dem sich alle relevanten Daten zur Anforderung der Webseite befinden – beispielsweise Anfrage-Parameter oder Cookies. In das ebenfalls übergebene `HttpServletResponse`-Objekt schreibt das Servlet seine Ausgabedaten hinein, damit der Servlet-Container sie an den anfragenden Web-Browser weiterleiten kann.

In der Methode wird zunächst der MIME-Typ der Ausgabedaten mit `setContentType()` auf `text/html` gesetzt – dadurch weiß der anfragende Web-Browser, wie die Daten darzustellen sind. Für normalen Text können Sie hier `text/plain` angeben. Danach wird mit `getWriter()` ein Ausgabekanal erfragt, in den dann die eigentlichen HTML-Daten der Webseite geschrieben werden. Dabei wird nicht nur statischer HTML-Code geliefert, sondern auch das aktuelle Datum ermittelt und in die Seite eingebaut.

Damit Sie das Servlet kompilieren können, benötigen Sie die Klassen der Pakete `javax.servlet` und `javax.servlet.http`, die nicht zur Java Standard Edition (J2SE), sondern zur Enterprise Edition (J2EE) gehören. Sie werden aber auch mit jedem Servlet-Container mitgeliefert, und Sie können sie auch separat von <http://java.sun.com/products/servlet/download.html> herunterladen – auf der Seite finden Sie im Bereich »Specifications« den Eintrag »Download class files 2.3«.

Das heruntergeladene ZIP-Archiv brauchen Sie nicht auspacken, Sie können es direkt auf den Klassenpfad des Compilers setzen (siehe Abbildung 10.1). In Xcode können Sie dort auch bei den »Java Archive Settings« einstellen, dass anstelle des JAR-Archivs eine Klassenhierarchie mit einzelnen Klassen erzeugt werden soll.

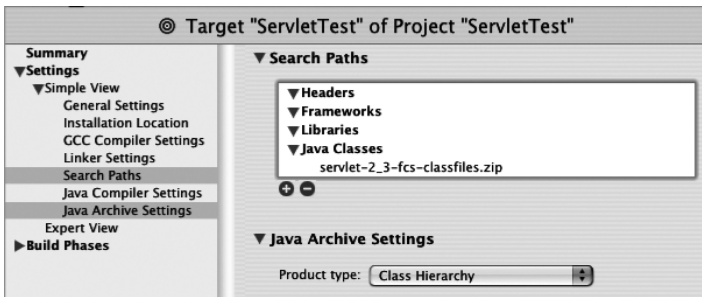


Abbildung 10.1 Xcode-Einstellungen zum Übersetzen einer Servlet-Klasse

Wenn Sie das Servlet nun übersetzen, finden Sie im `build`-Verzeichnis des Projekts die Klasse `ServletTest.class`. Wie diese Klasse im Servlet-Container installiert wird, wird im folgenden Abschnitt über »Tomcat« beschrieben. Der Aufruf des installierten Servlets liefert dann die in Abbildung 10.2 gezeigte Webseite.

Interessant ist der HTML-Code der dynamisch erzeugten Seite. Natürlich tauchen dort alle HTML-Elemente auf, die im Servlet fest kodiert sind. Das aktuelle Datum aber, das im Servlet mit einem `java.util.Date`-Objekt ermittelt wurde, ist ebenfalls nur noch eine Zeichenkette – der Client, also der Web-

Browser, bekommt überhaupt nichts davon mit, dass die Seite von einer Java-Klasse generiert wurde!



Abbildung 10.2 Die generierte Seite im Web-Browser

```
<html><head><title>Hallo</title></head><body>
<h1>Hallo Servlet-Welt!</h1>
<p>Zeit auf dem Server:
19.09.2004 19:53:03
</body></html>
```

Listing 10.2 Der vom Servlet erzeugte HTML-Code

10.2 Tomcat

»Tomcat« ist der Open-Source Servlet-Container des Apache-Jakarta-Projekts. Zusätzlich zu den mit Servlets und JSPs generierten dynamischen Webseiten liefert Tomcat auch ganz normale statische HTML-Seiten und andere Ressourcen (z.B. Bilder) – ein spezieller Web-Server (Apache) ist also gar nicht mehr nötig, auch wenn Tomcat in diesen integriert werden kann.

Sie können die aktuelle Tomcat-Version von <http://jakarta.apache.org/tomcat/> im Bereich »Download/Binaries« beispielsweise als ZIP-Archiv herunterladen – derzeit ist die Version 5.0.28 aktuell. Packen Sie das Archiv aus, und verschieben Sie das Verzeichnis in den /Programme-Ordner – im Terminal können Sie dann mit der englischen Bezeichnung /Applications/jakarta-tomcat-5.0.28/ darauf zugreifen. Natürlich können Sie als Installationsverzeichnis z.B. auch /usr/local/ verwenden, aber das erstgenannte Verzeichnis lässt sich aus dem Finder besser ansteuern. Wechseln Sie im Terminal in das bin-Verzeichnis und machen Sie die dortigen Skripte mit `chmod a+rx *` ausführbar. Nun können Sie Tomcat mit dem Skript `startup.sh` starten:

```
[straylight:/Applications/jakarta-tomcat-5.0.28/bin] much% ./
startup.sh
Using CATALINA_BASE:  /Applications/jakarta-tomcat-5.0.28
```

```
Using CATALINA_HOME: /Applications/jakarta-tomcat-5.0.28
Using CATALINA_TMPDIR: /Applications/jakarta-tomcat-5.0.28/temp
Using JAVA_HOME: /Library/Java/Home
```

Wenn Sie ungefähr obige Ausgabe (und keine Fehlermeldung) sehen, ist der Servlet-Container bereit, und Sie können in einem beliebigen Browser die Startseite auf dem lokalen Rechner am Port 8080 über die Webadresse `http://localhost:8080/` aufrufen (siehe Abbildung 10.3). Statt »localhost« können Sie alternativ auch die IP-Form `http://127.0.0.1:8080/` verwenden. Mit `./shutdown.sh` fahren Sie Tomcat dann wieder herunter.

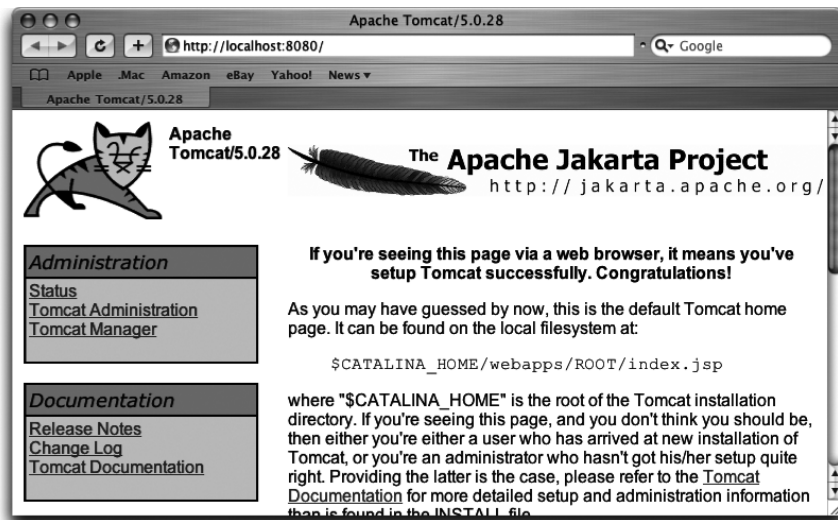


Abbildung 10.3 Lokale Startseite von Tomcat

Damit Sie Tomcat nicht jedes Mal von Hand starten müssen, können Sie im System ein so genanntes »Startup-Item« definieren, mit dem Tomcat bei jedem Login automatisch gestartet wird. Dazu erzeugen Sie das Verzeichnis `/Library/StartupItems/Tomcat/` und legen darin folgende Datei Tomcat an:

```
#!/bin/sh
# CD/examples/ch10/StartupItems/Tomcat/Tomcat
. /etc/rc.common

export JAVA_HOME=/Library/Java/Home
export CATALINA_HOME=/Applications/jakarta-tomcat-5.0.28

StartService ()
```

```

{
    ConsoleMessage "Tomcat wird gestartet"
    $CATALINA_HOME/bin/startup.sh
}

StopService ()
{
    ConsoleMessage "Tomcat wird beendet"
    $CATALINA_HOME/bin/shutdown.sh
}

RestartService () { StopService; StartService; }

RunService "$1"

```

Listing 10.3 Skriptdatei /Library/StartupItems/Tomcat/Tomcat

Den Pfad bei der Umgebungsvariable `CATALINA_HOME` müssen Sie natürlich an Ihre Installation anpassen. Im Skript finden Sie drei Unterroutinen zum Starten, Stoppen und Neustarten des Dienstes, die vom System bei Bedarf aufgerufen werden. Tomcat selbst bietet keine Möglichkeit zum Neustart, daher wird der Servlet-Container in diesem Fall einfach angehalten und dann wieder ganz normal gestartet. Machen Sie dieses Skript nun mit `chmod a+rx Tomcat` ausführbar.

Im selben Verzeichnis muss auch eine Datei `StartupParameters.plist` existieren, die das StartupItem beschreibt. Wenn Sie sich andere StartupItems ansehen, sieht der Inhalt der Datei oft wie folgt aus:

```

{
    Description      = "Tomcat";
    Provides         = ("Tomcat");
    Uses             = ("Network");
    OrderPreference = "None";
}

```

Listing 10.4 NeXTSTEP-Konfigurationsdatei

Hierbei handelt es sich um eine Konfigurationsdatei im alten NeXTSTEP-Format. Obwohl Mac OS X solche Dateien noch problemlos auswerten kann, sollten Sie nach Möglichkeit das neuere, zukunftssichere XML-Format für diese Konfigurationsdatei verwenden, das Sie mit dem Property List Editor aus dem Verzeichnis `/Developer/Applications/Utilities/` erzeugen können:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<!-- CD/examples/ch10/StartupItems/Tomcat/
StartupParameters.plist
-->
<plist version="1.0">
<dict>
    <key>Description</key>
    <string>Tomcat</string>
    <key>OrderPreference</key>
    <string>None</string>
    <key>Provides</key>
    <array>
        <string>Tomcat</string>
    </array>
    <key>Uses</key>
    <array>
        <string>Network</string>
    </array>
</dict>
</plist>

```

Listing 10.5 Mac OS X-XML-Konfigurationsdatei

Der Schlüssel `Provides` definiert den Dienstnamen, auf den andere Startup-Items zurückgreifen können. Hier ist beim Schlüssel `Uses` festgelegt, dass der `Network`-Dienst benutzt wird. Anstelle von `Uses` könnten Sie auch den Schlüssel `Requires` verwenden, der eine stärkere Bindung darstellt – falls der gewünschte Dienst nicht vorhanden ist, würde dieses StartupItem dann nicht gestartet werden. Jetzt können Sie das StartupItem testen und die einzelnen Aktionen im Terminal über den `SystemStarter` aufrufen:

```

sudo SystemStarter start Tomcat
sudo SystemStarter restart Tomcat
sudo SystemStarter stop Tomcat

```

Falls Sie Tomcat nicht mehr automatisch starten lassen wollen, genügt es, das Verzeichnis `Tomcat` aus `/Library/StartupItems/` zu entfernen.

Eine einfache Möglichkeit zum Starten und Beenden von Tomcat bietet der Tomcat Monitor, den Sie kostenlos auf der Seite <http://www.chimoosoft.com/tomcatmonitor.html> erhalten. Die Steuerung erfolgt entweder über den Programmdialog oder – auch bei ausgeblendetem Programm – über das Kontext-Popup-Menü des Dock-Symbols (siehe Abbildung 10.4).



Abbildung 10.4 Tomcat-Monitor

Nun können Sie das Testservlet aus dem vorangegangenen Abschnitt in Tomcat installieren. Kopieren Sie dazu die Datei `ServletTest.class` in das Verzeichnis `webapps/ROOT/WEB-INF/classes/` innerhalb des Tomcat-Ordners (das `classes`-Verzeichnis muss dabei je nach Tomcat-Version eventuell noch angelegt werden).

Der hier vorgestellte Einsatz eines Servlets im so genannten ROOT-Kontext funktioniert zum Einstieg und zum Testen gut – allerdings sollten Sie dies (und insbesondere die folgenden Anpassungen der Konfigurationsdateien) niemals im produktiven Einsatz verwenden. Abschnitt 10.4 stellt den Einsatz eines Servlets als vernünftige Webapplikation vor.

Damit Ihr Servlet in diesem ROOT-Kontext erkannt und ausgeführt wird, müssen Sie in der Datei `conf/web.xml` das so genannte »Invoker-Servlet« aktivieren. Stellen Sie dazu sicher, dass die folgenden beiden Abschnitte nicht mehr durch XML-Kommentare deaktiviert sind – am einfachsten suchen Sie nach der Zeichenkette »invoker«, die in den zwei Abschnitten vorkommt:

```
<servlet>
  <servlet-name>invoker</servlet-name>
  <servlet-class>
    org.apache.catalina.servlets.InvokerServlet
  </servlet-class>
```



```

    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>invoker</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>

```

Listing 10.6 Zwei Abschnitte müssen in web.xml aktiviert werden.

Nach der Änderung der Konfigurationsdatei können Sie Tomcat wieder starten und Ihr Servlet unter der Adresse *http://localhost:8080/servlet/ServletTest* aufrufen. Beachten Sie die Zeichenkette »servlet« innerhalb der URL – diese ist durch die Konfiguration des Invoker-Servlets bedingt. Der Web-Browser sollte Ihnen nun die im vorherigen Abschnitt in Abbildung 10.2 gezeigte Webseite darstellen.

Apple selbst liefert auch eine Tomcat-Installation aus, die Bestandteil des Applikations-Server-Pakets ist. Dieses Paket konnte optional von älteren Xcode-CDs installiert werden, eine aktuelle Version finden Sie in der Apple Developer Connection (*https://connect.apple.com/*) als »Application Servers Developer Preview«. Das von Apple installierte Tomcat finden Sie im Verzeichnis */Library/Tomcat/* – allerdings handelt es sich hierbei derzeit um eine nicht mehr aktuelle 4.1er-Version.

10.3 JavaServer Pages (JSP)

Während ein Servlet alle Ausgabedaten dynamisch selbst zusammenbaut und zum Web-Browser zurückliefert, besteht eine JSP-Seite vor allem aus statischem HTML-Code, in dem sich nur relativ kleine Java-Schnipsel befinden, die dann im Moment des Seitenaufrufs ersetzt werden. Die Beziehung von JSPs zu Servlets wird erst bei der Auswertung durch den Servlet/JSP-Container deutlich: Beim ersten Aufruf der JSP-Seite wird daraus automatisch der Java-Quelltext für eine Servlet-Klasse erzeugt. Dieser temporäre Quelltext wird dann kompiliert und anschließend als ganz normales Servlet ausgeführt. Im Folgenden sehen Sie ein kurzes Beispiel für eine JSP-Seite mit dynamischen Elementen:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<!-- CD/examples/ch10/jsptest.jsp -->
<%@ page session="false" %>
<html>
<head>
    <title><%= application.getServerInfo() %></title>
</head>
<body>
<h1>Hallo JSP-Welt!</h1>
<%
    java.util.Date datum = new java.util.Date();
%>

<p>Guten
<% if (datum.getHours() < 12) { %>
Morgen.
<% } else if (datum.getHours() < 18) { %>
Tag.
<% } else { %>
Abend.
<% } %>

</body>
</html>

```

Listing 10.7 JSP-Datei jsptest.jsp

Der HTML-Code der Seite sieht an sich normal aus – bis auf die JSP-Elemente, die mit `<%` beginnen. Zunächst stoßen Sie bei `<%@ page` auf eine JSP-Direktive, die keine Ausgabe generiert, sondern die zur Steuerung des JSP-Containers dient. Hier wird für diese Seite die Session-Verwaltung ausgeschaltet, d.h., es müssen keine Benutzerdaten für diese Sitzung gespeichert werden – dies bedeutet einen Performanzgewinn.

Im `<title>`-Element der Seite folgt nach `<%=` ein JSP-Ausdruck (eine JSP-Expression). Dies sind Java-Ausdrücke, deren Ergebnis an dieser Stelle der Seite eingetragen wird. Hier wird über das implizite JSP-Objekt `application` der Name des JSP-Containers ermittelt – beispielsweise »Apache Tomcat/5.0.28«.

Anschließend wechseln sich statischer HTML-Code und einige JSP-Scriptlets (mit `<% %>` eingeschlossene Java-Blöcke) ab. Im ersten Scriptlet wird ein Date-

Objekt erzeugt, mit dem in den folgenden Scriptlets anhand der Stunden die Tageszeit als fester HTML-Text eingefügt wird.

JSP-Seiten besitzen die Dateinamenserweiterung *.jsp, speichern Sie diese Datei daher beispielsweise als `jspstest.jsp` ab. Damit ist die JSP-Seite nun fertig, Sie müssen sie nicht noch kompilieren – dies macht ja der Servlet/JSP-Container zur Laufzeit. Kopieren Sie die Datei daher einfach in das Verzeichnis `webapps/ROOT/`, danach können Sie sie unter der Adresse `http://localhost:8080/jspstest.jsp` aufrufen. Beim ersten Aufruf der Seite braucht Tomcat bis zur Anzeige spürbar länger als bei den folgenden Aufrufen – der JSP-Text wird in ein Servlet übersetzt, dieses kompiliert und dann ausgeführt. Bei den folgenden Aufrufen kann dann sofort auf das bereits kompilierte JSP-Servlet zurückgegriffen werden.

Um den Java-Code so weit wie möglich aus dem HTML-Quelltext zu eliminieren, können Sie Element-Bibliotheken (»Tag Libraries« oder kurz »Taglibs«) installieren, die den JSP-Seiten die Java-Funktionalität in Form von XML-Elementen zur Verfügung stellen. Die wichtigste Element-Bibliothek ist die »JSP Standard Tag Library« (JSTL), die Sun auf der Seite `http://java.sun.com/products/jsp/jstl/` beschreibt.

10.4 Webapplikationen

Eine Webapplikation (kurz »web app«) fasst alle Elemente zusammen, die zu einer bestimmten Webanwendung gehören – statische HTML-Seiten, Servlets, JSP-Seiten, Bilder usw. Damit lässt sich die Webapplikation besser als Einheit verwalten und unabhängig von anderen Webanwendungen konfigurieren.

In diesem Abschnitt wird eine Anwendung vorgestellt, die zwar nur aus einem Servlet besteht, die aber die vollständige Struktur einer Webapplikation besitzt. Dazu sehen Sie zunächst das betreffende Servlet, das in einem Formular den Namen des Benutzers erfragt und ihn dann mit seinem Namen begrüßt:

```
//CD/examples/ch10/ServletFormular/ServletFormular.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletFormular extends HttpServlet {

    public void doGet(HttpServletRequest req,
                      HttpServletResponse res)
        throws ServletException, IOException {
```

```

String uri = req.getRequestURI();

res.setContentType("text/html");

PrintWriter out = res.getWriter();

out.println("<html><head><title>Hallo</title></head><body>");
out.println("<form method=POST action=\""+uri+"\">");
out.println("Wie hei&szlig;t Du?");
out.println("<input type=TEXT name=username size=20>");
out.println("<input type=SUBMIT value=\"Abschicken\">");
out.println("</form>");

out.println("</body></html>");
}

```

Die `doGet()`-Methode ist sehr ähnlich zu der Methode aus Abschnitt 0 aufgebaut. Allerdings wird hier der Code für ein HTML-Formular generiert, das der Benutzer mit einem SUBMIT-Knopf an den Server (d.h. hier wieder an den Servlet-Container) abschicken kann. Der eingegebene Benutzername wird dabei als Parameter mit dem Namen `username` mitgeschickt.

Zum Übertragen von Formulardaten (bzw. Webseiten allgemein) gibt es zwei Arten oder *Methoden*: Normalerweise werden Seiten mit der HTTP-GET-Methode abgefragt (daher der Name `doGet()`). Solche Abfragen können problemlos gepuffert und beliebig häufig ohne Seiteneffekte wiederholt werden. Die POST-Methode dagegen ist für einmalige Aktionen gedacht, die ein Web-Browser nicht zwischenspeichern oder wiederholen darf – beispielsweise das Bestellen eines Buches. Abgesehen von diesem (wichtigen!) Bedeutungsunterschied sind die Methoden technisch ansonsten identisch.

Eine Webadresse kann durchaus mit beiden Methoden angesprochen werden und darf dabei unterschiedliche Ergebnisse liefern. Daher schickt das Formular dieses Servlets seine Daten ganz einfach mit der POST-Methode an seine eigene Adresse, die es mit `getRequestURI()` ermittelt und in den Formular-Quelltext als `action` eingebaut hat. Die Behandlung der POST-Anfrage erfolgt dann in der entsprechenden Servlet-Methode `doPost()`:

```

public void doPost(HttpServletRequest req,
                   HttpServletResponse res)
    throws ServletException, IOException {
    String user = req.getParameter("username");
}

```

```

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<html><head><title>Hallo</title></head><body>");
        out.println("<p>Hallo " + user + "!</p>");
        out.println("</body></html>");
    }
}

```

Der Aufbau dieser Methode unterscheidet sich kaum von `doGet()` – beide liefern HTML-Code an den Aufrufer. Hier wird jedoch der `username`-Parameter des Formulars ausgewertet, der in der `HttpServletRequest`-Anfrage mitgeschickt wird und mit `getParameter()` abgefragt werden kann.

Wenn Sie das Servlet nun in Xcode übersetzen lassen, achten Sie darauf, dass Sie keine Klassenhierarchie mit einzelnen Klassen erzeugen lassen, sondern ein ganz normales JAR-Archiv. Dieses Archiv `ServletFormular.jar` wird nun als eigenständige **Webapplikation** in Tomcat installiert.

```

webapps/
  ROOT/
    jsptest.jsp
  WEB-INF/
    web.xml
    classes/
      ServletTest.class
    lib/
test/
  WEB-INF/
    web.xml
    classes/
    lib/
      ServletFormular.jar

```

Listing 10.8 Verzeichnisstruktur zweier Webapplikationen

In der Verzeichnisstruktur taucht zunächst zur besseren Übersicht der `ROOT`-Kontext auf, in dem Sie die bisherigen Testbeispiele installiert haben. Für eine eigene Webapplikation bilden Sie nun die Struktur des `ROOT`-Kontexts in einem separaten `webapps`-Unterverzeichnis nach. Dieses Unterverzeichnis – hier mit dem Namen `test` – definiert den so genannten **Servlet-Kontext**, dessen Name auch als Präfix innerhalb der Aufruf-URLs verwendet wird.

Direkt im `test`-Verzeichnis können statische Ressourcen (HTML-Dateien und Bilder) sowie JSP-Seiten abgelegt werden. Für Servlets und zur Konfiguration der Webapplikation ist das Verzeichnis `WEB-INF` wichtig. Hier platzieren Sie im Unterverzeichnis `classes` einzelne Servlet-Klassen und im Unterverzeichnis `lib` JAR-Archive, die entweder irgendwelche Erweiterungsbibliotheken oder eben auch Servlets enthalten.

Zur eigentlichen Konfiguration der Webapplikation müssen Sie nun noch eine Datei `web.xml` als **Deployment-Deskriptor** (Beschreibungsdatei für den Einsatz) bereitstellen. Die Konfiguration kann nahezu beliebig aufwändig werden, aber für den Einsatz nur eines Servlets reicht der folgende einfache Deskriptor aus:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<!-- CD/examples/ch10/webapps/test/WEB-INF/web.xml -->
<web-app>
  <servlet>
    <servlet-name>Test</servlet-name>
    <servlet-class>ServletFormular</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Test</servlet-name>
    <url-pattern>/MeinFormular.html</url-pattern>
  </servlet-mapping>
</web-app>
```

Listing 10.9 Deployment-Deskriptor `web.xml`

Für jedes Servlet taucht im Deployment-Deskriptor ein `<servlet>`-Element auf, mit dem der Servlet-Klasse ein beliebiger Name zugeordnet wird. Mit diesem Namen wird das Servlet innerhalb der Datei `web.xml` identifiziert – und oft entspricht der Servlet-Name der Einfachheit halber der Servlet-Klasse.

Für jedes Servlet können dann beliebig viele `<servlet-mapping>`-Elemente definiert werden. Hierdurch wird festgelegt, wie das Servlet innerhalb einer Webadresse angesprochen wird. Der Aufruf dieses Servlets erfolgt mit obiger Konfiguration beispielsweise mit `http://localhost:8080/test/MeinFormular.html` – der Anwender kann also an der URL gar nicht mehr erkennen, dass die Webseite dynamisch von einem Java-Servlet generiert wurde.

Um die Verteilung der Web-Applikationen zu vereinfachen, können Sie das gesamte `test`-Verzeichnis mit dem `jar`-Kommandozeilenwerkzeug als **Web-applikationsarchiv** (`test.war`) verpacken und das Archiv im `webapps`-Verzeichnis installieren. Tomcat packt das Archiv dann vor der Ausführung automatisch aus.

Es gibt zahlreiche Frameworks und Bibliotheken, die auf Servlets und JSP aufsetzen und Ihnen das Erstellen von Webapplikationen erleichtern. Ein bekanntes Framework ist »Struts«, das Apple im Artikel <http://developer.apple.com/internet/java/struts.html> vorstellt. Ebenfalls bei Apple finden Sie auf der Seite <http://developer.apple.com/internet/java/tomcat2.html> einen Artikel über eine Beispiel-Webapplikation, die eine Weblog-Anwendung mithilfe von Servlets, JSP und MySQL realisiert.

10.5 Literatur & Links

- ▶ H. Vonhoegen, »Einstieg in JavaServer Pages 2.0«, Galileo Computing 2004
Als Einsteiger in die Web-Server-Programmierung oder Umsteiger von anderen dynamischen Seitengeneratoren finden Sie in diesem Buch alle nötigen Grundlagen zu Java, JSP und Tomcat.
- ▶ H. Bergsten, »JavaServer Pages« 3rd ed., O'Reilly 2003
Ein umfassendes Werk zu JSP 2.0 und JSTL 1.1, das auch eigene Tag-Bibliotheken und Datenbankbindung behandelt.
- ▶ O. Rettig, »Tomcat 5«, Galileo Computing 2004
Dieses Buch stellt Ihnen nicht nur alle Neuerungen der Tomcat-Version 5.0 vor, sondern liefert Ihnen einen kompletten Überblick über die Tomcat-Architektur – inklusive Datenbankbindung.
- ▶ S. Wiesner, »Struts«, Galileo Computing 2004
Das Struts-Framework zur Entwicklung von Webapplikationen wird in diesem Buch kompakt und praxisnah behandelt.
- ▶ <http://java.sun.com/products/servlet/>
Suns Portalseite zur Servlet-Technologie – mit allen Spezifikationen
- ▶ <http://java.sun.com/products/jsp/>
Auf der JSP-Referenzseite finden Sie unter anderem Links zu den wichtigsten Tag-Bibliotheken sowie Informationen zur »JSP Standard Tag Library« (JSTL).
- ▶ <http://developer.apple.com/internet/>
Apple hat zahlreiche Artikel zur Server- und Webprogrammierung bzw. -administration ins Netz gestellt, darunter auch welche zu Tomcat und Struts.

11 J2EE und Enterprise JavaBeans (EJB)

11.1	JBoss	464
11.2	Enterprise JavaBeans (EJB)	467
11.3	Literatur & Links	478

- 1 **Grundlagen**
- 2 **Entwicklungsumgebungen**
- 3 **Grafische Benutzungsoberflächen (GUI)**
- 4 **Ausführbare Programme**
- 5 **Portable Programmierung**
- 6 **Mac OS X-spezielle Programmierung**
- 7 **Grafik und Multimedia**
- 8 **Werkzeuge**
- 9 **Datenbanken und JDBC**
- 10 **Servlets und JavaServer Pages (JSP)**
- 11 **J2EE und Enterprise JavaBeans (EJB)**
- 12 **J2ME und MIDP**
- A **Kurzeinführung in die Programmiersprache Java**
- B **Java auf Mac OS 8/9/Classic**
- C **Java 1.5 »Tiger«**
- D **System-Properties**
- E **VM-Optionen**
- F **Xcode- und Project Builder-Einstellungen**
- G **Mac OS X- und Java-Versionen**
- H **Glossar**
- I **Die Buch-CD**

11 J2EE und Enterprise JavaBeans (EJB)

*»Der Weltraum, unendliche Weiten. Wir schreiben das Jahr 2200.
Dies sind die Abenteuer des Raumschiffs Enterprise [...]
Viele Lichtjahre von der Erde entfernt dringt die Enterprise in Galaxien vor, die nie ein Mensch zuvor gesehen hat.«
(Aus »Star Trek«)*

J2EE, die »Java 2 Platform Enterprise Edition«, ist eine Sammlung diverser Spezifikationen und Programmierrichtlinien, die die Entwicklung großer verteilter, webbasierter Anwendungen vereinfachen und vereinheitlichen sollen. J2EE baut dafür auf J2SE auf und erweitert dies um diverse Klassenbibliotheken, in denen die Spezifikationen umgesetzt sind. Als erweiterte Dienste gegenüber der Java-Standardausgabe kommen unter anderem ein allgemeiner Namensdienst, eine Transaktionsverwaltung, Sicherheit durch Authentisierung und Authorisierung sowie diverse Techniken zur Persistenzsicherung hinzu.

Ein wichtiger Teil von J2EE sind die Enterprise JavaBeans (EJB), die ein Komponentenmodell für den Server spezifizieren. Der Begriff »Bean« (Bohne) soll dabei die Idee vieler kleiner Komponenten versinnbildlichen. Die EJB-Spezifikation hat sich über die Jahre weiterentwickelt und ist mittlerweile bei Version 2.1 angekommen (Version 3.0 befindet sich momentan in der Entwicklung). EJBs werden häufig fälschlicherweise mit J2EE gleichgesetzt, aber Sie können J2EE auch problemlos ohne sie einsetzen – für kleine und mittlere Projekte sind EJBs oft viel zu komplex, während andere J2EE-Technologien (Servlets, JSP) hierfür absolut sinnvoll sind.

Es gibt von Sun ein **J2EE-SDK**, das leider nicht für Mac OS X zur Verfügung steht (auch wenn sich zwischenzeitlich einzelne Versionen mit ein paar Tricks installieren ließen). Für die J2EE-Entwicklung unter Mac OS X stellt dies aber in der Regel kein Problem dar, denn Sie können die nötigen Bibliotheken und Klassen separat herunterladen (wie beispielsweise bei den Servlets) oder die Bibliotheken eines installierten J2EE-Servers verwenden.

So, wie Tomcat Servlet-Container ist (und damit bereits einen Teil von J2EE umsetzt), gibt es EJB-Container, die eine Laufzeitumgebung für EJB-Komponenten zur Verfügung stellen. Ein Server, der diese Container und viele weitere J2EE-Module besitzt, wird **J2EE-Applikations-Server** oder kürzer J2EE-Server genannt. Mit diesen Technologien können Ihre Server-Anwendungen eine Dreischicht-Architektur umsetzen – JSPs für die Darstellung, EJBs für das Datenmodell bzw. die Geschäftslogik und Servlets als Steuerungsschicht dazwischen.

Ein bekannter Open Source-J2EE-Server ist »JBoss«, der kostenlos ist und hervorragend unter Mac OS X läuft – daher konzentriert sich dieses Kapitel auf dieses Produkt, sowohl was die Installation des Servers als auch den Einsatz von EJBs damit betrifft. Als J2EE-Server sind weiterhin »JRun« (<http://www.macromedia.com/software/jrun/>) und der »Pramati Server« (<http://www.pramati.com/>) für Mac OS X verfügbar. Der Einsatz des bekannten »WebLogic Server« unter Mac OS X wird vom Hersteller Beas leider nicht unterstützt – es gibt im Netz aber einige Anleitungen, wie man diesen Server trotzdem einsetzen kann.¹

Bevor Sie nun sofort mit J2EE und EJBs loslegen, sei an dieser Stelle noch ein kurzer Hinweis auf WebObjects erlaubt, Apples Entwicklungs- und Einsatzumgebung für Enterprise-Applikationen. Mit WebObjects wurden bereits Web- und Enterprise-Applikationen genutzt, lange bevor es J2EE und EJBs gab – und auch heute noch hat WebObjects viele einzigartige Konzepte, die erst nach und nach von J2EE-Frameworks und -Bibliotheken übernommen werden. Seit Apple WebObjects komplett in Java neu programmiert hat, stellt die Anbindung an bestehende J2EE-Lösungen kein Problem mehr dar. Ausführliche Informationen zu WebObjects erhalten Sie auf den Seiten <http://www.apple.com/webobjects/> und <http://developer.apple.com/webobjects/>.

Gleich im Anschluss finden Sie Hinweise zur Installation von JBoss, danach folgt eine kurze EJB-Einführung. Dabei werden wirklich nur die absoluten Grundlagen vorgestellt – das gesamte Thema ist viel zu komplex, um hier annähernd vollständig besprochen zu werden.

Die Hinweise auf die Konfiguration und Verwaltung von J2EE-Umgebungen beziehen sich in den folgenden Abschnitten wie gehabt auf die Client-Variante von Mac OS X. Mac OS X Server bringt JBoss als J2EE-Server bereits mit, und die Verwaltung kann recht komfortabel mit dem Server-Administrationswerkzeug /Applications/Server erfolgen. Apple beschreibt dieses Programm ausführlich im Dokument http://images.apple.com/server/pdfs/JavaApplication_Server.pdf.

11.1 JBoss

Der »JBoss Application Server« ist ein kostenloser Open Source-J2EE-Server, der aus einer bestehenden J2SE-Installation eine vollständige J2EE-Umgebung macht. Trotz der freien Verfügbarkeit ist JBoss J2EE 1.4-zertifiziert – der Hersteller verdient sein Geld mit Support und Dokumentation. Intern baut

¹ Siehe <http://www.simgbrown.com/blog/2004/07/20/1090356177000.html>, <http://www.oreillynet.com/pub/wlg/4091> und http://tarunreddy.com/weblogic_on_mac/.

JBoss auf einen JMX-Bus auf (Java Management Extension), über den die diversen Komponenten (so genannte »MBeans«) zusammengeschlossen sind. Durch diesen modularen Aufbau können weitere Komponenten installiert und vorhandene ausgetauscht werden – beispielsweise kann als Servlet-Container wahlweise Tomcat oder Jetty eingesetzt werden. Während JBoss früher wahlweise mit Tomcat oder Jetty ausgeliefert wurde, kommt heute standardmäßig nur noch Tomcat zum Einsatz. JBoss 3.2.x setzt dabei Java 1.3 oder neuer voraus, für JBoss 4.x muss mindestens Java 1.4 vorhanden sein.

Der einfachste Weg, um JBoss auf den eigenen Rechner zu bekommen, besteht in der Installation von Apples **Application Servers**-Paket. Bei den »Xcode Tools« 1.0 und 1.1 war dies eine optionale Komponente, die Sie im Installationsprogramm unter »Anpassen« wählen konnten. Mittlerweile ist das Paket unabhängig von Xcode auf der Seite <http://www.apple.com/downloads/macosx/apple/applicationserverupdate.html> erhältlich – am besten verwenden Sie jedoch die aktuellste »Java Application Servers Developer Preview«, die Sie von Apples ADC-Seiten (<https://connect.apple.com/>) herunterladen können. Auch wenn auf den Seiten von Mac OS X Server die Rede ist, können Sie die Archive auch auf der normalen Client-Version von Mac OS X installieren. Das Paket installiert derzeit JBoss 3.2.x und Tomcat 4.1.x in den Verzeichnissen /Library/JBoss/ bzw. /Library/Tomcat/. Als weitere Werkzeuge werden Ant, log4j und XDoclet im Verzeichnis /Developer/Java/J2EE/ abgelegt. Ein größeres Beispielprojekt, das alle diese Technologien einsetzt, finden Sie im Verzeichnis /Developer/Examples/J2EE/.

Alternativ laden Sie eine aktuellere JBoss-Version direkt von <http://www.jboss.org/downloads/index> herunter, packen das ZIP-Archiv aus und installieren den JBoss-Ordner im /Programme/-Verzeichnis (oder in /usr/local/) – je nach Version können Sie dann im Terminal unter /Applications/jboss-4.0.0/ darauf zugreifen. Machen Sie nun zunächst die Skripte im bin-Verzeichnis mit `chmod a+rx bin/*` ausführbar, danach können Sie das Skript `run.sh` zum Starten von JBoss aufrufen:

```
[straylight:~] much% /Applications/jboss-4.0.0/bin/run.sh
```

```
JBoss Bootstrap Environment
JBoss_HOME: /Applications/jboss-4.0.0
JAVA: /Library/Java/Home/bin/java
JAVA_OPTS: -server -Xms128m -Xmx128m -Dprogram.name=run.sh
CLASSPATH: /Applications/jboss-4.0.0/bin/run.jar:/Library/Java/Home/lib/tools.jar
```

```

13:02:54,970 INFO [Server] Starting JBoss (MX MicroKernel)...
13:02:54,980 INFO [Server] Release ID: JBoss [Zion] 4.0.0 (build: CV
Stag=JBoss_4_0_0 date=200409200418)
13:02:54,983 INFO [Server] Home Dir: /Applications/jboss-4.0.0
...
13:02:56,965 INFO [ServerInfo] Java version: 1.4.2_
05,Apple Computer, Inc.
13:02:56,968 INFO [ServerInfo] Java VM: Java HotSpot(TM) Client VM 1
.4.2-38,"Apple Computer, Inc."
13:02:56,970 INFO [ServerInfo] OS-System: Mac OS X 10.3.5,ppc
...
13:04:17,280 INFO [Server] JBoss (MX MicroKernel) [4.0.0 (build: CVS
Tag=JBoss_4_0_0 date=200409200418)] Started in 1m:21s:320ms

```

Der Start dauert recht lange und gibt *sehr* viele Statusmeldungen aus. Sobald Sie obige »Started«-Meldung sehen, ist dafür dann aber auch ein kompletter J2EE-Server hochgefahren und bereit! Bei Apples Application Servers-Paket müssen Sie zum Starten entsprechend `/Library/JBoss/3.2/bin/run.sh` aufrufen.

Wie schon bei Tomcat können Sie nun unter der Adresse `http://localhost:8080/` auf den Server zugreifen und erhalten eine allgemeine Informationsseite. Unter `http://localhost:8080/jmx-console/` erreichen Sie die JMX-Konsole, die Ihnen alle installierten Komponenten anzeigt. Die Adresse `http://localhost:8080/web-console/` schließlich liefert neben allgemeinen Server-Informationen eine Übersicht über die verfügbaren Web- und Enterprise-Applikationen (siehe Abbildung 11.1).

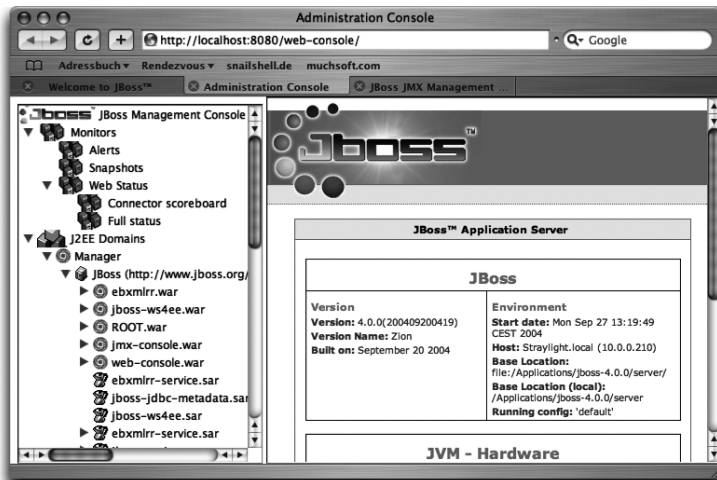


Abbildung 11.1 Jboss-Verwaltungskontrolle

Bei Server-Anwendungen ist nicht nur das Starten wichtig, sondern auch das geordnete Herunterfahren. JBoss 4 beenden Sie am besten in einem neuen Terminal-Fenster mit dem Kommando

```
/Applications/jboss-4.0.0/bin/shutdown.sh -S
```

Im ursprünglichen Terminal-Fenster werden dann zahlreiche Statusmeldungen ausgegeben, bis alle Anwendungen, Komponenten und JBoss selber beendet sind:

```
13:12:54,662 INFO [Server] JBoss SHUTDOWN: Undeploying all packages
...
13:13:07,900 INFO [Server] Shutdown complete
Shutdown complete
Halting VM
```

Bei Apples JBoss 3.2-Installation funktioniert das Shutdown-Skript nicht, hier beenden Sie den Server einfach mit `Ctrl+C` im JBoss-Terminal. Der Server wird dadurch aber nicht einfach aus dem Speicher entfernt, sondern kann sich auch noch korrekt herunterfahren.

Der Aufbau von Enterprise-Applikationen wird zusammen mit ihrem Einsatz im JBoss-Verzeichnis `server/default/deploy/` im folgenden Abschnitt über EJB beschrieben.

11.2 Enterprise JavaBeans (EJB)

Enterprise JavaBeans (EJB) sind Server-Komponenten, die zur Implementierung von Geschäftslogik eingesetzt werden. JSPs stellen die Ergebnisse der Berechnungen dann auf der Client-Seite dar, und Servlets dienen typischerweise als Steuerungsobjekte (Controller) zwischen der Darstellungs- und der Geschäftslogikschicht. EJBs werden vor allem bei mittleren und großen Projekten eingesetzt, bei denen es auf Skalierbarkeit und eine sichere Transaktionsverwaltung ankommt. Bei kleinen und mittleren Projekten reicht es dagegen oft aus, die Geschäftslogik als Servlet zu implementieren.

Es gibt zwei wesentliche Arten von Enterprise JavaBeans: die Session- und die Entity-Beans. Die Session-Beans sind funktionale Komponenten, die zur Verwaltung der Enterprise-Applikation dienen – beispielsweise eine Steuerungskomponente für einen Web-Shop. Entity-Beans stellen die Daten der Anwendung dar, die dauerhaft (persistent) über verschiedene Sitzungen (Sessions) hinweg gespeichert werden müssen – beispielsweise der Warenkorb des Benutzers (auch wenn viele existierende Shop-Systeme den Warenkorb leider am Ende der Sitzung löschen).

Die Session-Beans werden weiter in zustandsbehaftete (»stateful«) und zustandslose (»stateless«) Session-Beans unterteilt. Eine zustandsbehaftete Session-Bean behält ihre Daten über mehrere Methoden-Aufrufe hinweg, die zustandslose Variante kann nur mit den als Methoden-Parameter übergebenen Daten arbeiten. Aus Performanzgründen sollten Sie nach Möglichkeit mit zustandslosen Beans arbeiten, da der Server für sie weniger Aufwand betreiben muss.

Die Speicherung der Daten von Entity-Beans kann auf zwei Arten geschehen, mit CMP- (»Container Managed Persistence«) und mit BMP-Beans (»Bean Managed Persistence«). Bei ersterer kümmert sich der EJB-Container um die Sicherung, bei letzterer muss die Bean selbst den passenden Java-Code zum Speichern mitbringen. Trotz des Mehraufwands wird häufig BMP eingesetzt, da Sie mit dieser Variante mehr Kontrolle über die Datensicherung haben.

Zur Programmierung einer EJB-Komponente müssen Sie neben der eigentlichen Geschäftslogik-Implementierung in der Bean selbst noch zwei Schnittstellen bereitstellen: Zum einen ein so genanntes »Home«-Interface zur Erzeugung der Komponente auf dem Server, zum anderen ein so genanntes »Remote«-Interface, das alle von außen aufrufbaren Methoden der Komponente enthält.

11.2.1 Eine einfache Enterprise-Applikation

Eine komplette Enterprise-Applikation besteht üblicherweise aus EJB-Komponenten, Datenbankzugriffen, Servlets und JSP-Seiten. Das folgende Beispiel setzt der Einfachheit halber nur eine EJB-Komponente und ein Servlet ein. Die Komponente ist dabei eine einfache zustandslose Session-Bean, die als Geschäftslogik eine feste Zeichenkette zurückgibt, die bei Bedarf vom Servlet abgefragt und angezeigt wird.

Die Projektstruktur der Beispielapplikation ist in Abbildung 11.2 ersichtlich. Weil für eine Enterprise-Applikation einige Archive mit einem bestimmten Aufbau erzeugt werden müssen, kommt als Build-Werkzeug meistens Ant zum Einsatz, das die relevanten Schritte weitestgehend automatisiert. Dieses Projekt nutzt das von Xcode 1.5 im Verzeichnis `/Developer/Java/Ant/` installierte Ant 1.6 und nicht die ältere, vom Application Servers-Paket installierte Ant-Version im Verzeichnis `/Developer/Java/J2EE/Ant/`. Für grundlegende Informationen zu Ant sehen Sie bitte in Kapitel 8, *Werkzeuge*, nach.

Das Ant-Buildfile `build.xml` nutzt intern die Pfade `/Library/JBoss/` und `/Library/Tomcat/`, um auf bestimmte Bibliotheken für den Compiler-Klassenpfad zuzugreifen. Wenn Sie andere Installationen verwenden, müssen Sie einfach nur die entsprechenden Property-Definitionen am Anfang des Buildfiles

anpassen. Das Buildfile selbst ist nicht auf Geschwindigkeit, sondern auf die Übersichtlichkeit der nötigen Schritte optimiert. Bevor die einzelnen Schritte zum Zusammenbauen der Enterprise-Applikation vorgestellt werden, sehen Sie aber erst einmal die Quelltexte der EJB-Komponente:

```
//CD/examples/ch11/HalloEJBWelt/ejb/HalloEJBWeltRemote.java
import java.rmi.RemoteException;
import javax.ejb.EJBObject;
public interface HalloEJBWeltRemote extends EJBObject {
    public String hallo() throws RemoteException;
}
```

Listing 11.1 Das Remote-Interface mit den »sichtbaren« Methoden der Geschäftslogik

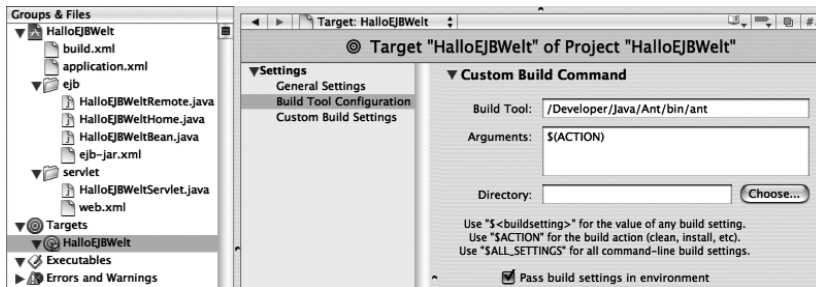


Abbildung 11.2 Xcode-Projektstruktur für eine Enterprise-Applikation

Die EJB-Klassen und -Schnittstellen befinden sich im Paket `javax.ejb`. Das öffentliche Remote-Interface muss daraus vom Interface `javax.ejb.EJBObject` erben. Als öffentliche Schnittstelle ist hier nur die Methode `hallo()` definiert, die einfach eine Zeichenkette zurückgibt. Da der Aufruf dieser Methode über das Netzwerk erfolgt und schief gehen kann, muss sie – wie jede Methode in diesem Interface – eine `java.rmi.RemoteException` werfen können. Am Paketnamen der Exception sehen Sie schon, dass hierbei dasselbe Konzept wie bei Java RMI (»Remote Method Invocation«, entfernter Methodenaufruf) zum Einsatz kommt.

```
//CD/examples/ch11/HalloEJBWelt/ejb/HalloEJBWeltHome.java
import java.rmi.RemoteException;
import javax.ejb.*;
public interface HalloEJBWeltHome extends EJBHome {
    public HalloEJBWeltRemote create()
        throws RemoteException, CreateException;
}
```

Listing 11.2 Das Home-Interface zum Erzeugen der Komponente

Danach definieren Sie das Home-Interface, das von der Schnittstelle `javax.ejb.EJBHome` erben muss. Mithilfe dieses Interfaces kann der Client (bei diesem Beispiel das weiter unten folgende Servlet) das EJB-Objekt auf dem Server erzeugen und sich eine Referenz darauf zurückgeben lassen. Als Minimalanforderung muss daher die Methode `create()` existieren, die eine Referenz auf das Remote-Interface zurückgibt.

```
//CD/examples/ch11/HalloEJBWelt/ejb/HalloEJBWeltBean.java
import javax.ejb.*;

public class HalloEJBWeltBean implements SessionBean {
    private SessionContext context;

    public String hallo() {
        return "Hallo und guten Tag!";
    }

    public void setSessionContext(SessionContext context) {
        this.context = context;
    }

    public void ejbCreate() throws CreateException { }
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbRemove() { }
}
```

Listing 11.3 Die Implementierung der Geschäftslogik als Bean

Nun folgt die eigentliche Geschäftslogik (»business logic«). Die Bean muss dafür die Schnittstelle `javax.ejb.SessionBean` implementieren, wodurch in dieser Klasse einige leere – weil bei diesem Beispiel nicht benötigte – Methoden auftauchen. Diese Methoden werden vom EJB-Container aufgerufen, wenn die entsprechenden Ereignisse eintreten. Ebenso wird der `SessionContext` vom Container gesetzt. Der Code der Geschäftslogik in der Methode `hallo()` ist ziemlich trivial – es wird einfach eine fest kodierte Zeichenkette zurückgegeben.

Damit sind die Java-Quelltexte für die EJB-Komponente vollständig. Wenn Sie sich die Klasse und die Schnittstellen genauer ansehen, bleibt eine Frage: Wer implementiert die Interfaces `HalloEJBWeltRemote` und `HalloEJBWeltHome`? Sollte dies nicht durch die Bean, also durch die Klasse `HalloEJBWeltBean`,

erfolgen? EJB verfolgt hier ein anderes Konzept als z.B. RMI, wo tatsächlich eine direkte Kopplung der Schnittstellen an die Implementierung im Quelltext sichtbar ist. Bei EJB wird die Verbindung erst zur Laufzeit vom EJB-Container hergestellt, der dafür automatisch passende Stellvertreterobjekte generiert. Der Client bekommt hiervon überhaupt nichts mit, da er nur die Schnittstellen verwendet und die tatsächliche Implementierung sowieso nicht kennt.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC
    '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
    'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<!-- CD/examples/ch11/HalloEJBWelt/ejb/ejb-jar.xml -->
<ejb-jar>
  <display-name>HalloEJBWelt</display-name>
  <enterprise-beans>
    <session>
      <ejb-name>HalloEJBWelt</ejb-name>
      <home>HalloEJBWeltHome</home>
      <remote>HalloEJBWeltRemote</remote>
      <ejb-class>HalloEJBWeltBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Listing 11.4 EJB-Deployment-Deskriptor `ejb-jar.xml`

Die Programmierung des Java-Codes ist nur die halbe Miete – je größer eine Web- oder Enterprise-Applikation wird, desto wichtiger und aufwändiger werden die Konfigurationsdateien. Für Ihre EJB-Komponenten müssen Sie einen so genannten EJB-Deployment-Deskriptor anlegen – ähnlich wie bei den Servlets. Innerhalb des `<session>`-Elements legen Sie fest, aus welchen Schnittstellen (`<home>` bzw. `<remote>`) und Klassen (`<ejb-class>`) sich die Komponente zusammensetzt. Das Element `<ejb-name>` gibt an, unter welchem Namen die Komponente auf dem Server gefunden werden kann.

Das Verpacken der Klassen- und Konfigurationsdateien zu einem JAR-Archiv folgt unten beim Ant-Buildfile. Davor benötigen Sie erst noch ein Servlet, das mit der EJB-Komponente kommuniziert – dieses Servlet ist die einzige Benutzerschnittstelle, die Ihre Enterprise-Applikation einem Anwender mit seinem Web-Browser bietet.

```

//CD/examples/ch11/HalloEJBWelt/servlet/HalloEJBWeltServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;

public class HalloEJBWeltServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        HalloEJBWeltRemote remote = null;

        try {
            Context ctx = new InitialContext();
            Object obj = ctx.lookup("HalloEJBWelt");
            HalloEJBWeltHome home =
                (HalloEJBWeltHome)
                    PortableRemoteObject.narrow( obj,
                                                  HalloEJBWeltHome.class );
            remote = home.create();
        }
        catch (Exception e) {
            e.printStackTrace(out);
        }

        out.println("<html><head><title>Hallo</title></head><body>");
        out.println("<h1>Hallo EJB-Welt!</h1>");
        if (remote != null) {
            out.println("<p>Die Bohne sagt: " + remote.hallo() );
        }
        out.println("</body></html>");
    }
}

```

Listing 11.5 Das Servlet als Benutzerschnittstelle der Enterprise-Applikation

Das Servlet besitzt eine normale `doGet()`-Methode, wie sie im vorangegangenen Kapitel beschrieben wurde. Als erstes wird darin mit der Klasse

`InitialContext` ein neuer JNDI-Kontext («Java Naming and Directory Interface») erzeugt, über den anschließend mit `lookup()` nach einer Komponente mit dem angegebenen Namen gesucht wird – der Name entspricht dem Eintrag `<ejb-name>` in der Datei `ejb-jar.xml`. Konnte eine passende Komponente gefunden werden, wird diese mit `PortableRemoteObject.narrow()` in die benötigte Home-Referenz umgewandelt. Auch wenn ab EJB 2.0 meistens eine direkte Typumwandlung mit einem Cast nach `lookup()` funktioniert, sollten Sie dennoch immer `narrow()` verwenden, um unabhängig von der Art der Objektübertragung im Netzwerk zu sein. Auf dieser Home-Referenz kann nun die `create()`-Methode aufgerufen werden, die im Erfolgsfall ein Remote-Objekt zurückgibt. Am Ende von `doGet()` wird mit diesem Remote-Objekt dann die Geschäftslogik aufgerufen, also die Methode `hallo()`.

Für den Einsatz des Servlets benötigen Sie noch einen Deployment-Deskriptor `web.xml`. In der hier nicht gezeigten Datei ist das Servlet so konfiguriert, dass es über die Adresse `/index.html` relativ zum Applikationskontext aufgerufen werden kann. Das Servlet und der Deployment-Deskriptor werden später zu einem Webapplikationsarchiv (`.war`) zusammengefasst.

Die EJB-Komponente und das Servlet werden nun zu einer Enterprise-Applikation verknüpft, wofür eine weitere Konfigurationsdatei nötig ist: der Enterprise-Applikations-Deployment-Deskriptor `application.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application
  PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.2//EN"
  'http://java.sun.com/j2ee/dtds/application_1_2.dtd'
<!-- CD/examples/ch11/HalloEJBWelt/application.xml -->
<application>
  <display-name>HalloEJBWelt</display-name>
  <module>
    <ejb>HalloEJBWelt.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>HalloEJBWelt.war</web-uri>
      <context-root>HalloEJBWelt</context-root>
    </web>
  </module>
</application>
```

Listing 11.6 Enterprise-Applikations-Deployment-Deskriptor `application.xml`

Innerhalb von `application.xml` legen Sie fest, aus welchen Modulen sich die Enterprise-Applikation zusammensetzt. Hier sind zwei `<module>`-Definitionen angegeben – ein `<ejb>`-Modul und ein `<web>`-Modul, die beide das jeweils nötige Archiv referenzieren. Beim Web-Modul ist zusätzlich noch angegeben, in welchem Kontext die Web-Applikation installiert werden soll, d.h. über welchen Pfad die Ressourcen in einer URL angesprochen werden.

JAR-Archiv, WAR-Archiv und der Deployment-Deskriptor werden als EAR-Archiv (Enterprise-Applikations-Archiv) verpackt, was das folgende **Ant-Buildfile** vollautomatisch durchführt:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- CD/examples/ch11/HalloEJBWelt/build.xml -->
<project name="HalloEJBWelt" default="build" basedir=".">
  <property name="product" value="\${ant.project.name}"/>
  <property name="jboss" location="/Library/JBoss/3.2"/>
  <property name="tomcat" location="/Library/Tomcat"/>

  <property name="src" location="."/>
  <property name="ejbsrc" location="\${src}/ejb"/>
  <property name="servletsrc" location="\${src}/servlet"/>
  <property name="build" location="build"/>
  <property name="ejbclasses" location="\${build}/ejbclasses"/>
  <property name="servletclasses"
            location="\${build}/servletclasses"/>
  <property name="jar" location="\${build}/\${product}.jar"/>
  <property name="war" location="\${build}/\${product}.war"/>
  <property name="ear" location="\${build}/\${product}.ear"/>
```

Zunächst werden die wichtigsten Pfade, der Projekt- und die Produktnamen festgelegt. Bei Bedarf ändern Sie einfach die fett gedruckten Stellen. Die üblichen Targets `init` und `clean`, die einfach nur einige Verzeichnisse anlegen bzw. die diversen Zwischenprodukte wieder löschen, sind hier nicht abgedruckt.

```
<target name="buildjar" depends="init">
  <javac srcdir="\${ejbsrc}" destdir="\${ejbclasses}">
    <classpath>
      <pathelement
        location="\${jboss}/client/jboss-j2ee.jar"/>
      <pathelement location="\${ejbclasses}"/>
    </classpath>
```

```

</javac>
<jar jarfile="\${jar}" basedir="\${ejbclasses}">
  <metainf dir="\${ejbsrc}" includes="ejb-jar.xml"/>
</jar>
</target>

```

Das Target `buildjar` übersetzt die Dateien der EJB-Komponente und fasst sie zu einem JAR-Archiv zusammen. Der Klassenpfad des Compilers verweist dabei neben dem Zielverzeichnis der EJB-Klassendateien auch auf eine Bibliothek innerhalb der JBoss-Installation, in der sich unter anderem das Paket `javax.ejb` befindet. Der `jar`-Task wird dann mithilfe des `metainf`-Unterelements so ausgeführt, dass der Deployment-Deskriptor `ejb-jar.xml` in das Verzeichnis `META-INF` innerhalb des JAR-Archivs kopiert wird.

```

<target name="buildwar" depends="init,buildjar">
  <javac srcdir="\${servletsrc}" destdir="\${servletclasses}">
    <classpath>
      <pathelement
        location="\${jboss}/client/jboss-j2ee.jar"/>
      <pathelement location="\${ejbclasses}"/>
      <pathelement
        location="\${tomcat}/common/lib/servlet.jar"/>
      <pathelement location="\${servletclasses}"/>
    </classpath>
  </javac>
  <war warfile="\${war}" webxml="\${servletsrc}/web.xml">
    <classes dir="\${servletclasses}"/>
  </war>
</target>

```

Das Target `buildwar` übersetzt die Servlet-Klasse und erzeugt daraus ein WAR-Archiv. Auch hier muss wieder der Klassenpfad des Compilers angepasst werden, der nicht nur das Paket `javax.ejb`, sondern auch das Paket `javax.servlet` kennen muss – daher wird zusätzlich zum JBoss-Archiv eine Tomcat-Bibliothek eingebunden. Um den Aufbau des Webarchivs kümmert sich der `war`-Task: Der angegebene Deployment-Deskriptor `web.xml` wird in das Verzeichnis `WEB-INF` gelegt, und die Klassendateien landen im Unterverzeichnis `classes`.

```

<target name="buildear" depends="buildjar,buildwar">
  <ear earfile="\${ear}" appxml="\${src}/application.xml">
    <fileset dir="\${build}" includes="*.jar,*.war"/>
  </ear>
</target>

```

```
</ear>
</target>
```

Auf den ersten beiden Targets baut nun das Target `buildear` auf – es erzeugt aus dem JAR- und dem WAR-Archiv sowie aus dem Deployment-Deskriptor `application.xml` ein EAR-Archiv. Dazu wird einfach der vorhandene `ear`-Task genutzt und passend konfiguriert.

```
<target name="build" depends="buildear">
  <copy file="{ear}" todir="{jboss}/server/default/deploy"/>
</target>
```

Zur Installation des EAR-Archivs innerhalb des J2EE-Servers kopieren Sie die Archivdatei einfach in das Verzeichnis `server/default/deploy/` innerhalb der JBoss-Installation. Diese Installation zum Einsatz einer Applikation wird auch **Deployment** genannt. Und weil Sie Ihre Applikation nach dem Übersetzen häufig sofort testen wollen, kopiert das `build`-Target das EAR-Archiv automatisch an die richtige Stelle.

Das Besondere an JBoss ist, dass es das `deploy`-Verzeichnis regelmäßig nach neuen oder geänderten Web- und Enterprise-Applikationen durchsucht und die gefundenen Applikationen automatisch zur Laufzeit installiert – man spricht von »Hot Deployment«. Sie müssen JBoss also nicht herunterfahren und neu starten, nur um eine neu kompilierte Applikation zu testen.

Ob Sie das EAR-Archiv nun per Hand in das `deploy`-Verzeichnis kopiert haben oder das Buildfile dies übernommen hat – nach ein paar Sekunden sollten Sie im JBoss-Terminal-Fenster in etwa folgende Statusmeldungen sehen:

```
19:07:00,077 INFO [MainDeployer] Starting deployment of package: file:/Library/JBoss/3.2/server/default/deploy/HalloEJBWelt.ear
19:07:00,091 INFO [EARDeployer] Init J2EE application: file:/Library/JBoss/3.2/server/default/deploy/HalloEJBWelt.ear
19:07:01,263 INFO [EjbModule] Deploying HalloEJBWelt
19:07:01,523 INFO [StatelessSessionInstancePool] Started jboss.j2ee:jndiName=HalloEJBWelt,plugin=pool,service=EJB
...
19:07:03,859 INFO [EARDeployer] Started J2EE application: file:/Library/JBoss/3.2/server/default/deploy/HalloEJBWelt.ear
19:07:03,862 INFO [MainDeployer] Deployed package: file:/Library/JBoss/3.2/server/default/deploy/HalloEJBWelt.ear
```


Damit ist Ihre Enterprise-Applikation bereit für den ersten Aufruf! Unter der Adresse <http://localhost:8080/HalloEJBWelt/index.html> sollten Sie ein mit Abbildung 11.3 vergleichbares Ergebnis vorfinden.



Abbildung 11.3 Test der Enterprise-Applikation im Web-Browser

Apple bietet im Dokument <http://developer.apple.com/internet/java/enterprise-java.html> einen weiteren Einstieg in die EJB-Programmierung – das dortige Beispiel realisiert eine Mailinglisten-Verwaltung mit einer CMP-Entity-Bean. Und auf der JBoss-Seite <http://www.jboss.org/docs/index#free-40x> erhalten Sie die »Getting Started«-Dokumentation mit weiteren Beispielen.

11.2.2 XDoclet

Bei größeren Projekten ist es ziemlich aufwändig, die zahlreichen Interfaces und Deployment-Deskriptoren für alle EJB-Komponenten von Hand zu pflegen. Daher verlagert die Idee des »attributorientierten Programmierens« alle Konfigurationsangaben als Metadaten in den eigentlichen Bean-Quelltext – so, wie Sie es von Javadoc kennen (siehe Abbildung 11.4).

Das Werkzeug »XDoclet« wertet diese Attribute aus und generiert daraus die Interface- und Deskriptor-Dateien. XDoclet ist als Task in Ant integrierbar – wenn Sie also XDoclet im Buildfile jedes Mal vor dem Kompilieren aufrufen, sind die generierten Dateien immer aktuell. Der große Vorteil dieses Ansatzes ist, dass Sie pro EJB-Komponente nur noch eine Datei pflegen müssen.

Wenn Sie Apples Applikations-Server-Paket installiert haben, finden Sie XDoclet im Verzeichnis `/Developer/Java/J2EE/XDoclet/`. Die aktuellste Version können Sie sich von <http://xdoclet.sourceforge.net/> herunterladen, dort finden Sie auch weiterführende Dokumentationen. Als Einstieg kann Ihnen auch ein Xcode-Projekt dienen, das mit einem der drei J2EE-Projekttypen erzeugt wurde – diese Typen verwenden Ant und XDoclet für den Build-Prozess.

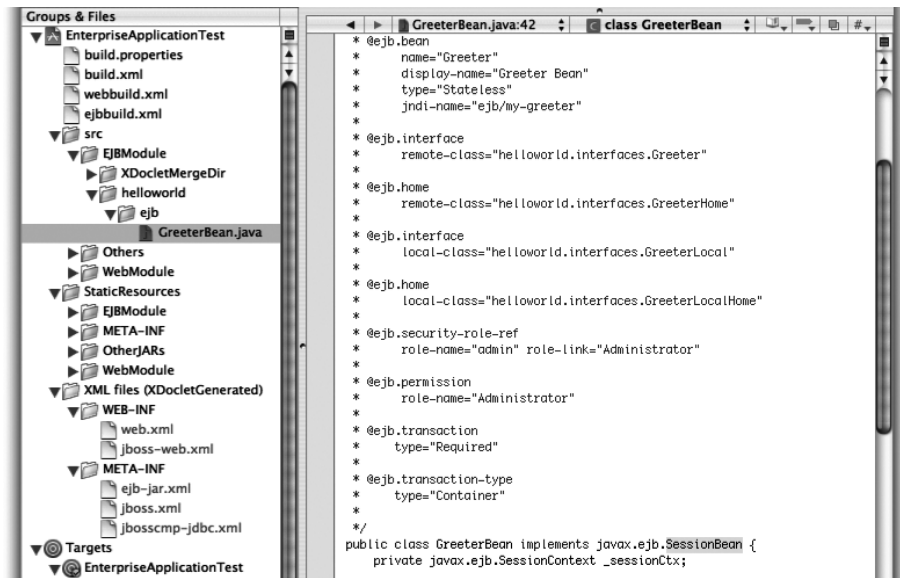


Abbildung 11.4 XDoclet in einem der Xcode-J2EE-Projekttypen

11.3 Literatur & Links

- ▶ Monson-Haefel/Burke/Labourey, »Enterprise JavaBeans« 4th ed., O'Reilly 2004
Der »Monson-Haefel« wird oft als das EJB-Standardwerk bezeichnet. Sie erhalten fundierte Informationen, auch zu den verschiedenen EJB-Versionen – allerdings kommt die Anbindung an andere J2EE-Technologien etwas kurz, und der Stil ist eher technisch-trocken. Enthält das »JBoss Workbook«.
- ▶ M. Kompf, »Enterprise JavaBeans 2.1«, Galileo Computing 2004
Ein praxisorientiertes Buch, das Sie sowohl mit als auch ohne EJB-Vorkenntnisse lesen können
- ▶ Backschat/Edlich, »J2EE-Entwicklung mit Open-Source-Tools«, Spektrum 2004
Ein eher theoretisches Buch zu den J2EE-Technologien und ihren zahlreichen Implementierungen – mit guten Vergleichen der Vor- und Nachteile der diversen Produkte. Enthält auch eine XDoclet-Beschreibung.
- ▶ Eickstädt/Reuhl, »J2EE mit Struts & Co. », Markt+Technik 2004
Wie das voranstehende Buch bietet dieses Werk einen guten Überblick über die zahlreichen J2EE-Produkte. Es werden zwar etwas weniger Themen besprochen, diese dafür aber ausführlicher und praxisorientierter.

- ▶ T. Langner, »Verteilte Anwendungen mit Java«, Markt+Technik 2002
Dieses Buch behandelt alle üblichen J2EE-Themen und legt den Fokus auf die Kommunikationsschicht in verteilten Systemen. Entsprechend ist ein ausführliches Kapitel über CORBA enthalten.
- ▶ Johnson/Hoeller, »J2EE Development without EJB«, Wrox 2004
Dieses Buch erklärt hervorragend, warum EJBs nicht immer die beste Lösung für ein Enterprise-Problem sind – und welche Technologien alternativ sinnvoll sind.
- ▶ J. Marker, »WebObjects 5 for Mac OS X«, Peachpit 2004
Wenn Sie sich gerne anhand ausführlich beschriebener Abbildungen in ein neues Gebiet einarbeiten, bietet dieses Buch einen hervorragenden schnellen Einstieg in Apples teilweise J2EE-kompatible WebObjects-Technologie.
- ▶ <http://java.sun.com/j2ee/>
Suns Portalseite zu J2EE mit allen Spezifikationen und zahlreichen Downloads – wenn auch leider oft nicht für Mac OS X
- ▶ <http://java.sun.com/products/ejb/>
Informationen und Spezifikationen speziell zu EJB
- ▶ <http://developer.apple.com/java/faq/j2ee.html>
Apple beantwortet einige Fragen zu Apples J2EE-Strategie.
- ▶ <http://www.apple.com/server/documentation/>
Eine Liste der verfügbaren Mac OS X Server-Dokumentationen. Für Java und J2EE ist insbesondere das Dokument »Java Application Server Administration« interessant.

12 J2ME und MIDP

12.1	MIDlets entwickeln und testen	485
12.2	Einsatz im mobilen Endgerät	496
12.3	Benutzungsoberflächen und Grafik	503
12.4	Literatur & Links	508

1	Grundlagen
2	Entwicklungsumgebungen
3	Grafische Benutzungsoberflächen (GUI)
4	Ausführbare Programme
5	Portable Programmierung
6	Mac OS X-spezielle Programmierung
7	Grafik und Multimedia
8	Werkzeuge
9	Datenbanken und JDBC
10	Servlets und JavaServer Pages (JSP)
11	J2EE und Enterprise JavaBeans (EJB)
12	J2ME und MIDP
A	Kurzeinführung in die Programmiersprache Java
B	Java auf Mac OS 8/9/Classic
C	Java 1.5 »Tiger«
D	System-Properties
E	VM-Optionen
F	Xcode- und Project Builder-Einstellungen
G	Mac OS X- und Java-Versionen
H	Glossar
I	Die Buch-CD

12 J2ME und MIDP

*»Am Heimcomputer sitz« ich hier, programmier« die Zukunft mir.«
(Kraftwerk)*

Sie haben bisher zwei Säulen der »Java 2 Platform« kennen gelernt: J2SE als Grundlage der Java-Entwicklung, vor allem für Desktop-Software, und J2EE für Server-Anwendungen. Beide Typen können Sie auf Ihrem Mac laufen lassen, d.h., der Rechner wird entweder von einem Endanwender benutzt, oder er führt als Server beispielsweise Webapplikationen aus. Sun hat noch eine dritte Säule für Java definiert – J2ME, die »Java 2 Platform Micro Edition« für Handys und andere mobile Geräte. Die Software dafür kann zwar auf dem Mac entwickelt und getestet werden, läuft dann aber letztendlich auf einem anderen System. In diesem Kapitel geht es also nicht um Java-Programmierung *für* Mac OS X, sondern um Java-Programmierung *auf* Mac OS X.

J2ME ist als separate Technologie nötig, denn Handys und Organizer (PDAs, Handhelds) stehen vor einem großen Problem, wenn sie Java-Programme ausführen sollen: Sie haben nur stark eingeschränkte Ressourcen zur Verfügung (was Sie vielleicht noch von Ihrem Heimcomputer kennen, den Sie vor zwanzig Jahren programmiert haben). Der Hauptspeicher ist knapp, der Hintergrundspeicher zur dauerhaften Speicherung ebenso, das Display ist sehr klein (falls überhaupt eines vorhanden ist!), es gibt keine einheitliche Bedienungsschnittstelle. Netzwerkverbindungen sind teuer oder langsam, die Akkus haben keine unbegrenzte Kapazität und die Prozessoren arbeiten recht gemächlich. Das normale J2SE mit seinem Speicher- und Geschwindigkeitshunger und der riesigen Klassenbibliothek ist also keine Option.

J2ME ist dabei kein Produkt, sondern eine Sammlung von Spezifikationen für mobile Endgeräte – diese sind immer noch zu unterschiedlich, um alle dieselbe Java-Umgebung ausführen zu können. Denken Sie nur an Java auf Chipkarten, verglichen mit Java auf Highend-PDAs. Daher definiert J2ME verschiedene **Konfigurationen**, die ein Minimum an Funktionalität für einen bestimmten Typ Endgerät festlegen. Derzeit gibt es folgende zwei Konfigurationen:

► **Connected Limited Device Configuration (CLDC)**

CLDC ist die Spezifikation für Lowend-Geräte, d.h. Handys oder Organizer mit wenig Speicher (üblicherweise bis 512 KByte) und eingeschränkter Netzwerkverbindung.

► **Connected Device Configuration (CDC)**

CDC deckt Geräte mit mehr Fähigkeiten ab, die aber dennoch für J2SE noch nicht ausreichend sind. Die Geräte haben typischerweise 2 MByte oder

mehr Speicher, schnellere Prozessoren, und können meistens auf eine dauerhafte Netzwerkverbindung zurückgreifen. In diese Kategorie fallen Highend-PDAs, SmartPhones und Set-Top-Boxen.

Wenn man sich die Ausstattung heutiger Handys ansieht, fallen diese eher in den Bereich CDC, und einige Geräte unterstützen tatsächlich beide Konfigurationen. Allerdings ist CLDC deutlich stärker verbreitet, daher wird im Folgenden nur diese Konfiguration besprochen.

Da die normale Java Virtual Machine (JVM) bei CLDC nicht genügend Ressourcen vorfindet, bringt diese Konfiguration eine eigene Virtual Machine mit: die **KVM** (kompakte oder Kilobyte Virtual Machine). Sie besitzt folgende Einschränkungen gegenüber der JVM:

- ▶ Es gibt weniger Bytecode-Instruktionen. Insbesondere sind Fließkommaoperationen für die KVM nicht verpflichtend und stehen dementsprechend auch oft nicht zur Verfügung! Die Verwendung von `float` und `double` führt zu Laufzeitfehlern, `java.lang.Float` und `java.lang.Double` gibt es erst gar nicht.
- ▶ Die Reflection-Klassen sind nicht vorhanden, auch die entsprechenden Methoden in `java.lang.Class` fehlen. `Class.forName()` kann allerdings genutzt werden.
- ▶ `Object.finalize()` und `clone()` gibt es nicht.
- ▶ Threads werden unterstützt, mit Ausnahme von Dämon-Threads und Thread-Gruppen.
- ▶ Auch die Exception-Behandlung erfolgt weitestgehend wie gewohnt. Allerdings können Errors nicht abgefangen werden, sondern führen zur sofortigen Beendigung des Programms.

Während die Konfiguration also die grundlegenden Eigenschaften der Virtual Machine beschreibt, legt sie beispielsweise nicht fest, welche Ein- und Ausgabegeräte es gibt. Dafür definiert J2ME diverse **Profile**, die auf einer Konfiguration aufbauen und diese ergänzen. Für CLDC gibt es z.B. PDAP, das »Personal Digital Assistant Profile«, oder MIDP, das »Mobile Information Device Profile«, das von den meisten Handys und – obwohl es PDAP gibt – von vielen PDAs (vor allem von PalmOS-Geräten) verwendet wird.

MIDP ergänzt CLDC um Klassen für persistente Speicherung, Netzwerkzugriffe und vor allem um Klassen für die Oberflächenprogrammierung. Hierbei werden nicht die AWT- oder Swing-Klassen verwendet, denn abgesehen vom Speicherbedarf sind diese Komponenten für Display-Größen ab 96x54 Pixel eher ungeeignet. Im Folgenden werden also MIDP-Applikationen entwickelt, die

auch **MIDlets** genannt werden. Abbildung 12.1 zeigt noch einmal die zur Ausführung von MIDlets nötige Infrastruktur.

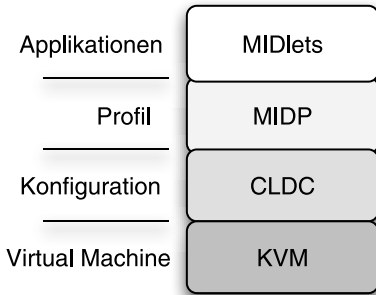


Abbildung 12.1 J2ME-Struktur für MIDP-Applikationen

Die spezielle Klassenbibliothek für die MIDlet-Programmierung befindet sich vor allem in Unterpaketen von `javax.microedition`, dazu stehen in den Paketen `java.lang`, `java.util` und `java.io` ausgewählte Klassen aus den entsprechenden J2SE-Paketen zur Verfügung.

Sun selbst gibt auf der Seite <http://developers.sun.com/techttopics/mobility/midp/articles/intro/> eine kurze Programmierereinführung in MIDP. In den folgenden Abschnitten sehen Sie die Entwicklung und Anwendung von MIDlets speziell auf Mac OS X.

12.1 MIDlets entwickeln und testen

Zum Entwickeln und Testen von MIDlets benötigen Sie eine MIDP-Implementation, die Ihnen die nötigen Klassenbibliotheken und Werkzeuge zur Verfügung stellt. Sun bietet dafür das »J2ME Wireless Toolkit« an, derzeit allerdings nur für Windows, Linux und Solaris. Glücklicherweise wurde aber Suns MIDP-Referenz-Implementation (RI) in der Version 1.03 auf Mac OS X portiert. MIDP ist zwar schon bei Version 2.0 angekommen, aber da Sie im realen Einsatz überwiegend Version 1.0 antreffen werden, stellt dies keine allzu große Einschränkung dar.

Laden Sie sich also von <http://mpowers.net/midp-osx/> das Archiv für MIDP 1.03 herunter und packen Sie es aus. Für die weiteren Beispiele wird angenommen, dass Sie die Dateien in Ihrem Benutzerverzeichnis in `~/midp/` ablegen. Dort sollten Sie nun ungefähr folgende Verzeichnisstruktur vorfinden:

```
/Users/benutzername  
  midp
```

```
bin
classes
docs
examples
```

Natürlich können Sie die Dateien auch irgendwo anders ablegen, beispielsweise in `/usr/local/`, dann müssen Sie im Folgenden einfach nur die Pfade korrekt anpassen. In den Unterverzeichnissen finden Sie u.a. einige kleine MIDlet-Beispiele von Sun sowie die MIDP-API-Dokumentation.

Als weitere Voraussetzung muss auf Ihrem Rechner das X Window System (X11) laufen, das der MIDP-Emulator zur Darstellung der MIDlets benutzt. Mit diesem Emulator testen Sie die MIDlets auf Ihrem Mac, damit Sie nicht jedes Mal die Daten in das reale Gerät übertragen müssen (das Sie vielleicht gar nicht besitzen). Das Gute an X11 ist: Es ist Bestandteil von Mac OS X 10.3! Allerdings wird diese Komponente nur installiert, wenn Sie eine benutzerdefinierte Systeminstallation (mit der Schaltfläche »Anpassen« im Installer) durchführen und X11 explizit auswählen. Wenn Sie das vergessen haben, finden Sie auf der dritten Installations-CD von MacOS X 10.3 im Verzeichnis `Packages` das Installationspaket `X11User.pkg`, mit dem Sie X11 nachträglich ins System einspielen können – oder Sie laden sich das Archiv von <http://www.apple.com/macosx/x11/> herunter. Nach erfolgreicher Installation finden Sie das Programm X11 im Verzeichnis `/Programme/Dienstprogramme/`. Starten Sie das Programm, und lassen Sie es einfach laufen. Das X11-Terminal-Fenster, das sich nach kurzer Zeit öffnet, können Sie schließen.

Jetzt können Sie die MIDP-Installation testen. Öffnen Sie dazu ein Terminal-Fenster (oder verwenden Sie das X11-Terminal), gehen Sie in das Verzeichnis `midp/` und rufen Sie dort ein Demoskript mit folgendem Befehl auf:

```
sh demos.sh
```

Als Ergebnis sollten Sie in etwa Abbildung 12.2 sehen. Der MIDP-Emulator stellt eine komplette Handy-Oberfläche dar, in dessen Display einige MIDlets zur Ausführung angeboten werden. Der Emulator wird wie ein normales Handy bedient – Sie müssen mit dem Mauszeiger also auf die Handy-Tasten klicken und nicht etwa direkt ins Display. Beendet wird der Emulator mit der Auflegen-Taste oder mit dem Ausschalt-Knopf oben links am Handy.

Nun können Sie sich um die eigentliche Programmierung kümmern. Oft wird für den Build-Prozess eines MIDlets ein Ant-Skript eingesetzt, wie dies z.B. auf <http://developers.sun.com/techtopics/mobility/midp/articles/ant/> oder speziell für Mac OS X auf <http://developers.sun.com/techtopics/mobility/midp/articles/osx/> beschrieben ist. Ant und die Integration eines Ant-Skripts in Xcode haben Sie

bereits in Kapitel 8, *Werkzeuge*, kennen gelernt, daher wird Ihnen hier vorgestellt, wie Sie MIDlets direkt mit Xcode erzeugen.



Abbildung 12.2 MIDP-Emulator mit einigen Demo-MIDlets

Legen Sie das neue Projekt in Xcode als »Java Tool« an, denn das bringt bereits alles mit, was Sie für ein MIDlet benötigen: ein JAR-Archiv als Produkt und ein dazugehöriges Manifest. Als Einstiegsbeispiel können Sie das folgende kurze, aber voll funktionsfähige MIDlet ausprobieren:

```
//CD/examples/ch13/HalloMIDP/HalloMIDP.java
package com.muchsoft.macjava.midp;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HalloMIDP extends MIDlet {
    private Form form = new Form( "Hallo J2ME!" );

    public HalloMIDP() {
        form.append( "System-Properties:\n" );

        String[] props = { "microedition.configuration",
                           "microedition.profiles",
                           "microedition.platform",
```

```

        "microedition.encoding",
        "microedition.locale" );

    for (int i = 0; i < props.length; i++) {
        form.append( props[i] + ": "
            + System.getProperty(props[i]) + "\n" );
    }
}

public void startApp() {
    Display.getDisplay(this).setCurrent( form );
}

public void pauseApp() { }
public void destroyApp(boolean unconditional) { }
}

```

Listing 12.1 Eine einfache MIDP-Applikation (MIDlet)

MIDlets müssen von der Klasse `javax.microedition.midlet.MIDlet` erben, und da die Ausgabe auf einem ziemlich kleinen Display erfolgt, wird die MIDP-Oberflächenbibliothek aus dem Paket `javax.microedition.lcdui` eingebunden. Als Oberflächenkomponente wird hier einzig die Klasse `Form` als Formular-Container verwendet. `Container` und `Component` wie bei AWT und Swing gibt es allerdings nicht, diese sind grob vergleichbar mit den LCDUI-Klassen `Displayable` und `Item`.

Ein MIDlet wird von der MIDP-Umgebung erzeugt und mit dem Standardkonstruktor initialisiert, wenn der Anwender das Programm startet. Hier wird im Konstruktor das Formular zusammengebaut – ein Dialog mit einem Titel (»Hallo J2ME!«) und einigen Zeichenketten als Inhalt.

Das Programm soll alle System-Properties samt Wert anzeigen. Standardmäßig vorhanden sind aber nur die fünf hier angegebenen Properties (»microedition.configuration« usw.). Leider gibt es bei MIDP die Klasse `java.util.Properties` nicht, daher können Sie nicht einfach alle vorhandenen Properties abfragen. Sie müssen also alle Property-Namen fest kodieren, auch eventuell vorhandene herstellereigenspezifische.

Mit `append()` werden zu dem `Form`-Objekt dann die Zeichenketten hinzugefügt. Intern kann ein Formular aber nur `Item`-Referenzen speichern, daher werden aus den Zeichenketten beim Hinzufügen intern `StringItem`-Objekte erzeugt. Ein Zeilenumbruch wird wie gewohnt mit `\n` kodiert, auch wenn der

MIDP-Emulator sowieso jedes `StringItem` in einer neuen Zeile darstellt – bei realen Geräten ist das leider nicht immer der Fall.

`MIDlet` ist eine abstrakte Klasse, Unterklassen müssen dadurch die Methoden `startApp()`, `pauseApp()` und `destroyApp()` implementieren. Zusammen mit dem Konstruktor entsprechen diese Methoden ungefähr den Lebenszyklus-Methoden von Java Applets. Nach der Erzeugung und dem Durchlaufen des Konstruktors befindet sich das `MIDlet` zunächst im Pause-Zustand. Danach wird es von der MIDP-Umgebung in den aktiven Zustand versetzt, und `startApp()` wird aufgerufen. Wenn das `MIDlet` unterbrochen werden muss, beispielsweise weil der Anwender einen Anruf entgegennimmt, wird `pauseApp()` aufgerufen und das `MIDlet` wieder in den Pause-Zustand versetzt. Hier sollten Sie nach Möglichkeit alle nicht benötigten Ressourcen freigeben – denken Sie immer daran, Speicher ist knapp. Nach Beendigung des Gesprächs wird das `MIDlet` dann weiter ausgeführt und wiederum `startApp()` aufgerufen, wo Sie die Ressourcen erneut anfordern können. Der dortige Code muss also sowohl für die erste Initialisierung als auch für beliebig viele Reinitialisierungen ausgelegt sein!

Wenn das Gerät das `MIDlet` beenden will, wird `destroyApp()` aufgerufen. Ist der `boolean`-Parameter `true`, wird das `MIDlet` danach auf jeden Fall gelöscht. Ansonsten kann das `MIDlet` eine `javax.microedition.midlet.MIDlet-StateChangeException` werfen, um das Beenden abzulehnen. Mit derselben `Exception` in `startApp()` kann das `MIDlet` auch das Starten verhindern. Tritt dagegen irgendeine Ausnahme in `pauseApp()` auf, wird das `MIDlet` sofort von der MIDP-Umgebung beendet.

Ein `MIDlet` kann seinen Zustand selbst wechseln. Mit der geerbten Methode `notifyPaused()` informiert es die Umgebung, dass es sich nun im Pause-Zustand befindet (`pauseApp()` wird dann nicht mehr automatisch aufgerufen). Mit `resumeRequest()` kann es anfordern, in den aktiven Zustand versetzt zu werden, woraufhin die MIDP-Umgebung kurze Zeit später `startApp()` aufrufen wird. Und mit `notifyDestroyed()` schließlich kann das `MIDlet` bekannt geben, dass es sich selbst in den Ende-Zustand versetzt hat – ab diesem Moment kann die Umgebung sämtliche `MIDlet`-Ressourcen löschen! Üblicherweise ruft das `MIDlet` vorher selbst noch `destroyApp()` auf, um dort eventuell vorhandenen Aufräum-Code auszuführen.

In diesem Beispiel passiert in `startApp()` nicht mehr, als das Formular-Objekt auf dem `Display` anzuzeigen. Dazu wird zunächst mit `Display.getDisplay(this)` der so genannte `Display`-Kontext für dieses `MIDlet` erfragt. Jedem `MIDlet` ist genau ein Kontext zugeordnet, der immer nur einen Dialog zur selben Zeit darstellen kann. Mit `setCurrent()` wird dann das `Form`-Objekt

zum aktuell angezeigten Dialog gemacht. Diese Zeile findet sich häufig schon im Konstruktor, und die meisten Geräte akzeptieren das auch. Die MIDP-Spezifikation sagt aber ausdrücklich, dass auf `Display` erst in `startApp()` zugegriffen werden darf (und ab dann solange, bis das MIDlet zerstört ist), daher wird hier der ganz korrekte Weg gegangen.

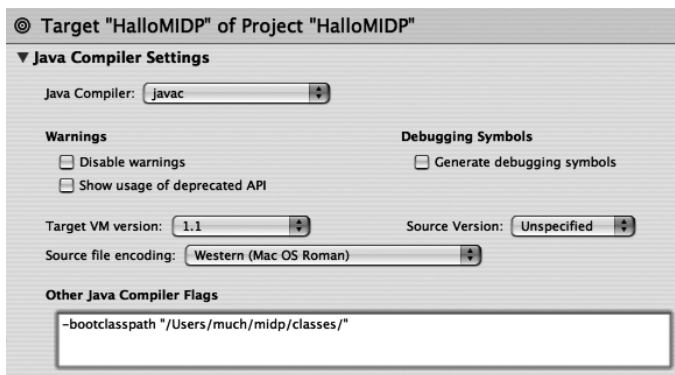


Abbildung 12.3 Java Compiler Settings

Zum Übersetzen dieses Quelltextes in Xcode müssen Sie noch die Target-Einstellungen anpassen, denn MIDlets sind keine J2SE-Anwendungen, für die Projekt-Schablonen wie »Java Tool« ausgelegt sind (siehe Abbildung 12.3). Im Bereich »Java Compiler Settings« müssen Sie als »Target VM version« den Eintrag »1.1« auswählen, weil die KVM mit neueren Klassendatei-Formaten nicht zurechtkommt. Im Feld »Other Java Compiler Flags« geben Sie dann mit der VM-Option `-bootclasspath` das `classes`-Verzeichnis in Ihrem `midp`-Ordner an. Damit legen Sie die Standard-Klassenbibliothek fest – Sie ersetzen also die umfangreiche J2SE-Bibliothek durch die kleine, eingeschränkte Bibliothek von J2ME. »Generate debugging symbols« sollten Sie besser ausschalten, weil Sie dadurch kleinere Klassendateien erhalten – und bei knappem Speicher ist das viel wert! Wenn Sie nun noch im Bereich »Java Archive Settings« den »Product type« auf »Java Archive« stellen (dies sollte die Standardvorgabe sein), kann der normale Java-Compiler Ihre MIDP-Applikation übersetzen.

Bevor Sie das nun aber tatsächlich testen, müssen Sie zu dem Build-Prozess noch einen weiteren Schritt hinzufügen. Alle MIDlets müssen vor dem Einsatz im Gerät eine Vorprüfung auf dem Entwicklungsrechner mit dem Programm `preverify` durchlaufen. Dadurch wird die KVM entlastet, die sonst – wie die JVM – beim Laden einer Klasse aufwändig die Integrität des Bytecodes prüfen müsste. Und damit geprüfte Klassen anschließend nicht verfälscht werden können, werden sie bei diesem Vorgang auch gleich noch automatisch signiert.

Machen Sie dafür in den Target-Einstellungen einen **Ctrl**+Klick auf »Build-Phases« und wählen Sie im Kontext-Popup-Menü »New Build Phase« und danach »New Shell Script Build Phase« aus. Ziehen Sie die Build-Phase mit der Maus an die passende Stelle, falls sie sich dort noch nicht befindet: zwischen »Sources«, wo die Java-Quelltexte übersetzt werden, und »Java Resources«, wo alle zusätzlichen Dateien in das temporäre Übersetzungsverzeichnis kopiert werden (siehe Abbildung 12.4).

Als Shell können Sie `/bin/sh` eingetragen lassen, in das Feld darunter fügen Sie folgendes Kommando zur Ausführung ein (als eine Zeile ohne Zeilenumbruch!):

```
/Users/much/midp/bin/preverify -classpath "/Users/much/midp/classes/" -d ${TEMP_DIR}/preverified ${CLASS_FILE_DIR} && cp -R ${TEMP_DIR}/preverified/* ${CLASS_FILE_DIR}
```

Die beiden Pfadangaben, die das `midp`-Verzeichnis beinhalten, müssen Sie natürlich an Ihre Installation anpassen. Sinn dieses Kommandos ist es, das `preverify`-Programm aus der MIDP-Referenz-Implementation aufzurufen, um alle gerade kompilierten Klassen, die sich im Verzeichnis `${CLASS_FILE_DIR}` befinden, zu überprüfen. `preverify` kann die überprüften Klassen aber nicht wieder in das Quellverzeichnis schreiben, daher wird mit der Option `-d` als Zielverzeichnis der Ordner `preverified` im temporären Verzeichnis `${TEMP_DIR}` gesetzt. Nach erfolgreicher Überprüfung werden dann einfach alle Klassen aus dem `preverified`-Verzeichnis zurück nach `${CLASS_FILE_DIR}` kopiert. Dazu wird an den ersten Befehl mit `&&` ein Kopier-Befehl (`cp`) angehängt.

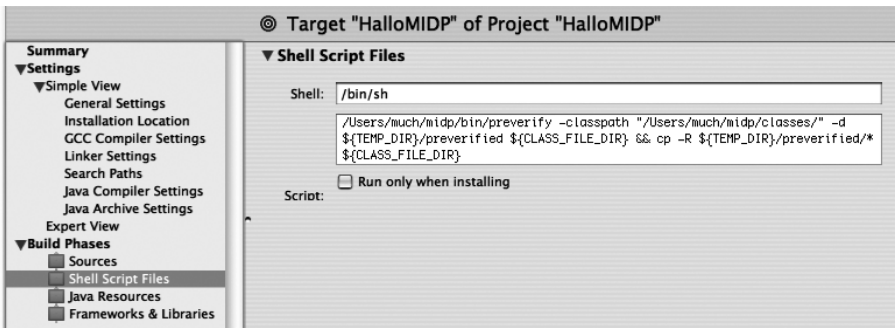


Abbildung 12.4 Neue Build-Phase: »Shell Script Files«

Die letzte Phase »Frameworks & Libraries«, in der das JAR-Archiv erzeugt wird, lassen Sie einfach unverändert bestehen.

Damit das Programm vom MIDP-Gerät gestartet werden kann, müssen Sie nun noch die Manifest-Datei anpassen, mit der unter anderem die MIDlet-Hauptklasse festgelegt wird. Das Manifest erfüllt damit einen ähnlichen Zweck wie bei JAR-Archiven, allerdings sind die Einträge ganz speziell für MIDP-Applikationen definiert. Für das Beispiel-MIDlet sieht das Manifest wie folgt aus:

```
MIDlet-1: HalloMIDP, /
hallo.png, com.muchsoft.macjava.midp.HalloMIDP
MIDlet-Name: HalloMIDP
MIDlet-Version: 1.0.0
MIDlet-Vendor: Thomas Much
MIDlet-Icon: /hallo.png
MIDlet-Info-URL: http://www.muchsoft.com/java/
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

Listing 12.2 MIDP-JAR-Manifest

Folgende Einträge müssen im Manifest zwingend vorhanden sein:

- ▶ **MIDlet-Name**
Ein JAR-Archiv kann mehrere MIDlets enthalten. Eine solche Zusammenfassung von MIDlets nennt man »MIDlet-Suite«, und dieser Eintrag legt den Namen der Suite als Information für den Anwender fest. Wenn Ihr JAR-Archiv nur ein MIDlet enthält, wird als Suite-Name üblicherweise derselbe Name wie beim MIDlet verwendet.
- ▶ **MIDlet-Version**
Legt die Version des MIDlets mit bis zu drei Stellen fest. Damit kann das Gerät beim Hochladen der Software entscheiden, ob das MIDlet eine eventuell vorhandene ältere Version ersetzt.
- ▶ **MIDlet-Vendor**
Gibt den Hersteller des MIDlets an.
- ▶ **MIDlet-*n***
Für jedes MIDlet in einer MIDlet-Suite (also im JAR-Archiv) gibt es einen solchen Eintrag, wobei *n* bei 1 startet und die MIDlets dann fortlaufend durchnummeriert werden. Als erster Wert wird der Name des MIDlets angegeben, danach ein MIDlet-Symbol im PNG-Format, am Ende der vollqualifizierte Klassenname. PNG (»Portable Network Graphics«, ein Ersatz für das lizenzpflichtige GIF-Format) ist das einzige Bildformat, das MIDP-Umgebungen unterstützen müssen. Das Symbol ist optional – wenn Sie kein Bild angeben wollen, lassen Sie diesen Platz einfach leer (es folgen dann zwei Kommas direkt aufeinander).

- ▶ `MicroEdition-Configuration`
Legt die J2ME-Konfiguration fest, die das MIDlet benötigt. Derzeit ist dies eigentlich immer »CLDC-1.0«.
- ▶ `MicroEdition-Profile`
Legt das CLDC-Profil fest, welches das MIDlet benötigt. Derzeit ist dies fast immer »MIDP-1.0«.

Ergänzend können folgende optionale Einträge auftauchen:

- ▶ `MIDlet-Description`
Als Information für den Benutzer können Sie hier eine beliebige Beschreibung der MIDlet-Suite eintragen.
- ▶ `MIDlet-Icon`
Zusätzlich zu jedem einzelnen MIDlet können Sie auch der gesamten MIDlet-Suite ein eigenes Symbol verpassen.
- ▶ `MIDlet-Info-URL`
Zeigt dem Anwender an, wo er weitere Informationen über die MIDlet-Suite im Web finden kann.
- ▶ `MIDlet-Data-Size`
Mit diesem Eintrag können Sie festlegen, wieviel dauerhaften (persistenten) Speicher in Bytes Ihr MIDlet benötigt. Fehlt diese Angabe, wird angenommen, dass das MIDlet keinerlei dauerhaften Speicher belegt. Ob das MIDlet mehr Speicher anfordern kann, ist geräteabhängig. Die Verwaltung des persistenten Speichers erfolgt bei MIDP mit dem »Record Management System« (RMS) aus dem Paket `javax.microedition.rms`.

Das Manifest dient also zur Beschreibung eines oder mehrerer MIDlets. Der Benutzer wird anhand dieser Daten entscheiden, ob er das MIDlet auf seinem Gerät installieren möchte. Beim mobilen Einsatz mit langsamen und teuren Netzwerkverbindungen ist dies aber ein Problem: Der Anwender muss erst das gesamte JAR-Archiv herunterladen, nur um dann vielleicht festzustellen, dass darin nicht die gesuchten Programme enthalten sind. Daher kann die Beschreibung eines MIDlets zusätzlich auch in einer so genannten JAD-Datei (mit der Dateinamenserweiterung `.jad`) erfolgen, die separat vom JAR-Archiv angeboten wird. In der JAD-Datei sind nur die relevanten Einträge für das zugehörige MIDlet enthalten, Informationen zur Suite und der Programmcode fehlen – die Datei kann also schnell geladen werden.

Alle optionalen Manifest-Einträge können Sie auch bei der JAD-Datei verwenden, wobei die Einträge aber natürlich nicht für die Suite, sondern für das einzelne MIDlet gelten. Folgende Einträge müssen zwingend vorhanden sein:

- ▶ MIDlet-Name
- ▶ MIDlet-Version
- ▶ MIDlet-Vendor
- ▶ MIDlet-Jar-URL

Dieser Eintrag gibt die URL an, wo das JAR-Archiv heruntergeladen werden kann. Wenn sich das JAR-Archiv beim lokalen Einsatz im selben Verzeichnis befindet, steht hier einfach der Archivname, ansonsten muss immer eine komplette Web-URL angegeben werden.

- ▶ MIDlet-Jar-Size

Gibt die exakte Größe des JAR-Archivs in Byte an.

Im Manifest und in den JAD-Dateien können übrigens noch beliebige weitere Einträge im Format *name:wert* auftauchen. Dies kann zur individuellen Konfiguration verwendet werden, indem Sie diese Werte im MIDlet mit der geerbten Methode `getAppProperty()` auswerten. Leider funktioniert das Auslesen nur bei Einträgen in den JAD-Dateien, obwohl es eigentlich auch für das Manifest spezifiziert ist.

Nun ist der Build-Prozess vollständig, und das Übersetzen des MIDlets sollte fehlerfrei funktionieren! Falls Sie dennoch Fehler gemeldet bekommen, können Sie sich das ausführliche Build-Protokoll (Build Transcript) anzeigen lassen, in dem beispielsweise auch Fehler aus der Vorprüfung mit `preverify` auftauchen. Rufen Sie dazu den Menüpunkt **Build · Show Detailed Build Results** auf. Am unteren Fensterrand besitzt der Dialog einen »Greifer«, den Sie nach oben ziehen können – oder Sie klicken auf den Text-Knopf rechts neben »Show Warnings« (siehe Abbildung 12.5).

Das erzeugte JAR-Archiv, das Sie bei Xcode wie immer im `build`-Verzeichnis des Projektes finden, kann nun auf einem MIDP-fähigen Gerät eingesetzt werden! Sie können das MIDlet aber auch direkt aus Xcode heraus im MIDP-Emulator testen, den Sie schon ganz zu Anfang beim Test der MIDP-Installation kennen gelernt haben. Dazu müssen Sie das erste Executable wie in Abbildung 12.6 anpassen.

Bei Suns Wireless Toolkit (WTK) heißt der Emulator ganz einfach `emulator`, bei der Mac OS X-Portierung der Referenz-Implementation wurde das Programm `midp` genannt – den Pfad darauf müssen Sie bei »Path to Executable« eintragen. Als »Launch Arguments« übergeben Sie mit `-classpath` das JAR-Archiv und dahinter den vollqualifizierten Klassennamen des MIDlets.

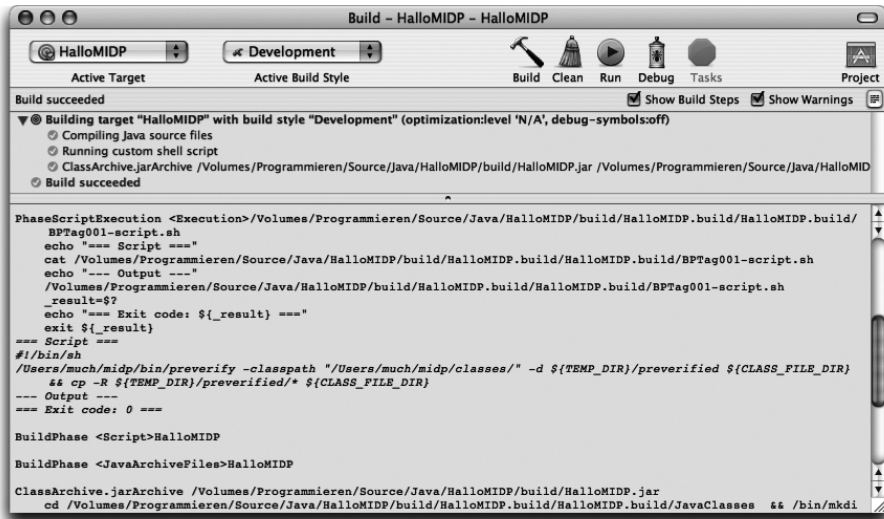


Abbildung 12.5 Ausführliches Build-Protokoll

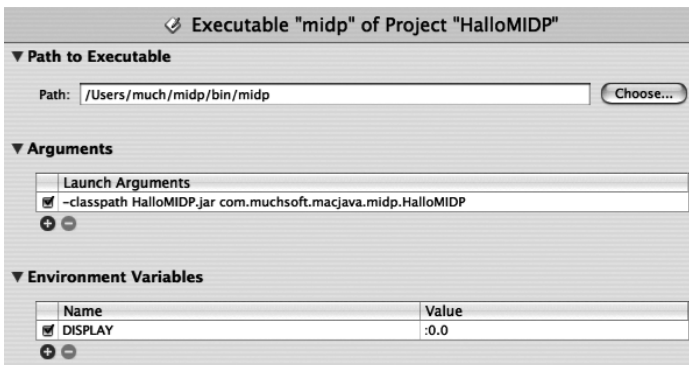


Abbildung 12.6 Executable an den MIDP-Emulator anpassen

Wichtig ist noch, dass Sie bei »Environment Variables« die Umgebungsvariable `DISPLAY` mit dem Wert `:0.0` eintragen. Der MIDP-Emulator verwendet ja zur Darstellung das X Window System (X11), was im Normalfall dazu verwendet wird, um grafische Ausgaben über ein Netzwerk zu transportieren. Mit diesem Wert der Umgebungsvariablen bringen Sie X11 dazu, die Daten auf Ihrem lokalen Rechner auszugeben.

Wenn Sie nun **Build · Build and Run** anwählen, wird nach dem Erzeugen des JAR-Archivs automatisch der MIDP-Emulator aufgerufen und sollte Ihnen das MIDlet wie in Abbildung 12.7 anzeigen.



Abbildung 12.7 Das erste eigene MIDlet im MIDP-Emulator

Nachteil der MIDP-Entwicklung auf Mac OS X ist, dass dafür nur die eben gesehene Referenz-Implementation zur Verfügung steht. Die meisten Handy-Hersteller bieten darauf aufbauend eigene Versionen an, die die jeweiligen Handys viel spezieller unterstützen – bis hin zur 1:1-Fehlerkompatibilität mit der Handy-Firmware. Leider gibt es solche speziellen Versionen meist nur für Windows. Eine Liste dieser Emulatoren hat Sun auf der Seite <http://developers.sun.com/techtomics/mobility/midp/articles/emulators/> zusammengestellt.

Immerhin können aber zwei andere Emulatoren auf Mac OS X genutzt werden. Und auch wenn sie nicht MIDP-zertifiziert sind, stellen sie dennoch eine wertvolle Hilfe beim Testen unter verschiedenen Umgebungen dar. Beide Entwicklungen sind Open Source:

► **ME4SE**

Ziel dieses Emulators ist es, die J2ME-Pakete in der ganz normalen J2SE-Umgebung zur Verfügung zu stellen.

<http://www.me4se.org/>

► **MicroEmulator**

Dieser MIDP-Emulator läuft als Applet im Browser und setzt nur ein JDK 1.1 voraus.

<http://www.barteo.net/microemulator/>

12.2 Einsatz im mobilen Endgerät

Der Test eines MIDlets im MIDP-Emulator ist schön und gut (und vor allem auch schnell und praktisch). Bevor Sie aber Ihre J2ME-Software veröffentlichen, sollten Sie diese auf jeden Fall in den realen Geräten testen, auf denen die MIDlets später laufen werden – der Emulator der Referenz-Implementation kann eben nicht alle Besonderheiten und Bugs eines speziellen Geräts berücksichtigen. Sie müssen also die MIDlets von Ihrem Rechner auf das MIDP-Gerät übertragen. Das Gerät ist dabei entweder lokal direkt mit Ihrem Rechner verbunden, oder es lädt das MIDlet von einem Web-Server herunter.

12.2.1 Lokale Installation

Zwei typische Anwendungsfälle werden in diesem Abschnitt vorgestellt: das Einspielen per Bluetooth auf ein Mobiltelefon und die Übertragung via USB-Docking-Station auf einen PalmOS-kompatiblen Organizer.

Bluetooth-Handy

Die Beschreibung setzt ein Bluetooth-Handy voraus, da Mac OS X 10.3 die zur Bluetooth-Kommunikation nötige Software bereits komplett mitbringt! Neuere PowerBooks besitzen zudem eine eingebaute Bluetooth-Schnittstelle, bei anderen Macs ist ein Bluetooth-Adapter intern von Apple oder extern als USB-Modul nachrüstbar.¹ Prinzipiell ist die Kommunikation mit dem Handy auch über ein USB-Datenkabel möglich, allerdings ist dafür dann eine spezielle Herstellerunterstützung nötig – und die ist für Mac OS X leider oft nicht vorhanden. Wenn überhaupt, kann dann nur mit teurer Zusatzsoftware von Drittherstellern gearbeitet werden.



Abbildung 12.8 Systemeinstellungen »Bluetooth«

Jedes Bluetooth-Gerät muss zunächst in den Systemeinstellungen im Bereich »Bluetooth« angemeldet werden (siehe Abbildung 12.8). Ein neues Gerät kön-

1 Als sehr gut hat sich der D-Link DBT-120 USB-Adapter erwiesen, da Apple offenbar selbst D-Link-Module verbaut und die Software-Unterstützung – vor allem bei Firmware-Updates – damit am problemlosesten ist.

nen Sie auf der Seite »Geräte« mit »Verbindung zu neuem Gerät« einrichten, alternativ rufen Sie mit »Neues Gerät konfigurieren« den Bluetooth Assistent auf, den Sie im Verzeichnis /Programme/Dienstprogramme/ finden.

Bevor das Gerät in der Liste auftaucht, muss eine Identifikation zwischen dem Rechner und dem Gerät stattfinden. Dazu fordert Sie die Mac-Bluetooth-Software auf, ein Passwort (eine Zahlenkombination) einzugeben, das Sie anschließend auf der Handy-Tastatur wiederholen müssen. Anschließend können Sie den Namen des Handys im Gerät selber beliebig verändern, beispielsweise können Sie – wie hier im Beispiel – die Typbezeichnung eintragen.

Der Einfachheit halber sollten Sie dann noch auf der Seite »Einstellungen« die Option »Bluetooth Status in der Menüleiste anzeigen« aktivieren. Damit erhalten Sie durch ein kleines Symbol oben rechts am Bildschirm schnellen Zugriff auf die wichtigsten Bluetooth-Funktionen (siehe Abbildung 12.9). Mit »Datei senden...« können Sie eine beliebige Datei zum Handy übertragen, »Gerät durchsuchen...« erlaubt das Herunterladen von Handy-Daten auf den Rechner (was ungefähr wie ein FTP-Download funktioniert).

Wenn Sie gleich ein MIDlet auf Ihr Handy übertragen haben, taucht es beim Durchsuchen des Geräts dann oft trotzdem nicht auf. Java-Daten werden bei den meisten Mobiltelefonen in einem versteckten Verzeichnis abgelegt, vermutlich um das Kopieren von Software zu erschweren. Auch das Löschen von MIDP-Applikationen ist nur auf dem Handy selbst möglich.

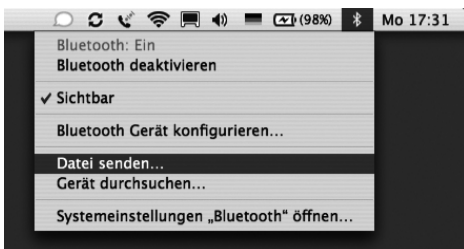


Abbildung 12.9 Bluetooth-Menü

Wenn Sie »Datei senden...« anklicken, erscheint zunächst eine Dateiauswahl, mit der Sie die zu übertragende Datei – hier also das JAR-Archiv des MIDlets – bestimmen. Danach öffnet sich das Programm Bluetooth Datenaustausch, das Sie im Dienstprogramme-Ordner auch direkt aufrufen können. Wählen Sie Ihr Handy aus der Liste der konfigurierten Geräte aus, oder lassen Sie sich mit »Suchen« alle Bluetooth-Geräte in der Umgebung anzeigen (siehe Abbildung 12.10).



Abbildung 12.10 Gerät im »Bluetooth Datenaustausch« auswählen

Bevor der Rechner die Datei übertragen kann, muss er sich die Erlaubnis vom Gerät einholen (siehe Abbildung 12.11). Ihr Handy wird Sie fragen, ob Sie die ausgewählte Software übertragen und installieren wollen – bestätigen Sie dies. Alle diese Abfragen sind aus Sicherheitsgründen notwendig und sinnvoll, damit nicht irgendwer auf Ihrem Gerät unbemerkt Software installieren kann.

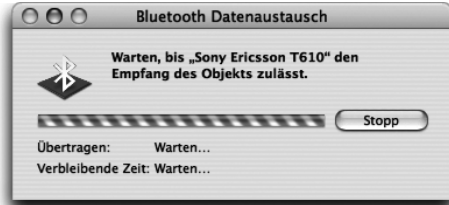


Abbildung 12.11 Das Übertragen muss im Handy bestätigt werden.

Danach wird die Datei dann übertragen (siehe Abbildung 12.12). Dank Bluetooth geschieht das bei den üblicherweise recht kleinen MIDlets meistens schneller, als die Übertragungszeit sinnvoll berechnet werden kann.

Nach dem Empfang der Software wird das MIDlet automatisch im Handy installiert – üblicherweise sagt Ihnen das Handy dann auch noch, wo Ihr Programm gespeichert wurde. Beim hier verwendeten Sony Ericsson T610 landen MIDlets beispielsweise in der Kategorie »Unterhaltung«, wo Sie sie wie die vorinstallierten Spiele aufrufen können.



Abbildung 12.12 Eine Datei wird zum Handy übertragen.

Wenn Sie Bluetooth aus der MIDP-Umgebung heraus ansprechen möchten, können Sie von der Seite <http://www.ayetana-gmbh.de/ayetana-gmbh/jsr82.xml> ein JAR-Archiv für Mac OS X herunterladen, das den entsprechenden Standard JSR-82 implementiert.

Organizer mit USB-Docking-Station

Neben Java-fähigen Mobiltelefonen eignen sich Palm-kompatible Organizer (PDAs) zum Einsatz von MIDlets, allerdings ist auf diesen Geräten meistens noch keine MIDP-Umgebung vorhanden. Sun bietet daher auf der Seite <http://java.sun.com/products/midp4palm/download.html> eine Laufzeitumgebung inklusive MIDP-Klassenbibliothek zum kostenlosen Download an. Geeignet ist das Archiv `midp4palm-1.0.zip` für PalmOS 3.5, das unter anderem von Palm, Handspring und Sony eingesetzt wird. In diesem Archiv ist die MIDP-Umgebung im PalmOS-Installationspaket `MIDP.prc` enthalten, außerdem finden Sie darin noch diverse Beispiele.

Wenn Sie sich die Beispiel-MIDlets ansehen, werden Sie feststellen, dass keines davon als JAR-Archiv vorliegt, sondern alle als PRC-Dateien. Da das PalmOS mit JAR- und JAD-Dateien nichts anfangen kann, müssen Sie auch Ihre eigenen MIDlets vor der Übertragung zum Organizer in das PRC-Format umwandeln. Zum Glück liefert Sun dafür das Archiv `Converter.jar` mit, das Sie einfach durch einen Doppelklick starten können (siehe Abbildung 12.13). Zum Umwandeln eines MIDlets rufen Sie im »PRC Converter« den Menüpunkt **File • Convert** auf und wählen die JAD-Datei Ihres MIDlets aus (ein JAR-Archiv mit Manifest reicht nicht aus). Aus der JAD-Datei und dem darin referenzierten JAR-Archiv erzeugt der »PRC Converter« dann ein PalmOS-Installationspaket.

Das PRC-Installationspaket können Sie nun mit der vom PDA-Hersteller mitgelieferten Synchronisationssoftware auf Ihren Organizer übertragen. Palm liefert zur Verwaltung des Handhelds beispielsweise die Software »Palm Desktop 4.0« aus, zu dem auch der »Hotsync Manager« gehört. Im Hotsync Manager können

Sie PRC-Pakete zur Übertragung bei der nächsten Rechner-PDA-Synchronisation vormerken – ziehen Sie die Dateien einfach auf das Hotsync-Fenster (siehe Abbildung 12.14).

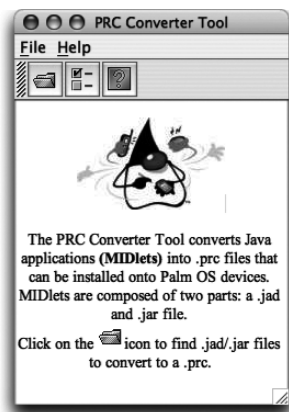


Abbildung 12.13 PRC Converter

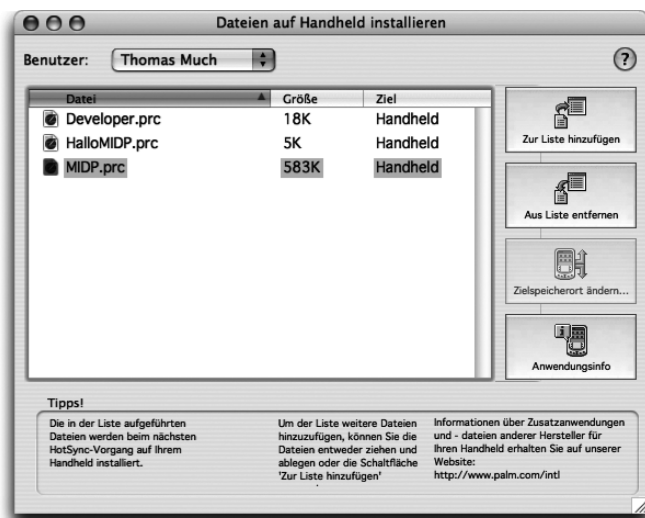


Abbildung 12.14 Dateien mit HotSync auf dem Handheld installieren

Nach der Übertragung finden Sie die Java-Umgebung (das Paket MIDP.prc) als Programm »Java™HQ« in der Programmgruppe »Alle«. Wenn Sie das Programm starten, können Sie darin generelle Einstellungen zum Speicher, zu Netzwerkverbindungen, zur Farbtiefe usw. vornehmen. Ausgeführt wird eine Java-Anwendung dann, indem Sie die Applikation anhand ihres Namens – beispielsweise »HalloMIDP« – in derselben Programmgruppe direkt aufrufen.

12.2.2 Installation vom Web-Server

Während die lokale Installation zum Testen hervorragend geeignet ist, ist die Installation vom Web-Server der geeignete Weg, um Ihr MIDlet an möglichst viele Anwender zu verteilen. Im Wesentlichen muss der Benutzer dafür in einem HTML-Dokument einen Link auf die JAD-Datei des MIDlets anwählen (beispielsweise über WAP):

```
<!-- CD/examples/ch13/Web-Sites/hallomidp.html -->
<html>
<head>
  <title>J2ME MIDP Demo</title>
</head>
<body>
  <p><a href="HalloMIDP.jad">HalloMIDP herunterladen
  und installieren</a></p>
</body>
</html>
```

Listing 12.3 HTML-Datei zum Einbinden eines MIDlets

Nach dem Anklicken des Links wird die JAD-Datei heruntergeladen. Da in der Datei eine URL auf das JAR-Archiv gespeichert ist, weiß der Client (das Handy oder der Organizer) dann, wo er den eigentlichen Programmcode nachladen kann. Für den Einsatz im lokalen Web-Server sieht die JAD-Datei wie folgt aus (die einzelnen Einträge wurden schon weiter vorne beschrieben):

```
MIDlet-Name: HalloMIDP
MIDlet-Version: 1.0.0
MIDlet-Vendor: Thomas Much
MIDlet-Info-URL: http://www.muchsoft.com/java/
MIDlet-Jar-URL: http://127.0.0.1/~much/HalloMIDP.jar
MIDlet-Jar-Size: 3741
```

Listing 12.4 JAD-Datei

Für den lokalen Web-Server wird die »localhost«-IP-Adresse 127.0.0.1 verwendet, »much« müssen Sie durch Ihren Benutzernamen ersetzen. Außerdem muss die exakte Größe des JAR-Archivs in Bytes angegeben werden.

Damit der Web-Server die JAD-Datei mit dem korrekten MIME-Typ ausliefert, muss die Dateinamenserweiterung `jad` in der Server-Konfigurationsdatei wie folgt bekannt sein:

```
text/vnd.sun.j2me.app-descriptor    jad
```

Dieser MIME-Typ ist beim mit MacOS X ausgelieferten Apache-Web-Server zwar eingetragen, die Dateinamenserweiterung fehlt aber noch und muss bei Bedarf ergänzt werden. Wie das geht, haben Sie in Kapitel 4, *Ausführbare Programme*, bei »Java Web Start« bereits kennen gelernt.

12.3 Benutzungsoberflächen und Grafik

Dieses Kapitel kann und soll zwar keine komplette J2ME-Anleitung sein, aber sobald Sie einmal mit der Entwicklung von MIDP-Applikationen begonnen haben, möchten Sie sicherlich noch weitere Möglichkeiten nutzen. Daher finden Sie im Folgenden Ausschnitte aus einem etwas größeren Beispiel, das mehr Oberflächenelemente einsetzt, vor allem aber die Verwendung von grafischen Elementen sowie die Ereignisverarbeitung demonstriert.

```
//CD/examples/ch13/MIDPDemo/MIDPDemo.java
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class MIDPDemo extends MIDlet implements CommandListener {
```

Das Interface `javax.microedition.lcdui.CommandListener` definiert eine Methode `commandAction()`, mit der Sie auf `Command`-Objekte reagieren können (der Programmcode für die Implementierung folgt weiter unten). Mit `Command`-Objekten werden bestimmte Knöpfe des Handys, die so genannten »Soft Buttons«, je nach Kontext geeigneten Aktionen zugeordnet. In der `MIDlet`-Klasse werden zunächst einige Objektvariablen definiert:

```
private Form form = new Form("MIDPDemo");
private Command fertig = new Command("Ende",Command.EXIT,1);
private Command grafik = new Command("Grafik",Command.SCREEN,1);
private Zeichnung zeichnung = new Zeichnung( this );
private ChoiceGroup farbe;
```

Neben dem schon bekannten `Form`-Objekt für den Dialog werden hier zwei `Command`-Objekte erzeugt. Den Kommandos wird eine Beschriftung, ein Typ (ein `EXIT`-Knopf zum Beenden und ein `SCREEN`-Knopf für beliebige Aktionen) und eine Priorität mitgegeben. Die Priorität gibt eine Rangfolge vor, welche Kommandos angezeigt werden, wenn der Platz auf dem Display knapp wird – die nicht so wichtigen Kommandos landen dann z.B. in einem automatisch generierten Untermenü. Die letzten beiden Variablen werden gleich noch vorgestellt.

```

public MIDPDemo() {
    farbe = new ChoiceGroup( "Farbe",
                             ChoiceGroup.EXCLUSIVE,
                             new String[] { "Rot", "Gruen", "Blau",
                                             null });

    form.append( farbe );
    form.addCommand( fertig );
    form.addCommand( grafik );
    form.setCommandListener( this );
}

```

Im Konstruktor wird zunächst ein `ChoiceGroup`-Objekt erzeugt, das dem Anwender eine Liste von Auswahlmöglichkeiten anbietet. Mit der Konstanten `EXCLUSIVE` wird festgelegt, dass immer nur ein Wert der Liste markiert sein kann (»Radio Buttons«) – alternativ könnten Sie hier `MULTIPLE` für normale Ankreuzkästchen (»Checkbox«) übergeben. Die Werte der Liste werden direkt als `String[]`-Array übergeben. Titel der Liste ist »Farbe«.

Beachten Sie, dass hier keine Umlaute verwendet werden! Unicode-Unterstützung ist für MIDP-Umgebungen nicht verpflichtend, und viele Endgeräte kommen damit auch nicht zurecht. Als Kodierung wird meistens das im Web übliche ISO 8859-1 verwendet. Die MIDP-API kennt generell keine Klassen zur Lokalisierung bzw. Internationalisierung (I18N), Sie müssen bei Bedarf also alles »zu Fuß« erledigen.

Anschließend wird der Formularinhalt wieder mit `append()` an den Dialog angehängt. Im Gegensatz dazu werden die Kommando-Objekte mit `addCommand()` hinzugefügt – sie werden aber ebenfalls dem `Form`-Objekt zugeordnet. Schließlich wird mit `setCommandListener()` das Objekt gesetzt, das beim Anwählen eines Kommandos aufgerufen wird. Da es zu jedem Dialog immer nur einen Listener geben kann, wird hier einfach das `MIDlet`-Objekt selbst übergeben, das ja das passende Interface `CommandListener` implementiert.

```

public void startApp() {
    Calendar cal = Calendar.getInstance();
    cal.setTime( new Date() );
    Ticker t = new Ticker( "Gestartet: " + cal.toString() );
    form.setTicker( t );
    Display.getDisplay(this).setCurrent( form );
}

```

In der `startApp()`-Methode wird nicht nur das Formular-Objekt angezeigt, sondern vorher noch ein so genannter `Ticker` gesetzt. Ein `Ticker` ist ein Objekt, das ständig eine Zeichenkette auf dem `Display` durchscrollt, beispielsweise oberhalb des Formulars (wo genau der Text angezeigt wird, ist Sache der MIDP-Umgebung). Mit `setTicker()` wird das `Ticker`-Objekt dem Formular zugeordnet. Ab diesem Zeitpunkt kann das Gerät den Text durchlaufen lassen, es gibt keine Möglichkeit zum Anhalten – Sie können den `Ticker` mit `setTicker(null)` nur wieder löschen.

Als `Ticker`-Zeichenkette werden Zeit und Datum zum Zeitpunkt des `startApp()`-Aufrufs ermittelt. Bei mehrfachem Aufruf kann sich der Text also ändern! Allerdings läuft die Uhr im `Ticker` nicht ständig mit, dafür wäre `Thread`-Programmierung nötig. Mit `new Date()` wird der jetzige Zeitpunkt abgefragt. Allerdings besitzt `java.util.Date` bei MIDP keine eigene `toString()`-Methode, so dass man den Umweg über ein `Calendar`-Objekt gehen muss! `java.util.Calendar` bietet die Funktionalität der J2SE-Klasse `GregorianCalendar` in eingeschränkter Form, aber immerhin ist darin ein geeignetes `toString()` vorhanden.

```
public void commandAction(Command c, Displayable d) {
    if (c == grafik) {
        int sel = farbe.getSelectedIndex();
        if (sel >= 0) {
            zeichnung.setRGB( 0x000000ff << (8 * (2-sel)) );
            Display.getDisplay(this).setCurrent(zeichnung);
        }
    }
    else if (c == fertig) {
        destroyApp(false);
        notifyDestroyed();
    }
}
```

Die Ereignisverarbeitungs-Methode `commandAction()` wird aufgerufen, wenn der Benutzer ein Kommando-Objekt angewählt hat. Bei diesem Beispiel wird die übergebene `Displayable`-Referenz ignoriert, aber Sie könnten damit feststellen, zu welchem Formular das Ereignis gehört. Da hier nur ein Formular eingesetzt wird, ist eine Abfrage überflüssig.

Am Ende der Methode wird mit der `fertig`-Referenz und dem Eingabeparameter `c` geprüft, ob der »Ende«-Knopf angewählt wurde. In diesem Fall wird häufig zunächst `destroyApp()` aufgerufen (falls Sie dort Aufräum-Code programmiert haben). Anschließend informiert das MIDlet die MIDP-Umge-

bung mit `notifyDestroyed()`, dass es keine weiteren Ressourcen-Zugriffe durchführen wird und beendet werden möchte.

Davor wird auf den `grafik`-Knopf geprüft, bei dessen Anwahl ein gefüllter Kreisbogen (»Tortenstück«) in der ausgewählten Farbe gezeichnet werden soll. Dazu wird mit `getSelectedIndex()` der ausgewählte `ChoiceGroup`-Eintrag ermittelt und daraus (ein wenig trickreich) ein RGB-Farbwert berechnet. Die Darstellung kann aber nicht im derzeitigen Formular erfolgen, sondern muss in einer speziellen Grafikumgebung durchgeführt werden. Daher wird mit `setCurrent()` das Zeichnung-Objekt, das ganz am Anfang in der `MIDlet`-Klasse erzeugt wurde, als aktueller Dialog gesetzt. Und da immer nur ein Dialog zur selben Zeit aktiv sein kann, wird nun der Grafik-Dialog angezeigt, der in einer separaten Klasse realisiert ist:

```
//CD/examples/ch13/MIDPDemo/Zeichnung.java
import javax.microedition.lcdui.*;
class Zeichnung extends Canvas {
```

Zum Darstellen von Grafik müssen Sie eine Klasse von `javax.microedition.lcdui.Canvas` (`Canvas` bedeutet »Leinwand«) ableiten. Darin können Sie dann mit diversen Grafikfunktionen Linien, Rechtecke usw. zeichnen, Bilder anzeigen und auf so genannte Low-Level-Ereignisse (z. B. Tastendrucke) reagieren. Die `Soft-Buttons` können auch beim `Canvas` mit Kommandos belegt werden, aber Oberflächenelemente wie `ChoiceGroup` stehen nicht zur Verfügung.

Da `Canvas` die abstrakte Methode `paint()` deklariert, müssen Sie diese Methode implementieren. Wie beim `AWT` erhalten Sie beim Aufruf einen Grafik-Kontext, allerdings besitzt die `MIDP-Graphics`-Klasse weniger Methoden als das `J2SE`-Pendant:

```
    public void paint(Graphics g) {
        g.setColor(rgb);
        g.fillArc(0,0,this.getWidth(),this.getHeight(),0,270);
    }
```

Zunächst wird der vorher berechnete RGB-Farbwert gesetzt – aber Achtung: Farbe muss nicht von allen Geräten unterstützt werden! Es kann also sein, dass die Anzeige nur mit Graustufen oder sogar nur in Schwarz-Weiß erfolgt. Dann wird mit `fillArc()` ein gefüllter Kreisbogen von 0 bis 270 Grad gezeichnet. Die Ausgabe erfolgt im gesamten nutzbaren `Canvas`-Bereich: von der Position (0,0) links oben bis zur maximalen Breite und Höhe, für die ein `Canvas`-Objekt die beiden Methoden `getWidth()` und `getHeight()` besitzt. Der nutzbare

Bereich entspricht übrigens meistens nicht dem gesamten Display-Bereich, da auch noch Platz für andere Elemente wie die Soft-Buttons freigehalten wird.

Damit ist die Zeichnung fertig, nun folgt noch die Ereignisbehandlung. Diese wird hier nicht mit Kommando-Objekten durchgeführt, stattdessen soll ganz einfach jeder Tastendruck die Grafik wieder verschwinden lassen. Dazu müssen Sie die `Canvas`-Methode `keyPressed()` überschreiben, in der Sie auf solche Low-Level-Ereignisse reagieren können:

```
public void keyPressed(int keyCode) {
    midlet.startApp();
}
```

Als Parameter wird der Tastencode übergeben. Mit den `Canvas`-Konstanten `KEY_NUM0`, `KEY_NUM1` usw. können Sie die Tasten der Handy-Tastatur abfragen. Oder Sie wandeln den Code mit `getGameAction()` in eine so genannte »Game Action« um, dann können Sie mit den Konstanten `UP`, `DOWN`, `FIRE` usw. portabel auf Cursorbewegungen z.B. mit einem Handy-Joystick testen.

Hier interessiert der Tastencode nicht, denn die Grafik soll ja bei irgendeiner gedrückten Taste verschwinden. Dazu wird nicht etwa der `Canvas` ausgeblendet, sondern einfach das ursprüngliche Formular wieder sichtbar gemacht, indem man die `startApp()`-Methode des MIDlets direkt aufruft. Darin wird dann ja wieder das `Form`-Objekt mit `setCurrent()` als aktiver Dialog gesetzt.

Fertig! Abbildung 12.15 zeigt, wie das gestartete MIDlet im MIDP-Emulator aussieht. Sie sehen, dass bei der Grafikanzeige der alte Dialoghintergrund nicht automatisch gelöscht wird – dazu könnten Sie `fillRect()` verwenden. Allerdings gibt es reale Geräte, die den Hintergrund bei jedem neuen Dialog weiß füllen. Damit Sie von solchen kleinen Abweichungen im Verhalten nicht überrascht werden, bleibt Ihnen nur eines: das Testen des MIDlets in allen gewünschten Endgeräten.



Abbildung 12.15 Ein MIDlet mit Grafikausgabe

Wenn Sie J2SE- oder J2EE-Umgebungen gewohnt sind, müssen Sie bei J2ME einige Einschränkungen hinnehmen: Sie müssen Speicher und Rechenzeit spa-

ren, selbst wenn dadurch eventuell die Eleganz des Codes unter objektorientierten Gesichtspunkten leidet. Beherzigen Sie folgende Tipps:

- ▶ Vermeiden Sie häufige `String`-Operationen. Verwenden Sie dann besser die Klasse `StringBuffer`.
- ▶ Innere Klassen sollten Sie nach Möglichkeit nicht einsetzen (dies gilt nicht für statische innere Klassen).
- ▶ Setzen Sie Objekt-Referenzen auf `null`, sobald Sie diese nicht mehr benötigen.
- ▶ Vermeiden Sie unnötige Doppelinitialisierungen von Objekt- und Klassenvariablen, die schon von der Laufzeitumgebung auf `0` bzw. `null` gesetzt werden.
- ▶ Setzen Sie Synchronisation von Threads nur sehr sparsam ein. Synchronisation kostet sehr viel Zeit und wird nur bei nebenläufigen Applikationen wirklich gebraucht.
- ▶ Mit einem so genannten »Obfuscator« können Sie die Dateigröße der Klassen verringern und dadurch Speicherplatz sparen. Ein Beispiel dafür finden Sie auf der Seite <http://developers.sun.com/techtopics/mobility/midp/ttips/proguard/>; ProGuard haben Sie in Kapitel 8, *Werkzeuge*, schon kennen gelernt.

12.4 Literatur & Links

- ▶ K.Topley, »J2ME in a Nutshell«, O'Reilly 2002
Obwohl dieses Buch nur auf Englisch verfügbar ist, ist es dennoch gut zu lesen. Und vor allem beschreibt es J2ME mit den verschiedenen Konfigurationen und Profilen ziemlich komplett.
- ▶ M.Kroll/S. Haustein, »J2ME – Developer's Guide«, Markt+Technik 2003
Trotz des Titels ist dies ein deutsches Buch, das einen guten Einstieg in die J2ME-Programmierung bietet.
- ▶ <http://www.j2meforum.com/>
Diese Seite bietet deutschsprachige Diskussionsgruppen zu allen J2ME-Themen und ist hochgradig empfehlenswert!
- ▶ <http://java.sun.com/products/cldc/>
Informationen von Sun zur CLDC-Konfiguration
- ▶ <http://java.sun.com/products/midp/>
Auch zum MIDP-Profil hält Sun Informationen bereit.
- ▶ <http://developer.apple.com/qa/qa2001/qa1232.html>
Auf dieser Seite beantwortet Apple die wichtigsten Fragen zur X11-Implementation von Mac OS X.

A Kurzeinführung in die Programmiersprache Java

*»Das Leben ist kurz, lang die Kunst.«
(Hippokrates)*

Wenn Sie bisher noch nie oder wenig mit Java programmiert haben, kann Ihnen dieses Kapitel die wichtigsten Grundlagen vermitteln oder Ihr Wissen wieder auffrischen. Voraussetzung dafür ist allerdings, dass Sie bereits irgendeine Programmiersprache beherrschen, damit Begriffe wie »Variable«, »Schleife« und »Funktion« (bzw. »Prozedur«, »Unterroutine« oder wie auch immer sie von der jeweiligen Sprache genannt werden) keine Fremdwörter für Sie sind. Ebenso ist dieses Kapitel keine Einführung in die objektorientierte Programmierung (OOP). Viele OO-Konzepte werden zwar gezeigt, allerdings nur um darzustellen, wie sie in Java funktionieren.

Wenn Sie die Grundlagen der objektorientierten Programmierung lernen wollen oder eine allgemeine Programmierintroduction benötigen, finden Sie am Schluss Hinweise auf geeignete Literatur und Online-Quellen.

A.1 Programme, Klassen, Pakete

Programme bestehen in Java aus einem oder mehreren Modulen, den **Klassen**:

```
public class HalloWelt
{
    /* Inhalt der Klasse: Daten und Methoden */
} // Ende class HalloWelt
```

Listing A.1 Quelltext-Datei HalloWelt.java

Die Zeichenketten `public` und `class` gehören zu den reservierten **Schlüsselwörtern**, von denen Java etwa 50 besitzt. Diese Schlüsselwörter können nicht als Bezeichner (Variablen-, Methoden- oder Klassenname) verwendet werden.

Die Textformatierung ist bei Java-Quelltexten nicht entscheidend. Sie könnten das gesamte Programm in einer Zeile schreiben – ob das dann noch übersichtlich und wartbar ist, ist natürlich eine andere Sache ... Versuchen Sie daher, die zusammengehörigen geschweiften Klammern auf einer Ebene zu belassen und den Inhalt dieses Blocks einen Tabulator oder zwei bzw. vier Leerzeichen einzurücken (so wie oben abgebildet). Während oben die öffnende Klammer alleine auf einer Zeile steht (was üblich ist und die Block-Übersichtlichkeit

erhöht), wird bei den anderen Quelltexten in diesem Buch aus Platzgründen die andere übliche Schreibweise verwendet: Die öffnende Klammer befindet sich noch in der Zeile darüber am Ende der vorhergehenden Anweisung. Die freie Textformatierung geht bei Java so weit, dass sich überall dort, wo ein Leer- bzw. Trennzeichen steht, beliebig viele Leerzeichen, Tabulatoren, Kommentare oder Zeilenumbrüche befinden können.

Nach `class` folgt der Klassenname, der ein gültiger **Bezeichner** sein muss – wie bei vielen anderen Sprachen sind darin keine Leerzeichen erlaubt, der Name muss mit einem Buchstaben beginnen, danach können dann auch Ziffern verwendet werden. Diese Bezeichner-Regeln gelten auch für Variablen- und Methodennamen. Die **Java-Programmierrichtlinien** besagen, dass ein Klassenname mit einem Großbuchstaben beginnen sollte. Besteht ein Name aus mehreren Wörtern, werden alle Wörter zusammen und deren Anfangsbuchstaben jeweils groß geschrieben.

Der Inhalt der Klasse folgt in geschweiften Klammern, außerhalb sind keine Anweisungen erlaubt – es gibt in Java keinen »globalen« Code. Was Sie dort verwenden können, sind **Kommentare**:

Mit `//` wird ein Kommentar bis zum Zeilenende definiert,

mit `/*` ein Kommentar innerhalb einer Zeile `*/`.

Diese zweite Form kann auch für mehrzeilige Kommentare eingesetzt werden.

Üblicherweise wird jede Klasse in einer eigenen Datei gespeichert. Wenn diese Klasse die `public`-Kennzeichnung besitzt (was bei Ihren Klassen bis auf weiteres der Fall sein sollte), muss die Datei exakt so wie die Klasse heißen – achten Sie auf die Groß-/Kleinschreibung! Die Datei erhält allerdings zusätzlich die Dateinamenserweiterung `.java` als Kennzeichnung für Java-Quelltexte. Halten Sie sich aber besser daran, die **Quelltext-Datei** immer so wie die Klasse zu benennen, egal ob die Klasse `public` ist oder nicht. Weitere Klassen in einer Datei sind also nur möglich, wenn diese nicht `public` sind – das wird normalerweise nur für Hilfsklassen ausgenutzt, die nur von der jeweiligen `public`-Klasse verwendet werden.

Die Kennzeichnung `public` legt die **Sichtbarkeit** der Klasse fest. Ein Java-Programm besteht fast immer aus mehreren Klassen, häufig auch aus solchen in fremden Bibliotheken. Dadurch ist aber die Eindeutigkeit eines Klassennamens – beispielsweise `Test` – nicht mehr unbedingt gegeben. Daher werden Klassen so genannten **Paketen** (»**Packages**«) zugeordnet und damit quasi in Schubladen einsortiert:

```
package com.muchsoft.java;
public class HalloWelt { /*... */ }
```

Die `package`-Anweisung kann einmal am Anfang jeder Quelltext-Datei auftauchen und gilt für alle Klassen dieser Datei (und nebenbei ist `package` damit eine der sehr wenigen Anweisungen, die doch außerhalb von Klassen stehen können bzw. müssen). Der komplette Klassenname von `HalloWelt` lautet nun `com.muchsoft.java.HalloWelt`, man spricht hierbei vom »vollqualifizierten« Klassennamen. Dieser Klassenname ist nun weltweit eindeutig – natürlich nur, sofern der Paketname eindeutig ist. Dies erreicht man dadurch, dass jeder Programmierer bzw. jede Firma als Präfix des Paketnamens einfach die eigene Domain in umgekehrter Reihenfolge in Kleinschreibung voranstellt (das »www« wird dabei weggelassen), hier also `com.muchsoft`. Für die Eindeutigkeit der folgenden Unterpakete ist nun jeder Domaininhaber selbst verantwortlich. Fehlt die Paket-Angabe, wird die Klasse dem so genannten »anonymen« Paket ohne Namen zugeordnet.

Die Sichtbarkeit von Klassen definiert auch den **Zugriffsschutz**. Es geht hierbei nicht um Sicherheit vor Hacker-Angriffen, sondern darum, dass Sie der Compiler warnen kann, wenn Sie aus Versehen eine Klasse verwenden wollen, die Sie besser nicht einsetzen sollten – beispielsweise eine interne Klasse eines anderen Projekt-Teams, die keine öffentlichen Schnittstellen definiert. `public`-Klassen können von sämtlichen anderen Klassen genutzt werden, egal welchem Paket diese anderen Klassen zugeordnet sind. Fehlt die Kennzeichnung ganz, kann die Klasse nur von Klassen im selben Paket verwendet werden. Diese Art der Gruppierung bzw. Sortierung von Klassen ist also hervorragend dazu geeignet, die Übersicht über viele (hunderte oder tausende) Klassen zu behalten!

Um eine Klasse eines anderen Pakets zu verwenden, ist es immer möglich, den vollqualifizierten Namen anzugeben (sofern Sie die Klassen sehen und benutzen dürfen), z.B. `java.awt.Button`. Diese ausführliche Schreibweise wird schnell lästig, daher kann sie mit der `import`-Anweisung abgekürzt werden:

```
package com.muchsoft.java;
import java.awt.*;
import java.awt.event.*;
public class HalloWelt {
    Button knopfOK;
    // ...
}
```

Hier werden durch das Sternchen zunächst alle Klassen aus dem Paket `java.awt` importiert. Durch diese Anweisung wird kein Quelltext oder Binär-

code eingebunden, es handelt sich einfach nur um einen Hinweis für den Compiler: Wenn `Button` im Quelltext auftaucht, ist `java.awt.Button` gemeint. Sie können beliebig viele `import`-Anweisungen angeben und statt des Sternchens auch gezielt einzelne Klassen importieren. Die Anweisungen gelten aber nur für die jeweilige Datei und müssen in anderen Quelltext-Dateien je nach Bedarf erneut aufgeführt werden.

Unterpakete – hier `java.awt.event` als Unterpaket von `java.awt` – müssen separat importiert werden. Das Sternchen importiert also wirklich nur die Klassen des angegebenen Pakets! Dies ist absolut sinnvoll, denn wenn das Sternchen auch Unterpakete mit einbinden würde, wäre die Wahrscheinlichkeit viel zu hoch, dass darin doppelte Klassennamen auftauchen – die gewünschte Eindeutigkeit ginge also verloren. Klassen aus dem Paket `java.lang` müssen nicht importiert werden und stehen immer zur Verfügung. Bei den Klassen handelt es sich um die absoluten Grundlagen-Klassen wie `String` oder `System`.

Die **Java-API-Dokumentation** ist nach Paketen und Klassen sortiert. Wenn Sie Apples Java-Entwicklerpaket installiert haben, finden Sie die Dokumentation im Verzeichnis `/Developer/Documentation/Java/Reference/` bzw. (bei neueren Installationen) in `/Developer/ADC Reference Library/documentation/Java/Reference/`. Online ist die Dokumentation bei Sun auf der Seite <http://java.sun.com/reference/api/> abrufbar.

Pakete werden zwar prinzipiell auf eine entsprechende Verzeichnishierarchie abgebildet, normalerweise ist diese aber innerhalb eines JAR-Archivs versteckt. Wie Sie Zugriff auf Klassen in solchen Archiven haben, ist in den Kapiteln 1, *Grundlagen*, und 4, *Ausführbare Programme*, ausführlich beschrieben – Sie müssen die Bibliothek, also das JAR- oder ZIP-Archiv, in den Klassenpfad einbinden. Die `import`-Anweisungen finden dann auch Klassen in diesen Archiven.

Die **Übersetzung** der Quelltexte zu ausführbarem Bytecode erfolgt mit dem Java-Compiler `javac`, dem Sie die zu übersetzenden Quelltexte mitgeben:

```
javac HalloWelt.java
javac *.java
```

Für jede Klasse wird dabei eine Klassendatei mit der Dateinamenserweiterung `.class` angelegt, die den Bytecode der Klasse enthält. Damit das Java-Programm gestartet werden kann, muss eine Klasse – die so genannte **Hauptklasse** – eine `main()`-Methode besitzen, wie Sie sie schon in Kapitel 1, *Grundlagen*, gesehen haben. Diese ist nicht bei jeder Klasse nötig, denn Sie können natürlich auch Bibliotheken schreiben, die nicht direkt gestartet, sondern in andere Java-Programme eingebunden werden.

Der **Aufruf** des Programms erfolgt durch Starten der Java-Laufzeitumgebung mit dem Kommando `java`, dem Sie den Namen der Hauptklasse (ohne das `.class`-Suffix!) mitgeben:

```
java HalloWelt
```

A.2 Methoden und Anweisungen

Unterprogramme, Subroutinen, Prozeduren, Funktionen – all das hat in Java einen einzigen Namen: **Methoden**. Jede Methode trägt einen Namen (z.B. `main`), der wieder ein gültiger Bezeichner sein muss. Die Java-Programmierrichtlinien geben vor, dass ein Methodename mit einem Kleinbuchstaben beginnt. Bei mehreren Wörtern werden diese wieder zusammenschrieben, wobei alle folgenden Wörter mit Großbuchstaben beginnen.

```
public class HalloWelt {
    public static void main(String[] args) {
        // Anweisungen der main()-Methode:
        System.out.println( "Hallo Welt!" );
        System.out.println( verdoppleWert(3) );
    }

    // Methode "verdoppleWert" (s.u.)
}
```

Nach dem Namen folgen in runden Klammern die **formalen Parameter**. Erwartet die Methode keine Parameter, müssen Sie trotzdem die (leeren) Klammern angeben. Jeder Parameter wird mit einem Datentyp (z.B. `String[]` für ein Zeichenketten-Array) und Namen angegeben, mehrere Parameter werden durch Komma getrennt.

Vor dem Namen der Methode steht der **Rückgabotyp**: Zahlen, Zeichenketten, Objekte usw. Wenn hier `void` (»leer«) angegeben ist, hat die Methode keine Rückgabe – sie ist dann also am ehesten mit einer Prozedur zu vergleichen.

`static` macht aus der Methode eine so genannte **Klassenmethode** – Sie müssen kein Objekt erzeugen, um die Methode nutzen zu können. Grob gesagt schaltet `static` die Objektorientierung aus, Sie können dann prozedural programmieren.

`public` sorgt wieder dafür, dass die Methode von Klassen aus beliebigen Paketen genutzt werden kann. Sie können auch hier diese Kennzeichnung weglassen, dann steht diese Methode nur für Klassen im selben Paket zur Verfügung. Es gibt noch zwei weitere Kennzeichnungen: `protected` lässt die

Nutzung der Methode im selben Paket und in Unterklassen (in beliebigen Paketen) zu. Eine `private`-Methode kann nur innerhalb derselben Klasse aufgerufen werden.

Mit der Sichtbarkeit, `static`, dem Namen und den Parametern können Sie Methoden beliebig formulieren. Einzige Ausnahme ist `main()` – diese Methode muss exakt so aussehen wie hier angegeben, ansonsten findet das Programm seinen Startpunkt nicht (dies gilt auch für `String[] args`, was Sie angeben müssen, selbst wenn Sie keine Kommandozeilenparameter nutzen). Wenn ein Programm nicht gestartet werden kann, ist häufig ein Programmierfehler bei dieser Methode Schuld – der Name `main` ist eventuell falsch geschrieben, oder die eckigen Klammern hinter `String` fehlen.

Ausgehend von `main()` rufen Sie dann alle weiteren Methoden in derselben Klasse oder in anderen Klassen auf. Denken Sie daran, dass solche **Methodenaufrufe** nur innerhalb einer Methode zulässig sind und nicht außerhalb von Klassen! Hier wird beispielsweise der Ausgabebefehl `System.out.println()` aufgerufen, der auszugebende Text wird als Parameter (Argument) in Klammern übergeben. Eine **Anweisung** ist immer erst mit dem Semikolon zu Ende, dadurch können Sie an jeder Leerstelle (außer in Zeichenketten) einen Zeilenumbruch durchführen. Aber auch lange Zeichenketten können problemlos getrennt werden, denn Sie können **Zeichenketten** einfach verknüpfen: `"Hallo" + " Welt!"`. Dies funktioniert auch, wenn auf einer Seite des Plus-Operators eine Zahl oder Variable steht: `"Die Antwort lautet " + ergebnis`.

`println()` führt immer einen Zeilenumbruch bei der Ausgabe durch, `print()` lässt diesen weg. Innerhalb von Zeichenketten können Sie so genannte »Escape-Sequenzen« verwenden, z.B. `\n` für den Zeilenumbruch oder `\t` für einen Tabulator. Damit könnten Sie obige Ausgabe auch wie folgt formulieren:

```
System.out.print( "Hallo Welt!\n" );
```

Wenn Sie Textausgaben – vor allem Zahlen und Datumsangaben – formatieren möchten, stehen Ihnen dazu im Paket `java.text` unter anderem die Klassen `NumberFormat`, `DecimalFormat` und `SimpleDateFormat` zur Verfügung.

```
public static int verdoppleWert(int wert) {  
    int ergebnis;  
    ergebnis = wert * 2;  
    return ergebnis;  
}
```

Die Methode `verdoppleWert()` wird aus der `main()`-Methode heraus aufgerufen. Dabei ist die Reihenfolge der Methoden egal – Java kennt alle Methoden einer Klasse, auch wenn sie erst weiter unten im Quelltext definiert sind.

In Java müssen Sie alle **Variablen** vor der ersten Verwendung mit ihrem Datentyp und Namen deklarieren. Hier wird die lokale (automatische) Variable `ergebnis` angelegt, anschließend folgt eine Wertzuweisung durch eine Formel (auch Ausdruck oder »Expression« genannt). Beides kann auch in einem Schritt erfolgen:

```
int ergebnis = wert * 2;
```

Da die Methode in ihrem Kopf den Rückgabotyp `int` angibt, muss die Methode am Ende mit `return` einen zu diesem Typ passenden Wert an den Aufrufer zurückliefern. Dies kann ein Wert, eine Variable oder eine Formel sein – hauptsächlich, der Ergebnistyp passt zum deklarierten **Rückgabotyp**. Daher könnten Sie alle drei Anweisungen der Methode zu einer einzigen zusammenfassen:

```
return (wert * 2);
```

Die Klammern sind hierbei optional, können aber (sparsam eingesetzt) die Übersichtlichkeit erhöhen. `void`-Methoden benötigen kein `return`, denn sie geben ja keinen Wert zurück. Solche Methoden können aber trotzdem mit `return;` (also ohne Wert) verlassen werden.

`static`-Methoden aus fremden Klassen können mit dem Klassennamen davor aufgerufen werden:

```
double wurzelAusVier = Math.sqrt( 4 );
```

A.3 Variablen und Datentypen

Sie haben gerade schon gesehen, dass Java eine **streng typisierte Programmiersprache** ist, d.h., Sie müssen sämtliche Variablen vor ihrer Verwendung deklarieren und dabei den Datentyp festlegen. Für Variablennamen gelten dieselben Konventionen wie für Methoden – sie sollten mit einem Kleinbuchstaben beginnen, bei mehreren Wörtern sind die Anfangsbuchstaben der Folgewörter groß. Wie alle Bezeichner in Java können Variablennamen beliebig lang sein und werden auch mit ihrem vollständigen Namen verglichen.

In Java gibt es acht eingebaute **einfache (primitive) Datentypen**: `byte`, `short`, `int` und `long` für ganze, vorzeichenbehaftete Zahlen. Alle Datentypen haben unabhängig vom Betriebssystem eine feste Größe und einen festen Aufbau. Bei den Ganzzahl-Typen sind die Größen beispielsweise 8, 16, 32 und 64 Bit. Als Fließkommatypen stehen `float` für einfache und `double` für dop-

pelte Genauigkeit zur Verfügung. Mit `boolean` werden die Wahrheitswerte `false` und `true` behandelt, `char` speichert ein einzelnes Unicode-Zeichen (16 Bit). Wenn Sie mit Zahlen arbeiten, werden Sie normalerweise `int` für ganze Zahlen und `double` für Fließkommazahlen einsetzen.

Alle anderen Typen sind in Java **Referenzdatentypen**, also Objekte! Beispielsweise können Sie beliebig genaue Zahlen mit den Klassen `BigInteger` und `BigDecimal` aus dem Paket `java.math` verrechnen.

Java achtet als typstrenge Sprache darauf, dass Zuweisungen zwischen verschiedenen Datentypen nicht zu einem Informationsverlust führen. Wenn die nötige Umwandlung ohne Verlust durchgeführt werden kann, passiert dies automatisch:

```
int i    = 5;
double d = i;
```

Hier wird der `int`-Wert einer `double`-Variablen zugewiesen. Da der Wertebereich hierbei erweitert wird, kann Java einen automatischen (impliziten) »**Typecast**« (kürzer auch einfach »**Cast**« genannt) durchführen. Der umgekehrte Fall erfordert eine explizite Umwandlung durch den Programmierer, denn hierbei geht Genauigkeit verloren – die Nachkommastellen werden abgeschnitten (auch wenn sie einfach nur Null sind):

```
double d = 5.0;
int i    = (int)d;
```

Neben dem einfachen Abschneiden von Nachkommastellen können Sie Zahlen natürlich auch **runden**. Dazu können Sie diverse Methoden der Klasse `java.lang.Math` nutzen, beispielsweise `round()`, `floor()` und `ceil()`.

A.4 Fallunterscheidungen und Schleifen

Java besitzt natürlich das klassische Konstrukt zur **Fallunterscheidung**, `if-else`. Das können Sie einfach wie folgt nutzen:

```
if (a==3) Anweisung1;
else Anweisung2;
```

Die Bedingung muss in runde Klammern eingeschlossen sein und einen `boolean`-Ergebnis liefern. Sowohl nach `if` als auch nach `else` kann genau eine Anweisung folgen. Damit Sie später weitere Anweisungen zu den Zweigen hinzufügen können und zur besseren Übersichtlichkeit sollten Sie `if-else` immer wie folgt programmieren:


```

if (a==3) {
    Anweisung1;
}
else if (b==7) {
    Anweisung2;
}
else {
    Anweisung3;
}

```

Bei beiden Beispielen sind alle `else`-Zweige optional. Zusätzlich zu `if-else` gibt es auch eine mehrseitige Fallunterscheidung mit der `switch`-Anweisung.

Java kennt drei **Schleifen**formen: `for`, `while` und `do-while`. Das folgende Beispiel zählt mit allen drei Formen von 0 bis 9 (jeweils inklusive):

```

for (int i = 0; i <= 9; i++) {
    System.out.println( i );
}

```

Bei der `for`-Schleife sind alle Daten im Schleifenkopf angegeben: die Initialisierung, die Bedingung und die Zählweisung. Die Zählweisung verwendet den so genannten Post-Inkrement-Operator, der die angegebene Variable um eins hoch zählt. Beachten Sie, dass die Schleifenvariable `i` im Schleifenkopf deklariert wird und daher nicht außerhalb der Schleife genutzt werden kann.

```

int j = 0;
while (j <= 9) {
    System.out.println( j );
    j = j + 1;
}

```

Bei der `while`-Schleife müssen Sie die Zählvariable außerhalb deklarieren und innerhalb an der passenden Stelle hochzählen.

```

int k = 0;
if (k <= 9) {
    do {
        System.out.println( k );
        k += 1;
    } while ( k <= 9);
}

```

Die `do-while`-Schleife ist im Gegensatz zu den beiden anderen Formen nicht abweisend, d. h., sie wird auf jeden Fall einmal durchlaufen. Damit die drei Beispiele identisch sind, ist vorher noch die `if`-Abfrage nötig – die Schleife darf

nicht starten, wenn der Zählwert bereits zu Anfang zu hoch ist. Wenn man sicher sein kann, dass die Schleife garantiert immer einmal durchlaufen werden muss, können Sie auf diese Abfrage natürlich verzichten. Und noch eines sehen Sie hier: die dritte Möglichkeit, eine Variable hochzuzählen – mit dem `+=`-Operator.

A.5 Klassen und Objekte

Bei der prozeduralen Programmierung mit statischen Methoden lief der Code in der Klasse ab. Bei der **objektorientierten Programmierung** dient die Klasse nur noch als Schablone, die die Struktur und das Verhalten (d.h. die Daten und Methoden bzw. Attribute und Operationen) für ein Objekt festlegt. Sie müssen dann erst ein Objekt der Klasse erzeugen, um die Methoden nutzen zu können. **Objekte** nennt man auch Exemplare oder Instanzen einer Klasse, daher wird die Objekterzeugung auch Instanzierung genannt. Das folgende Beispiel zeigt eine Klasse, deren Objekte eine `long`-Zahl speichern und zurückgeben können:

```
public class PositiveZahl {
    // Objektvariablen:
    private long zahl;

    // Konstruktoren:
    public PositiveZahl() {
        this.zahl = 0L;
    }

    public PositiveZahl(long anfangswert) {
        init( anfangswert );
    }

    // Methoden:
    private void init(int wert) {
        if (wert < 0) wert = 0;
        this.zahl = wert;
    }

    public void setZahl(long neuerWert) {
        init( neuerWert );
    }

    public long getZahl() {
        return this.zahl;
    }
}
```

```

    }

    public String toString() {
        return "Zahl=" + this.zahl;
    }
}

```

Nach der Erzeugung eines Objekts muss dieses initialisiert werden. Dazu wird ein **Konstruktor** aufgerufen, der bei Java immer exakt so heißen muss wie die Klasse – aber Achtung: Auch wenn ein Konstruktor wie eine Methode aussieht, fehlt der Rückgabetypp! Es kann mehrere Konstruktoren geben, die jeweils unterschiedliche Parameter akzeptieren müssen (man spricht vom Überladen von Konstruktoren, was auch bei Methoden möglich ist). Dadurch können Sie ein Objekt auf verschiedene Arten initialisieren, aber für jedes Objekt wird nur einer der Konstruktoren aufgerufen – je nachdem, wieviele Parameter Sie mitgeben. Bei diesem Beispiel wäre die Objekterzeugung mit `new PositiveZahl()` oder mit `new PositiveZahl(42)` möglich. In einem der beiden Konstruktoren wird zum Setzen der **Objektvariablen** `zahl` die interne Hilfsmethode `init()` aufgerufen, die den übergebenen Wert auf Gültigkeit prüft.

Destruktoren gibt es in Java nicht – dieses von C++ bekannte Konzept macht bei Sprachen mit automatischer Speicherverwaltung keinen Sinn. Stattdessen besitzen Objekte häufig eine Methode zum Schließen von Kanälen oder Freigeben von Ressourcen (z.B. `close()` oder `dispose()`). Nachdem Sie diese Methode aufgerufen haben, können Sie das Objekt einfach vergessen – es wird dann bei Gelegenheit von der »Garbage Collection« (GC) gelöscht.

Es folgen mit `setZahl()` und `getZahl()` Methoden zum Setzen und Lesen der Objektdaten – die so genannten »**Setter**« und »**Getter**«. Nur über solche Setter und Getter sollte ein Zugriff auf die Daten möglich sein. Die Objektvariablen selbst sollten immer auf `private` gesetzt sein! Im Setter können Sie dann auch den übergebenen Wert auf Gültigkeit prüfen, und durch ein so kontrolliertes Setzen der Daten kümmert sich das Objekt selbst darum, immer einen konsistenten Zustand zu speichern – man spricht von **Kapselung**.¹

¹ Der hier programmierte konsistente Zustand ist eigentlich, dass eine nicht negative Zahl gespeichert wird (und keine positive, wie der Klassenname suggeriert). Dieser Unterschied ist aber wohl nur für Mathematiker relevant.

In den Methoden wird **this** verwendet, die Selbst-Referenz jedes Objekts. Sie können damit ausdrücken, dass Sie bestimmte Objekteigenschaften genau dieses Objekts meinen – und nicht irgendeines anderen Objekts, was durch Assoziationen (siehe Abschnitt A.5.2) möglich wäre.

Die Klasse besitzt keine `main()`-Methode, kann also nicht gestartet werden. Zum Aufrufen bzw. zum Testen der Klasse `PositiveZahl` schreiben Sie daher eine zweite Klasse in einer eigenen Datei:

```
public class PositiveZahlTest {
    public static void main(String[] args) {
        PositiveZahl pz1 = new PositiveZahl();
        PositiveZahl pz2 = new PositiveZahl( 23 );
        pz1.setZahl( 42 );
        System.out.println( pz1.toString() );
        System.out.println( pz2 );
    }
}
```

Bei der Programmausführung starten Sie nun nur diese Testklasse, in deren `main()`-Methode zwei `PositiveZahl`-Objekte erzeugt werden. Der Zugriff auf Objekteigenschaften geschieht immer über eine Referenz, den Punkt-Operator und den Methodennamen, hier also z.B. `pz1.getZahl()` (beachten Sie, dass Sie auch beim Aufruf einer Methode leere Klammern angeben müssen, wenn diese keine Parameter annimmt).

In der letzten Anweisungszeile wird die Referenz `pz2` auf dem Bildschirm ausgegeben. Da Java nicht weiß, wie Objekte allgemein angezeigt werden sollen, ruft die Laufzeitumgebung an dieser Stelle automatisch die Methode `toString()` auf, mit der das Objekt selbst seine Daten als druckbare Zeichenkette zurückgibt (in der Zeile darüber ist der Aufruf noch explizit programmiert). Der Methodename ist von der Standard-Klassenbibliothek exakt festgelegt, und Sie können in Ihren Klassen eine eigene `toString()`-Methode bereitstellen – wie hier geschehen. Tun Sie dies nicht, erbt Ihre Klasse eine Standardimplementierung dieser Methode.

A.5.1 Vererbung

Bei der **Vererbung** erweitert eine Klasse (die **Unterklasse** oder Subklasse) eine bereits bestehende Klasse (die dadurch zur **Oberklasse** wird, auch Super- oder Basisklasse genannt) – ohne den Quelltext der Oberklasse zu verändern. Die Vererbungsbeziehung zwischen zwei Klassen wird durch das Schlüsselwort `extends` hergestellt:

```

public class PositiveZahlMitGeschmack extends PositiveZahl {
    public static final String SUESS = "süß";
    public static final String SAUER = "sauer";

    private String taste;

    public PositiveZahlMitGeschmack(long zahl, String taste) {
        super( zahl );
        this.taste = taste;
    }

    public String toString() {
        return super.toString() + ", Geschmack=" + taste;
    }
}

```

In der Unterklasse tauchen nur noch die neuen Eigenschaften der Unterklasse auf. Die Eigenschaften der Oberklasse wurden geerbt und können hier verwendet werden, als wenn sie hier programmiert wären (d.h., es ist auch ein Zugriff mit `this` möglich).

Auch wenn die Unterklasse sonst alle Eigenschaften der Oberklasse erbt – die **Konstruktoren** werden nicht vererbt. Sie müssen pro Klasse alle gewünschten Konstruktoren neu programmieren. Allerdings können Sie auf die Konstruktoren der Oberklasse mit `super()` zugreifen, was dazu verwendet wird, um Initialisierungswerte an die Oberklasse weiterzureichen (man spricht von **Konstruktorverkettung**). Je nachdem, welche Parameter Sie bei dem Aufruf mitgeben, wird der passende Konstruktor der Oberklasse ausgeführt. Wenn Sie `super()` verwenden, muss dies die erste Anweisung sein. Lassen Sie den Aufruf weg, fügt der Compiler automatisch den `super()`-Aufruf (ohne Parameter) ein.

Die Methode `toString()` ersetzt die gleichnamige Methode in der Oberklasse für Objekte der Unterklasse – man spricht vom **Überschreiben** einer Methode. Damit die ersetzte Funktionalität dennoch nicht verloren ist, können Sie innerhalb der überschreibenden Methode mit `super.toString()` darauf zugreifen.

In Java kann es immer nur eine direkte Oberklasse geben. Damit entfallen die Probleme, die beispielsweise C++ mit der Mehrfachvererbung hat. Damit Java trotzdem nicht weniger mächtig ist, wurde das Konzept der Interfaces (siehe Abschnitt A.5.4) eingeführt.

Was an diesem Quelltext noch auffällt, sind die beiden **Konstanten** SUESS und SAUER am Anfang. Durch die Kennzeichnung `public static` ist eine Variable öffentlich und ohne Objektbezug nutzbar, durch `final` ist nur eine einzige Wertzuweisung erlaubt – der Wert ist dann konstant. Die Java-Programmierrichtlinien besagen, dass Konstanten immer komplett großgeschrieben werden, mehrere Wörter werden durch Unterstriche getrennt.

In der Klasse `PositiveZahlMitGeschmack` fehlen sicherlich noch Setter und Getter für die Objektvariable `taste`, was aber hier der Übersichtlichkeit halber weggelassen wurde. Im Testprogramm würde ein Objekt dieser Klasse wie folgt erzeugt werden:

```
PositiveZahlMitGeschmack pzm;
pzm = new PositiveZahlMitGeschmack(
    7, PositiveZahlMitGeschmack.SUESS );
System.out.println( pzm );
```

Bei der Ausgabe der `pzm`-Referenz wird automatisch wieder die Methode `toString()` aufgerufen, was folgende Ausgabe zur Folge hat:

```
Zahl=7, Geschmack=süß
```

A.5.2 Assoziationen

Assoziationen bezeichnen Beziehungen zwischen Objekten – ein Objekt kennt ein anderes, um darin Methoden aufzurufen. Im Quelltext wird dies einfach durch eine Objektvariable gelöst, die eine Referenz auf das andere Objekt speichert:

```
import java.awt.Color;
public class PositiveZahlMitFarbe extends PositiveZahl {
    private Color farbe;

    public PositiveZahlMitFarbe(Color farbe) {
        super( 1L );
        this.farbe = farbe;
    }

    public String toString() {
        return super.toString() + ", Farbe=" + farbe;
    }
}
```

Dadurch, dass diese Klasse eine Referenz auf ein `Color`-Objekt speichert, kann ein Objekt dieser Klasse das Farb-Objekt benutzen (umgekehrt geht das nicht – es sei denn, das fremde Objekt speichert auch eine Referenz auf dieses Objekt).

Der Konstruktor übergibt an die Oberklasse den Wert `1L`, d.h. die Zahl Eins als `long`-Wert. Damit ein `PositiveZahlMitFarbe`-Objekt zur Laufzeit wirklich ein `Color`-Objekt kennt, müssen Sie ihm eine Referenz auf ein fertig erzeugtes `Color`-Objekt bei der Erzeugung mitgeben:

```
PositiveZahlMitFarbe pzmf1, pzmf2;  
pzmf1 = new PositiveZahlMitFarbe( Color.green );  
pzmf2 = new PositiveZahlMitFarbe( new Color( 255,0,0 ) );
```

Der erste Konstruktor erhält ein in der Standard-Klassenbibliothek vordefiniertes konstantes Farb-Objekt. Seit Java 1.4 können Sie alternativ auch `Color.GREEN` verwenden, Sun hält sich nun also auch endlich an seine Konventionen.

A.5.3 Polymorphismus

Die drei obigen Klassen kann man nun verwenden, um Polymorphismus (Vielfältigkeit) zu zeigen, was bei Java standardmäßig immer aktiv ist (falls Sie nicht einzelne Methoden oder ganze Klassen als `final` kennzeichnen). Betrachten Sie folgenden Testcode:

```
PositiveZahl pz;  
pz = new PositiveZahl();  
System.out.println( pz );  
pz = new PositiveZahlMitFarbe( Color.yellow );  
System.out.println( pz );  
pz = new PositiveZahlMitGeschmack( 2, "salzig" );  
System.out.println( pz );
```

Als Ausgabe erhalten Sie folgende drei Zeichenketten:

```
Zahl=0  
Zahl=1, Farbe=java.awt.Color[r=255,g=255,b=0]  
Zahl=2, Geschmack=salzig
```

Eigentlich ist nur das erste erzeugte Objekt ein `PositiveZahl`-Objekt, das direkt zum Typ der Referenzvariablen `pz` passt. Durch die Vererbungsbeziehung sind die Unterklassen aber kompatibel mit der Oberklasse, denn sie besitzen alle Methoden, die über die `pz`-Referenz aufgerufen werden können – wenn sie die Methoden nicht selbst überschreiben, erben sie diese. Daher erlaubt Java generell, dass Unterklassen-Objekte einer Oberklassen-Referenz

zugewiesen werden können. Wichtig: Das Unterklassen-Objekt wird dabei nicht verändert, Sie als Programmierer ändern nur Ihre Sichtweise auf das Objekt!

Obwohl Sie also immer über dieselbe Referenz die Methode `toString()` aufrufen (implizit bei der Ausgabe der Referenz bei `println()`), wird jeweils die Methode des aktuell in der Referenz gespeicherten Objekts ausgeführt. Diesen dynamischen Methodenaufruf nennt man Polymorphismus – Sie sagen dem Compiler, welcher Methodenname aufgerufen werden soll, und zur Laufzeit wird unter diesem Namen dann der korrekte, aber je nach Objekt verschiedene Code ausgeführt.

In der zweiten Zeile der Ausgabe sehen Sie übrigens, welche Zeichenkette die `toString()`-Methode der vordefinierten Klasse `Color` zurückgibt.

A.5.4 Abstrakte Klassen und Interfaces

Abstrakte Klassen dienen dazu, unvollständige Oberklassen zu schaffen, die dann in Unterklassen jeweils spezialisiert vervollständigt werden müssen. Das Besondere bei Java dabei ist, dass dies vom Compiler überwacht wird! Eine abstrakte Klasse enthält normalerweise mindestens eine abstrakte Methode, die mit dem Schlüsselwort `abstract` gekennzeichnet wird und die anstelle des Methodenrumpfs nur ein Semikolon bekommt:

```
public abstract class Zahl {
    private int wert;
    public int getWert() {
        return this.wert;
    }
    public abstract double getPreis();
}
```

In dieser Klasse wurden der Konstruktor und der Setter weggelassen. Weil Sie keinen Konstruktor programmiert haben, wird vom Compiler automatisch ein Standardkonstruktor (ohne Parameter) generiert.

Dennoch kann von dieser Klasse kein Objekt erzeugt werden, denn dadurch würde sich ein unvollständiges Objekt im Speicher befinden, bei dem Sie nicht existenten Code aufrufen könnten – ein Absturz wäre vermutlich die Folge. Daher verhindert bereits der Compiler das Erzeugen eines solchen Objekts. `new Zahl()` wird nicht kompiliert, Sie erhalten stattdessen eine Fehlermeldung!

Wenn Sie aber eine Unterklasse schreiben, die von `Zahl` erbt und die fehlende Methode `getPreis()` implementiert, können Sie nun von der Unterklasse wie gewohnt Objekte erzeugen:

```
public class KommerzielleZahl extends Zahl {
    public double getPreis() {
        return Math.abs(this.wert) / 10.0;
    }
}
// Test in main():
Zahl z1 = new Zahl(); // Compilerfehler
Zahl z2 = new KommerzielleZahl(); // OK
```

Die abstrakte Oberklasse wird hier als Datentyp verwendet, der eine gewisse Mindestschnittstelle vorgibt. Ein Methodenaufruf dieser Schnittstelle kann dann unabhängig von der tatsächlichen Implementierung programmiert werden. Und zur Laufzeit wird dann die korrekte Methode der jeweiligen Unterklasse – polymorph! – aufgerufen:

```
Zahl z;
// irgend ein Unterklassen-Objekt zuweisen...

System.out.println( z.getPreis() );
```

Interfaces (Schnittstellen, bei anderen Sprachen auch Protokoll genannt) erfüllen in etwa denselben Zweck, sie werden aber nicht als `class`, sondern als `interface` programmiert:

```
public interface Kostenpflichtig {
    public double getPreis();
}
```

In einem Interface gibt es keinerlei implementierenden Code und keine Variablen – nur Methodendeklarationen (die so genannten Signaturen). Sie sehen wie abstrakte Methoden aus, das Schlüsselwort `abstract` fehlt hier aber. Eine Klasse kann ein solches Interface hinter `implements` auflisten und verpflichtet sich damit, für alle Methoden des Interfaces Programmcode bereitzustellen (ansonsten wäre die Klasse unvollständig und müsste `abstract` deklariert werden):

```
public class Buch implements Kostenpflichtig {
    private String titel, autor;
    public double getPreis() {
```

```

        return 34.95;
    }
}

```

In der Klasse können natürlich beliebige Erweiterungen (Setter, Getter, Konstruktoren usw.) auftauchen – wichtig ist nur, dass die Methoden des Interfaces implementiert sind. Eine Klasse kann auch mehrere Interfaces implementieren, die Namen werden dann durch Komma getrennt hinter `implements` angegeben.

Verwendet wird ein Interface wie bei der abstrakten Oberklasse, es wird der dynamische Methodenaufruf und Polymorphismus genutzt:

```

Kostenpflichtig i = new Buch();
System.out.println( i.getPreis() );

```

Die implementierenden Klassen sind also datentypkompatibel zum Interface, wie dies schon bei Unterklassen und ihren Oberklassen der Fall war. Und wie bei der Vererbung wird durch Interfaces eine »ist ein«- bzw. »ist«-Beziehung aufgebaut: Ein Buch »ist« Kostenpflichtig.

Der große Vorteil von Interfaces besteht darin, dass Sie damit zum einen gemeinsame Schnittstellen zwischen ganz unterschiedlichen Klassen schaffen – und zum anderen die Vererbung mit `extends` immer noch nutzen können! Bei der Klasse Buch könnte dies wie folgt aussehen:

```

public class Buch
    extends IrgendeineOberklasse
    implements Kostenpflichtig, EinWeiteresInterface
    { /*...*/ }

```

A.5.5 Innere und anonyme Klassen

Bisher hatten Sie es nur mit Klassen zu tun, die alleine oder nacheinander in einer Datei programmiert waren. Es ist aber auch möglich, eine Klasse in einer anderen zu verschachteln:

```

public class Aussen {
    private int aussenwert;

    public void tuWas() {
        Runnable r = new Innen();
    }
}

```

```

class Innen implements Runnable {
    public void run() {
        System.out.println( aussenwert );
    }
}

```

Die Besonderheit bei **inneren Klassen** ist, dass diese Zugriff auf die Objektvariablen des umgebenden Objekts haben, selbst wenn diese `private` sind! Das heißt aber gleichzeitig auch, dass immer ein umgebendes Objekt existieren muss – daher werden Objekte innerer Klassen fast ausschließlich in Objektmethoden der umgebenden Klasse erzeugt, wie hier in `tuWas()`.

Die Klasse `Innen` greift in ihrer Methode `run()` auf die Objektvariable `aussewert` der umgebenden Klasse `Aussen` zu. Das Interface `java.lang.Runnable` stammt dabei aus der Thread-Programmierung, auch wenn Threads hier nicht zum Einsatz kommen.

Wenn Sie die innere Klasse nur an einer einzigen Stelle benötigen, können Sie die innere Klasse auch als anonyme Klasse direkt innerhalb der Methode programmieren, die diese Klasse benötigt:

```

public void tuWas() {
    Runnable r = new Runnable() {
        public void run() {
            System.out.println( aussenwert );
        }
    };
}

```

Hier sieht es so aus, als ob ein Objekt eines Interfaces erzeugt wird (was natürlich nicht möglich ist) – aber zu diesem Interface wird die passende Implementierung sofort dahinter in geschweiften Klammern programmiert. Das Einzige, was fehlt, ist der Name für diese Klasse – daher nennt man sie **anonyme Klasse**. Auch anonyme Klassen sind innere Klassen und haben somit Zugriff auf die Variablen des umgebenden Objekts. Da die Klasse in einer Methode definiert wird, spricht man auch von lokalen Klassen. Wenn Sie darin auf lokale Variablen der umgebenden Methode zugreifen wollen, müssen Sie diese Variablen `final` deklarieren.

Anonyme Klassen werden häufig direkt als Parameter bei Methoden- oder Konstruktoraufrufen übergeben – beispielsweise beim Registrieren von Listener-Objekten bei der AWT-Programmierung. Wenn Sie ein `Runnable`-Objekt als

Parameter für einen Thread-Konstruktor verwenden wollen, könnte das wie folgt aussehen:

```
public void irgendwo() {
    final int lokal = 24;
    Thread t = new Thread( new Runnable() {
        public void run() {
            System.out.println( lokal );
        }
    });
    t.start();
}
```

A.6 Fehlerbehandlung mit Exceptions

Java besitzt einen ausgefeilten Mechanismus zur Behandlung von Laufzeitfehlern: Exceptions (Ausnahmefehler). Innerhalb eines kritischen Blocks (`try`) werden potenziell fehlerhafte Aufrufe ausgeführt. Sobald ein Fehler auftritt, wird der `try`-Block abgebrochen – die restlichen Anweisungen des Blocks werden übersprungen. In einem `catch`-Block kann dann auf den Fehler reagiert werden. Und egal ob ein Fehler aufgetreten ist oder nicht, am Ende wird immer der `finally`-Block für Aufräumarbeiten durchlaufen. Die Fehlerbehandlung wird generell innerhalb von Methoden durchgeführt:

```
public void irgendwo() {
    try {
        ventil.oeffnen();
        strom.erzeugen();
        // ...
    }
    catch (Exception e) {
        e.printStackTrace();
        // Fehler protokollieren
    }
    finally {
        ventil.schliessen();
    }
}
```

Nach einem `try`-Block muss mindestens entweder ein `catch`- oder der `finally`-Block auftauchen. Es kann mehrere `catch`-Blöcke mit unterschiedlichen `Exception`-Unterklassen geben, von denen dann der zum Fehler jeweils

passendste Block ausgeführt wird. Alternativ kümmert sich – wie hier – ein einziger `catch`-Block mit der Oberklasse `Exception` um alle Fehler. Bei den Fehlerklassen merken Sie es schon: Fehler werden in Java nicht als Zahlen oder Zeichenketten gemeldet, sondern als Objekte.

Nicht behandelte Exceptions werden zur aufrufenden Methode weitergereicht. Wenn ein solches Fehlerobjekt schließlich aus der `main()`-Methode herausgegeben wird, führt dies zum Programmabbruch durch das Laufzeitsystem. Wenn eine Methode nicht alle Fehler behandelt (oder die Exceptions vielleicht sogar bewusst erzeugt, um einen Fehlerzustand zu melden), muss sie dies im Methodenkopf signalisieren:

```
public void bla() throws DeineException {
    try {
        blub();
    }
    catch (MeineException e) {
        System.err.println( "Aua: " + e );
    }
}

public void blub() throws MeineException, DeineException {
    if (ich.kopfschmerzen()) {
        throw new MeineException();
    }
    else if (du.kopfschmerzen()) {
        throw new DeineException();
    }
    System.out.println("Alles OK! Sonnenschein!");
}
```

Hinter `throws` tragen Sie alle Exceptions ein, die potenziell von der Methode gemeldet werden können. Das sind alle Exceptions, die nicht in der Methode behandelt werden – egal ob sie nun in der Methode selbst erzeugt werden (wie bei `blub()`) oder ob die Methode nicht alle der empfangenen Exceptions behandelt (wie bei `bla()`). Nach dem Werfen einer Exception mit `throw` wird die Methode sofort an dieser Stelle beendet.

Wenn Sie wie hier im Beispiel eigene Fehlerklassen definieren wollen, leiten Sie einfach eine Unterklasse von der Klasse `java.lang.Exception` ab.

A.7 Literatur & Links

- ▶ B. Steppan, »Einstieg Java 5« 2. Aufl., Galileo Computing 2004
C. Ullenboom, »Java ist auch eine Insel« 4. Aufl., Galileo Computing 2004
Beide Bücher wurden schon in Kapitel 1, *Grundlagen*, vorgestellt und können prima zum Lernen und zum Nachschlagen genutzt werden.
- ▶ Ratz/Scheffler/Seese, »Grundkurs Programmieren in Java« Band 1 & 2, Hanser 2001/2003
In zwei Bänden führt Sie dieses Werk sehr ausführlich durch alle wichtigen Java-Themen – von Variablen und Schleifen bis hin zu Threads und Netzwerkprogrammierung.
- ▶ D. Abts, »Grundkurs Java«/»Aufbaukurs Java«, Vieweg 1999/2003
Wenn Sie schon einige (wenige) Vorkenntnisse mitbringen, geleiten Sie diese beiden Bände kurz und präzise durch alle Java-Bereiche – von einfachen Klassen bis hin zu Datenbanken und Netzwerkprogrammierung.
- ▶ <http://www.boku.ac.at/javaeinf/jein.html>
Die »Java Einführung« ist ein ausführlicher Java-Grundkurs, der komplett online und zum Herunterladen verfügbar ist.
- ▶ <http://www.javabuch.de/>
Das »Handbuch der Java-Programmierung« können Sie auch als Buch erwerben. Auf dieser Seite steht die HTML-Ausgabe zum freien Download zur Verfügung.
- ▶ <http://www.mindview.net/Books/TIJ/>
»Thinking in Java« ist eine der bekanntesten englischsprachigen Java-Anleitungen im Web – und kann auch als Buch erworben werden.
- ▶ <http://java.sun.com/learning/new2java/>
Sun zeigt Ihnen mit wenigen (englischsprachigen) Schritten alle wichtigen Themen für einen erfolgreichen Einstieg in die Java-Technologie.
- ▶ <http://java.sun.com/docs/codeconv/>
Von dieser Seite können Sie sich die allgemein akzeptierten Java-Programmierrichtlinien (»Code Conventions«) herunterladen. Wenn Sie sich daran halten, werden sich andere Java-Programmierer besser in Ihren Quelltexten zurechtfinden.
- ▶ <http://www.dclj.de/faq.html>
Teilnehmer der Newsgroup `de.comp.lang.java` (»dclj«) haben Antworten auf die häufigsten Fragen zu Java zusammengestellt.

B Java auf Mac OS 8/9/Classic


*»Kommt Zeit, kommt Rat, kommt Fusselbart/
Kommt Rasierapparat und vergessen ist alles, was früher war.«
(Element Of Crime)*

Java kann auf Apple-Rechnern seit 1997 genutzt werden, und es hat – wie Sie schon in Kapitel 1, *Grundlagen*, erfahren haben – eine bewegte Geschichte hinter sich. Apple hat in den vergangenen zehn Jahren nicht nur einen kompletten Wechsel auf eine andere Prozessorarchitektur vollzogen (von 68k- zu PPC-Prozessoren), sondern seit dem Jahr 2000 auch das Betriebssystem gewechselt (vom mittlerweile 20 Jahre alten Mac OS, einer reinen Apple-Entwicklung, zu Mac OS X, das auf FreeBSD, einer UNIX-Variante, basiert). Den ersten Wechsel hat Apples ursprüngliche Java-Implementierung, die »Macintosh Runtime for Java« (später »Mac OS Runtime for Java« genannt), kurz MRJ, noch überlebt. Mit der Umstellung auf Mac OS X hat Apple aber auch eine komplett neue Java-Umgebung entwickelt, und MRJ ist seitdem Geschichte.

MRJ wurde erstmals mit Mac OS 8 ausgeliefert und kann bis Mac OS 9 genutzt werden. Auch wenn MRJ die Version 2.2 erreichte, wird damit nur das JDK 1.1.8 implementiert und ist somit für aktuelle Entwicklungen uninteressant. *Da Apple beides, sowohl MRJ als auch Mac OS 9, nicht mehr weiterentwickelt, handelt es sich also um ein totes System!* Zunehmend steigen die Mac-Anwender auf Mac OS X um – oft beim Kauf eines neuen Rechners, der gleich mit Mac OS X ausgeliefert wird und genug Leistung bietet, um aktuelle Applikationen ausführen zu können. Trotzdem finden Sie hier noch einen kurzen Überblick, wie Java auf den alten Mac-Systemen genutzt wurde und wie Sie damit auch heute noch bestehende Anwendungen testen können.

Mac OS booten

Wie können Sie also das alte Mac OS nutzen? Alle bis Ende 2002 verkauften Rechner konnten Mac OS direkt starten, seit 2003 nur noch wenige, speziell dafür hergestellte Modelle. 2004 dürfte kein einziger neuer Apple-Computer mehr das alte Mac OS booten können.¹

Wenn Sie einen hinreichend alten Mac besitzen, können Sie beim Rechnerstart einfach die -Taste gedrückt halten, bis der Boot-Manager erscheint. Dieser zeigt Ihnen alle installierten, startfähigen Systeme an, und das kann eben auch

¹ Das Dokument <http://docs.info.apple.com/article.html?artnum=86209> beschreibt, welche Rechner nur noch Mac OS X booten können.

ein Mac OS 9 sein. Wählen Sie das System aus, klicken Sie auf den Rechtspfeil, und das alte Mac OS wird gestartet.

Alternativ öffnen Sie die Mac OS X-Systemeinstellungen. Wie Abbildung B.1 zeigt, können Sie im Bereich »Startvolume« das System für künftige Rechnerstarts festlegen. Beide Varianten funktionieren wie gesagt nur, wenn Ihr Rechner überhaupt noch ein altes Mac OS starten kann, ansonsten werden Ihnen nur Mac OS X-Systeme angeboten.



Abbildung B.1 Startvolume in Mac OS X zum Booten nach Mac OS 9 ändern

Nach dem Neustart arbeiten Sie ganz normal in einem eigenständigen Betriebssystem, das sich allerdings deutlich von Mac OS X unterscheidet. Wenn Sie aus dem alten Mac OS zurück zu Mac OS X wechseln möchten, können Sie dazu entweder wieder den Boot-Manager verwenden, oder aber Sie wählen im Apfel-Menü unter »Kontrollfelder« den Eintrag »Startvolume« aus. Sie sehen dann wie in Abbildung B.2 das entsprechende Mac OS-Kontrollfeld, in dem Sie als Startvolume wieder Mac OS X auswählen können.

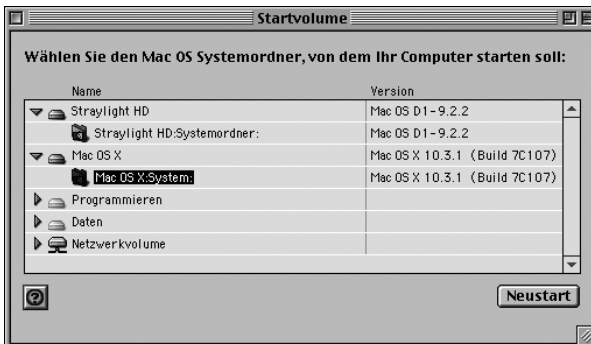


Abbildung B.2 Startvolume in Mac OS zum Booten nach Mac OS X ändern

Classic starten

Dieses ständige Betriebssystem-Wechseln und Rechner-Neustarten ist aufwändig. Trotzdem wird Mac OS 9 noch eine Weile weiterleben, und zwar in der sogenannten »Classic«-Umgebung von Mac OS X. Classic ist ein eigener Mac OS X-Prozess, in dem ein komplettes Mac OS 9 läuft (dieser Prozess wird auch »TruBluEnvironment« genannt, wie Sie im Terminal mit `ps -x` feststellen können). Durch diese Umgebung können Sie alte Mac OS-Applikationen parallel zu Ihren Mac OS X-Anwendungen benutzen. Classic kümmert sich darum, dass die alten Programme möglichst nahtlos in Apples aktuelles Betriebssystem integriert werden.

Ob Sie Mac OS 9 installiert haben und damit Classic verwenden können, sehen Sie am einfachsten daran, dass sich auf Ihrer Festplatte die Verzeichnisse `Systemordner` und `Applications (MacOS 9)` befinden. Meistens liegen diese auf demselben Volume wie Mac OS X, aber wie Apples aktuelles Betriebssystem konnten auch schon die alten Versionen auf beliebigen Volumes installiert werden. Bis Mac OS X 10.1 wurde zusätzlich Mac OS 9 mitgeliefert, mittlerweile müssen Sie das alte System separat erwerben. Wenn Sie aber Mac OS 9.1 (oder neuer) auf CD besitzen, können Sie dieses auch nachträglich noch unter Mac OS X 10.2 oder 10.3 installieren.²

Gestartet wird Classic normalerweise ganz einfach dadurch, dass Sie eine alte Mac-Applikation per Doppelklick ausführen lassen. Wenn Sie darüber lieber etwas mehr Kontrolle haben möchten, öffnen Sie **Systemeinstellungen · Classic**. Dort können Sie die Classic-Umgebung mit dem »Start«-Knopf explizit aufrufen und ein paar sinnvolle Einstellungen vornehmen.

Da Sie vermutlich nicht mehr für das alte Mac OS entwickeln wollen, sollten Sie Classic beim Anmelden *nicht* automatisch starten lassen. Aus genau diesem Grund ist es auch sinnvoll, wenn Sie sich vor dem Classic-Start warnen lassen – falls Sie aus Versehen eine alte Mac-Applikation aufrufen, können Sie dies 30 Sekunden lang abbrechen. Vor allem auf älteren Rechnern dauert das Aktivieren von Classic nämlich durchaus eine Minute oder länger ... Sicherheits halber sollten Sie dann noch im Bereich »Weitere Optionen« den Ruhezustand auf »Nie« einstellen, sonst gibt es unter Umständen Probleme, wenn Ihr Rechner aus dem Ruhezustand aufwachen soll.

² Apple listet auf <http://docs.info.apple.com/article.html?artnum=25517> auf, welche Rechner mit welchen Systemversionen ausgeliefert wurden.



Abbildung B.3 Systemeinstellungen für die Classic-Umgebung

Wie auch immer Sie Classic gestartet haben, Sie sollten nun ein Fenster wie in Abbildung B.4 sehen, in dem Mac OS 9 geladen wird. Am unteren Rand des Fensters werden die aktiven Mac OS-Systemerweiterungen angezeigt (bei einem wenig genutzten MacOS sind das deutlich weniger als hier gezeigt). Sobald Classic komplett gestartet wurde, verschwindet das Fenster wieder. Und falls Sie bis dahin nur ein kleineres Fenster sehen, blenden Sie die Anzeige durch Anklicken des Dreiecks ein.



Abbildung B.4 Start der Classic-Umgebung von Mac OS X

MRJ verwenden

Nun können Sie den Apple Applet Runner im Verzeichnis /Applications (MacOS 9)/Apple Extras/MacOS Runtime For Java/Apple Applet Runner starten. Wie jede MacOS X-Applikation wird auch dieses MacOS-Programm im Dock angezeigt. Sobald aber eine alte Mac-Anwendung aktiv ist, wird plötzlich eine optisch ganz andere Menüleiste angezeigt, ebenso haben die Fenster solcher Classic-Applikationen ein anderes Aussehen.



Abbildung B.5 Mac OS X- und Classic-Anwendungen nebeneinander im Dock

Wenn Sie im Apple Applet Runner im »Applets«-Menü den Eintrag »Sun Tumbling Duke« anwählen, sollten Sie in etwa Abbildung B.6 zu sehen bekommen – in der Classic-Umgebung wird ein Applet mit Apples altem Java-System angezeigt. Und falls Sie gleichzeitig eine MacOS X-Anwendung laufen haben, bemerken Sie noch etwas: Bis auf die unterschiedliche Optik sind Classic-Programme für Anwender ganz normal und parallel zu benutzen! Einer der relevanten Unterschiede liegt aber in der Prozessverwaltung. Wenn eine MacOS X-Anwendung abstürzt, wird nur diese eine Anwendung aus dem Speicher entfernt, alle übrigen Applikationen merken davon nichts. Stürzt dagegen ein Classic-Programm ab, können andere Classic-Applikationen davon mit betroffen sein – innerhalb des Classic-Prozesses gibt es keinen separaten Speicherschutz für die einzelnen Classic-Anwendungen.

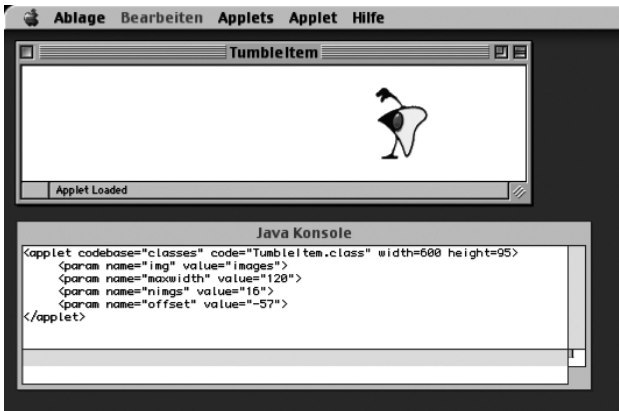


Abbildung B.6 Applet Runner (Mac OS 9)

Wenn Classic-Java-Programme Text mit `System.out.println()` oder ähnlichen Methoden ausgeben, geht ein spezielles Konsolenfenster für die Textausgabe auf, da das alte MacOS keine in das System integrierte Shell besitzt. Ebenso kennt das MacOS kein Java-Plugin. Jedes Programm, das Applets einbindet (beispielsweise ein Browser), muss die Applet-Einstellungen explizit setzen oder vom Benutzer konfigurieren lassen. Im Applet Runner erreichen Sie diese Konfiguration über **Ablage · Voreinstellungen**.³

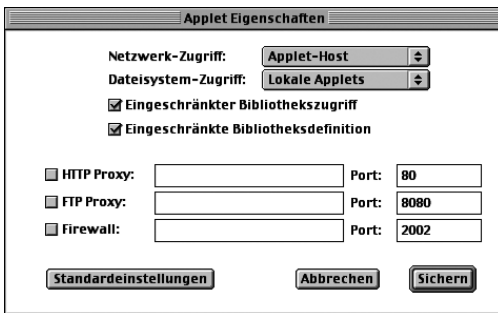


Abbildung B.7 Applet-Eigenschaften anstelle eines Java-Plugins

Das Einbinden der Applets und Setzen der Einstellungen geschieht über die »JManager«-Programmierschnittstelle, mit der eine Java Virtual Machine erzeugt werden kann. Die JManager-API ist somit ein Ersatz für einen Teil des Java Native Interface (JNI). Als Ergänzung zu JNI ist die »JDirect«-Schnittstelle zu sehen, mit der native Betriebssystemaufrufe aus Java heraus durchgeführt werden können, ohne Module in einer anderen Sprache (z.B. C) schreiben zu müssen. Bei MacOS X wurde JManager durch die Java Embedding API beziehungsweise durch das Java-Plugin ersetzt. JDirect steht zum Teil noch zur Verfügung, sollte aber nicht mehr benutzt werden.

Eine gute, wenn auch etwas veraltete Sammlung von Fragen und Antworten rund um MRJ findet sich im Web auf <http://www.ping.be/~ping0172/MRJ-FAQ.html>. Apple beschreibt die Classic-Java-Technologien auf folgenden Seiten:

► **MRJ**

<http://developer.apple.com/technicalnotes/LegacyTechnologies/idxJava-date.html>
<http://developer.apple.com/qa/indexes/Java11-a.html>

► **MRJToolkit**

<http://developer.apple.com/documentation/macos8/LegacyJava/LegacyAPI/MRJToolkit/mrjtoolkit.html>

³ Bei MacOS-Programmen befinden sich die Einstellungen also nicht im Programm-Menü! Leider ist der Applet Runner dazu noch eine ganz besondere Ausnahme, denn eigentlich sollten die Voreinstellungen von Classic-Applikationen im Bearbeiten-Menü auftauchen.

► **JBindery**

<http://developer.apple.com/documentation/macos8/LegacyJava/LegacyAPI/JBindery/JBindery.html>

► **JManager**

<http://developer.apple.com/documentation/macos8/LegacyJava/LegacyAPI/JManager/JManager.html>

► **JDirect**

<http://developer.apple.com/documentation/macos8/LegacyJava/LegacyAPI/JDirect/JDirect.html>

<http://developer.apple.com/technotes/tn/tn2002.html>

Wenn Sie tatsächlich noch für Mac OS Classic entwickeln müssen, sollten Sie folgende Systemeinschränkungen beachten:

- Dateinamen haben eine maximale Länge von 31 Zeichen! Glücklicherweise gilt diese Einschränkung nicht für Dateien in zip- oder jar-Archiven.
- Als Trennzeichen in Pfadangaben wird der Doppelpunkt verwendet, der Schrägstrich ist dagegen ein gültiges Zeichen für Dateinamen. Falls Sie bestehende Konfigurationsdateien, in denen solche Pfade und Dateinamen gespeichert sind, einfach auf ein Mac OS X-System (oder jedes andere Java-System) kopieren, sind diese Namen dort ungültig – die entsprechenden Pfade und Dateien werden nicht mehr gefunden.
- Das klassische Mac OS kennt nur kooperatives Multitasking. Ein laufender Prozess muss also freiwillig Rechenzeit abgeben, damit ein anderer Prozess zum Zug kommt. MRJ emuliert zwar das für ein Java-System nötige präemptive Multitasking, bei dem ein Thread jederzeit unterbrochen werden kann. Man merkt aber oft die Beschränkungen des zugrunde liegenden Systems, wenn es zu unerwarteten Hängern und Verzögerungen kommt.

Da MRJ bei der Implementierung des JDK 1.1.8 stehen geblieben ist, ist insbesondere die Swing-Bibliothek nicht automatisch vorhanden. Wenn Sie also Anwendungen mit einer solchen Benutzungsoberfläche verwenden wollen, müssen Sie zuvor Swing 1.1.1 von der Seite <http://java.sun.com/products/jfc/download.html> herunterladen. Zur Installation kopieren Sie dann am einfachsten das Archiv `swingall.jar` (oder die Einzelarchive) in das Verzeichnis `/Systemordner/Systemerweiterungen/MRJ Libraries/MRJClasses`.

Bei Java-Anwendungen, die sehr viel Speicher anfordern, kann es in der Classic-Umgebung zu Speicherproblemen kommen. Beim Start von Classic wird von Mac OS X so genannter »temporärer Speicher« reserviert, um die Programmausführung optimieren zu können. Normalerweise sind dies 128 MByte

(wenn der Rechner genug RAM besitzt), aber manchmal reicht dies nicht aus und Ihr Programm wird mit einem `java.lang.OutOfMemoryError` beendet.


Zum Erhöhen des temporären Speichers rufen Sie **Systemeinstellungen · Classic** auf und klicken dort dann mit gedrückter -Taste »Speicher/Versionen« an. Auf der Seite wird dann unten rechts die Schaltfläche »Speicher anpassen« angezeigt (siehe Abbildung B.8), klicken Sie diese an. Ziehen Sie den Schieber in Richtung »Temporärer Speicher« – Sie können auf diese Weise den temporären Speicher von ca. 48 MByte (Schieber ganz links) bis ca. 512 MByte (Schieber ganz rechts) festlegen. Mehr temporärer Speicher bedeutet allerdings auch, dass dann nicht mehr soviel Programmspeicher zur Verfügung steht, was bei anderen Programmen zu Problemen führen kann. Es gibt leider keine allgemeine Regel für die optimale Speicherverteilung, Sie werden dies ausprobieren müssen. Alle Änderungen werden erst beim nächsten Classic-Neustart aktiv.



Abbildung B.8 Speichereinstellungen für die Classic-Umgebung

Die letzten Updates

Auch wenn Mac OS 9 und MRJ nicht mehr weiterentwickelt werden, sollten Sie wenigstens deren neueste Versionen einsetzen, falls Sie Java in der Classic-Umgebung nutzen. Die aktuellste **Mac OS 9**-Version ist 9.2.2. Sie müssen mindestens Mac OS 9.1 einsetzen, um es in der Classic-Umgebung verwenden zu können. Auf der Seite <http://docs.info.apple.com/article.html?artnum=75288> listet Apple alle Updates der 9er-Reihe auf und beschreibt, in welcher Reihenfolge Sie Ihr System am besten auf Mac OS 9.2.2 aktualisieren.

Die letzte **MRJ**-Version ist 2.2.6, auch wenn auf Apples Webseiten nur die Version 2.2.5 verlinkt ist. Die neuere Version, die eine Fehlerbereinigung für Oracle-Client-Applikationen enthält, kann von <http://docs.info.apple.com/article.html?artnum=120209> heruntergeladen werden.

Die Entwicklerwerkzeuge schließlich sind bei Version 2.2 stehen geblieben. Das **MRJ SDK** enthält unter anderem JBindery, mit dem Sie Java-Klassen und -Bibliotheken als ausführbare Mac OS-Programme verpacken können. Auf der Seite <http://developer.apple.com/java/classic.html> finden Sie einen Link zum Download.

C Java 1.5 »Tiger«

»Baby you can turn me on!«
(Tom »The Tiger« Jones)

Mit dem Herbstanfang 2004 ist nicht nur mal wieder ein Sommer zu Ende, sondern auch das Warten auf Suns neue Java-Version, die auf <http://java.sun.com/> für die Betriebssysteme Windows, Linux und Solaris zur Verfügung steht. Sun hat bei der Versionsnummer die »1« weggelassen und nennt die neue Version nun **J2SE 5.0** (Codename »Tiger«).

Da Apple prinzipbedingt mit der Veröffentlichung neuer JDKs etwas hinterherhinkt, dürfte es noch ein wenig dauern, bis Java 1.5 auf dem Mac genutzt werden kann. Vermutlich kann man im Frühjahr 2005 mit Apples Umsetzung rechnen – wahrscheinlich zeitgleich mit dem kommenden MacOS X 10.4, das ebenfalls den Codenamen »Tiger« trägt.

Sie müssen auf Mac OS X aber nicht auf die spannenden und nützlichen neuen Eigenschaften von Java 1.5 verzichten, denn Sie können eine allgemein verfügbare Vorabversion des Compilers und der Laufzeitumgebung problemlos zum Laufen bekommen! Wie das geht, wurde mehrfach in diversen Sun-Foren und in Apples Java-Mailingliste beschrieben.¹ Im Folgenden sehen Sie darauf aufbauend eine vereinfachte Version, die zum einen mit Java 1.4.2 zurechtkommt, und die zum anderen eine bestehende Java-Installation nicht verändert.

Herunterladen und installieren

Laden Sie sich von http://java.sun.com/developer/earlyAccess/adding_generics/ die »Early Access«-Version des JSR014-Prototyps herunter. Bei der folgenden Anleitung wird davon ausgegangen, dass es sich um das Archiv `adding_generics-2_4-ea.zip` handelt.

Packen Sie das Archiv aus und legen Sie es beispielsweise in Ihr Benutzer-Verzeichnis. Es sollte nun also einen Ordner `/Users/much/adding_generics-2_4ea/` geben, wobei Sie das »much« natürlich durch Ihren Benutzer-Kurznamen ersetzen müssen.

Skripte anpassen

Ersetzen (!) Sie nun die UNIX-Shell-Skripte `bootstrap`, `javac` und `java` im Verzeichnis `adding_generics-2_4ea/scripts/` durch folgende Versionen, die

¹ <http://lists.apple.com/archives/java-dev/2004/Feb/05/j2se15genericsworksunder.txt>

Sie auch auf der Buch-CD im Verzeichnis `examples/ch16/scripts/` finden. Die fett gedruckten Pfadangaben müssen Sie an Ihre Installation anpassen.

```
#!/bin/sh
JSR14DISTR="/Users/much/adding_generics-2_4ea"
mkdir -p newrt
sh -x ${JSR14DISTR}/scripts/javac -g -d newrt \
    -sourcepath src `find src -name \*.java -print`
( cd > /dev/null src ; tar cf -
  `find . -name \*.java -prune -o \
    -type f -print` ) | ( cd > /dev/null newrt ; tar xf - )
cd newrt
jar cf ../newrt.jar *
cd ..
rm -rf newrt
```

Listing C.1 Shell-Skript bootstrap

```
#!/bin/sh
JSR14DISTR="/Users/much/adding_generics-2_4ea"
J2SE14HOME="/Library/Java/Home"
J2SE14CURR="/System/Library/Frameworks/JavaVM.framework/Versions
/CurrentJDK/"
if [ ! "$*" ]; then
    exec ${J2SE14HOME}/bin/javac \
        -J-Xbootclasspath/p:${JSR14DISTR}/gjc-rt.jar
fi
exec ${J2SE14HOME}/bin/javac \
    -J-Xbootclasspath/p:${JSR14DISTR}/gjc-rt.jar \
    -bootclasspath ${JSR14DISTR}/gjc-rt.jar:${JSR14DISTR}/
    collect.jar:${J2SE14CURR}/Classes/classes.jar \
    -source 1.5 -target jsr14 "$@"
```

Listing C.2 Shell-Skript javac

```
#!/bin/sh
JSR14DISTR="/Users/much/adding_generics-2_4ea"
J2SE14HOME="/Library/Java/Home"
exec ${J2SE14HOME}/bin/java \
    -Xbootclasspath/p:${JSR14DISTR}/gjc-rt.jar "$@"
```

Listing C.3 Shell-Skript java

Die Skripte sind insbesondere dahin gehend angepasst, dass sie mit `/Library/Java/Home` das neueste installierte JDK verwenden. Und da die Java-Implementierung von MacOSX kein `rt.jar` kennt, greifen die Skripte auf `/System/Library/Frameworks/JavaVM.framework/Versions/CurrentJDK/Classes/classes.jar` zurück.

Die beiden Skripte `javac` und `java` kopieren Sie nun in das Verzeichnis `/usr/local/bin` – allerdings unter einem anderen Namen (`tigerc` und `tiger`), damit Sie bequem zwischen den 1.4er- und den 1.5er-Tools unterscheiden können:

```
cd ~/adding_generics-2_4ea/  
sudo cp scripts/javac /usr/local/bin/tigerc  
sudo cp scripts/java /usr/local/bin/tiger
```

Nach Eingabe Ihres Administrator-Passworts werden die beiden Dateien kopiert. Wenn Sie nun das Terminal-Fenster schließen und ein neues öffnen, können Sie die Kommandozeilenbefehle `tigerc` und `tiger` aufrufen!

Einmalig initialisieren

Zuvor müssen Sie aber das `bootstrap`-Skript noch einmalig ausführen, damit ein neues JAR-Archiv für die Laufzeitumgebung generiert wird:

```
cd ~/adding_generics-2_4ea/  
sh scripts/bootstrap
```

Im Verzeichnis `adding_generics-2_4ea` sollte sich nun die Datei `newrt.jar` befinden. Der neue Compiler und die neue Laufzeitumgebung sind einsatzbereit!

Übersetzen und ausführen

Zum Testen übersetzen Sie einfach das beiliegende Beispiel und führen es aus:

```
cd ~/adding_generics-2_4ea/examples/  
tigerc Test.java  
tiger Test
```

Wenn keine Fehler auftreten, sollten Sie dann folgende Ausgabe sehen:

```
zero  
one  
Addition: 1 plus 1 equals 2  
sin(PI/12) = 0.25881904510252074  
Colors are RED GREEN and BLUE
```

```
found RED.  
found GREEN.  
found BLUE.
```

Was hier vielleicht nicht besonders spektakulär aussieht, wird durch neue Sprachkonstrukte erreicht, die die Programmierung zum Teil deutlich vereinfachen und auf die viele Entwickler dementsprechend lange gewartet haben. Unter anderem gehören dazu

► Generische Typen (»Generics«)

Während die dynamischen Datenstrukturen des Java-Collection-Frameworks bisher immer mit `Object`-Referenzen gearbeitet haben, können Sie nun beim Erzeugen einer solchen Datenstruktur festlegen, von welchem Typ die Referenzen sein müssen. Versuchen Sie dann, einen anderen Objekttyp in die Struktur einzufügen, bekommen Sie einen Compilerfehler:

```
LinkedList<String> strliste = new LinkedList<String>();  
strliste.add("eins");  
strliste.add("zwei");  
String str = strliste.get(0);
```

Anstelle von bisher nur `LinkedList` geben Sie also zusätzlich den gewünschten Objekttyp in spitzen Klammern an, hier `LinkedList<String>`. Beachten Sie, dass in der letzten Zeile der Typecast fehlt! Wo Sie bisher `String str = (String)strlist.get(0)` schreiben mussten, haben die diversen Auslesemethoden nun den passenden Rückgabetyt.

► Erweiterte for-Schleife (»foreach«)

Mit einer `for`-Schleife können Sie nun sehr einfach durch alle Elemente einer dynamischen Datenstruktur des Java-Collection-Frameworks oder eines Arrays iterieren. Dazu geben Sie im Schleifenkopf einfach eine Variable (vom passenden Basisdatentyp), gefolgt von einem Doppelpunkt und der Referenz auf die Datenstruktur an:

```
for (String s : strliste) {  
    System.out.println(s);  
}
```

► Automatische Hüllobjekte

Bisher war es recht mühsam, primitive Datentypen in Objekt-Kollektionen einzufügen. Sie mussten – beispielsweise mit `new Integer(42)` – erst ein Hüllobjekt erzeugen und konnten dies dann einfügen. Java 1.5 verpackt primitive Typen bei Bedarf nun automatisch in ihre jeweiligen Hüllobjekte (»Autoboxing«):

```
LinkedList<Integer> intliste = new LinkedList<Integer>();
intliste.add(42);
int i = intliste.get(0);
```

In der letzten Zeile sehen Sie außerdem, dass verpackte Objekte automatisch ausgepackt werden (»Autounboxing«). Bisher hätten Sie hier `int i = ((Integer)intliste.get(0)).intValue()` schreiben müssen.

► Aufzählungen (Enumerationen)

Java 1.5 kennt nun auch endlich Aufzählungen, wie sie schon aus C++ und anderen Sprachen bekannt sind:

```
enum Farbe { ROT, BLAU, WEISS };
Farbe autofarbe = Farbe.BLAU;
```

Ein Enumerationstyp kann auch bei `switch` verwendet werden. Dabei brauchen Sie bei `case` die Enumerationsklasse nicht angeben, Sie schreiben also einfach `BLAU` anstatt `Farbe.BLAU`:

```
switch (autofarbe) {
case BLAU:
    break;
case ROT:
    break;
}
```

► Statische Imports

```
import static java.lang.Math.*;
```

Mit dem neuen `import static`-Befehl können Sie alle (oder bestimmte) statischen Elemente einer Klasse so importieren, dass Sie im Quelltext ohne den Klassennamen darauf zugreifen können:

```
double eins = sin(PI/2.0);
```

► Variable Argumentlisten

Bisher war es nicht möglich, Argumentlisten mit variabler Länge in Java zu programmieren. Nun können Sie am Ende einer Argumentliste einen Objekttyp gefolgt von drei Punkten `...` deklarieren. Die übergebenen Parameter werden dann in den Objekttyp konvertiert und als ganz normales Array übergeben, so dass Sie mit `length` die Anzahl der Argumente ermitteln können:

```
public static void vieleArgumente(Object ... args) {
    for (int i = 0; i < args.length; i++) {
        System.out.println( args[i] );
    }
}
```

Natürlich könnten Sie hier bei der `for`-Schleife auch wieder die neue »foreach«-Version einsetzen!

Der Aufruf der Methode könnte wie folgt aussehen. Bei der Methode kommt dann ein Array mit einem `Integer`-, einem `String`- und einem `Double`-Objekt an:

```
vieleArgumente(1,"zwei",3.0);
```

In der J2SE 5.0-API findet die variable Argumentliste übrigens bei `System.out.printf()` Verwendung – damit können Sie Textausgaben von C- und C++-Programmen deutlich einfacher nach Java übernehmen.

Auf der CD finden Sie für diese neuen Syntaxelemente im Verzeichnis `examples/ch16/test15/` ein kurzes Beispiel `Test15.java`. Denken Sie aber immer daran, dass es sich bei der »Early Access«-Version noch nicht um das vollständige J2SE 5.0 handelt.

Literatur & Links

- ▶ F. Esser, »Das Tiger-Release: Java 5 im Einsatz«, Galileo Computing 2004
Dieses Buch verzichtet auf Grundlagenwissen und stellt Ihnen die Neuerungen in Java (1.5) kompakt und praxisnah dar.
- ▶ <http://www.galileocomputing.de/artikel/gp/artikelID-152>
Der Autor von »Java ist auch eine Insel« gibt Ihnen einen kleinen Einblick in die wichtigsten Neuerungen von Java 1.5.
- ▶ <http://java.sun.com/developer/technicalArticles/releases/j2se15/>
Auch von Sun gibt es eine kurze Übersicht über Java 1.5.

D System-Properties

*»Mögen wir noch so viele gute Eigenschaften haben,
aber die Welt achtet vor allem auf unsere schlechten.«
(Jean Baptiste Molière)*

System-Properties (»Eigenschaften«, Singular: »Property«) betreffen den Zustand der Java-Laufzeitumgebung, der ausgelesen und unabhängig vom Programmcode beeinflusst werden kann. Die vordefinierten System-Properties werden dabei meistens nur gelesen und normalerweise nicht verändert. Die konfigurierbaren System-Properties werden dagegen meistens nur geschrieben und nur selten ausgelesen.

D.1 Vordefinierte System-Properties

Mit den vordefinierten System-Properties können Sie bestimmte Eigenschaften des Systems und der Java-Implementierung abfragen. Eine Liste aller vordefinierten Properties erhalten Sie mit folgender Anweisung:

```
//CD/examples/ch17/PropsTest.java
System.getProperties().list(System.out);
```

Bei der Abfrage einzelner Properties erhalten Sie als Ergebnis immer eine Zeichenkette (oder die null-Referenz), und nach Möglichkeit sollten Sie zur Auswertung auch Zeichenkettenvergleiche einsetzen:

```
if (System.getProperty("os.version").startsWith("10.3"))
    { /* ... */ }
```

Die folgende Tabelle listet die wichtigsten vordefinierten Properties auf, die MacOS X und Apples Java-Implementierung betreffen. Die Werte stammen von MacOS X 10.3.5 mit dem Java 1.4.2 Update 1 – bei anderen Versionen werden Sie in den Properties entsprechend andere Versionsnummern vorfinden.

Property	Wert	Bemerkung
java.version	1.4.2_05	Liefert die Nummer der von Apple implementierten Sun-Java-Version (siehe Anhang G).
java.vm.version	1.4.2-38	

Tabelle D.1 Eine Auswahl der vordefinierten System-Properties

Property	Wert	Bemerkung
java.runtime.version	1.4.2_05-141	Liefert die Versionsnummer von Apples Java-Implementierung (siehe Anhang G).
mrj.version	141	Lieferte früher die Version von Apples Java-Implementierung. Sollte heutzutage (Mac OS X 10.2 und neuer) nicht mehr verwendet werden (siehe Anhang G).
java.vendor	Apple Computer, Inc.	
java.vendor.url	http://apple.com/	
java.vendor.url.bug	http://developer.apple.com/java/ (obwohl Fehler besser mit dem Bug Reporter auf der Seite https://bugreport.apple.com/ gemeldet werden sollten)	
java.vm.vendor	»Apple Computer, Inc.«	
os.name	MacOSX	Indem Sie diese Property abfragen, können Sie portabel feststellen, ob Ihre Applikation auf Mac OS X läuft (siehe Kapitel 1 und Anhang G).
os.version	10.3.5	Entspricht der Mac OS X-Version aus »Über diesen Mac« im Apfel-Menü, d.h., bei Mac OS X 10.3 steht hier auch nur 10.3 drin.
os.arch	ppc	Mac OS X läuft auf PowerPC-Prozessoren, abgekürzt PPC. Apple bezeichnet diese Prozessoren je nach Leistung und Fähigkeiten mit G3, G4 und G5.
sun.cpu.endian	big	Apple verwendet seit jeher – auch schon bei den 68k-Rechnern – »Big Endian«-Prozessoren, bei denen die Bytes eines Speicherwortes in der »richtigen« (natürlichen) Reihenfolge abgelegt sind.
line.separator	\n	Ein Zeilenumbruch wird bei Mac OS X Unix-typisch mit einem Newline-Zeichen (ASCII 10) erzeugt. Bis Mac OS 9 hat Apple hierfür \r (Carriage Return, ASCII 13) verwendet (siehe Kapitel 5).
path.separator	:	Mehrere Pfadangaben werden mit einem Doppelpunkt getrennt, beispielsweise beim Setzen des Klassenpfades (siehe Kapitel 5).

Tabelle D.1 Eine Auswahl der vordefinierten System-Properties (Forts.)

Property	Wert	Bemerkung
<code>file.separator</code>	/	Innerhalb einer Pfadangabe werden Verzeichnisse und Unterverzeichnisse mit dem Schrägstrich (Slash) getrennt. Mac OS 9 setzt dazu den Doppelpunkt ein, Windows den umgekehrten Schrägstrich (Backslash) (siehe Kapitel 5).
<code>file.encoding</code>	MacRoman	Mac OS X kommt zwar problemlos mit Texten in den diversen Unicode-Kodierungen zurecht, aber bei 8-Bit-Texten wird in Westeuropa und Amerika nach wie vor die vom klassischen Mac OS stammende MacRoman-Kodierung eingesetzt (siehe Kapitel 5).

Tabelle D.1 Eine Auswahl der vordefinierten System-Properties (Forts.)

D.2 Konfigurierbare System-Properties

Mit den folgenden Properties können Sie das Laufzeitverhalten Ihrer Applikation beeinflussen. Meistens wird ein boolescher Wert (`true`, `false`) zur Konfiguration verwendet, zum Teil auch beliebige Zeichenketten oder Zahlen. Zum Setzen einer Property haben Sie mehrere Möglichkeiten. Zum einen können Sie die Property in der Kommandozeile setzen:

```
java -Dapple.laf.useScreenMenuBar=true MeineApplikation
```

Dabei kann die VM-Option `-D` beliebig häufig angegeben werden. Zum anderen können Sie das Setzen direkt im Quelltext durchführen:

```
System.setProperty( "apple.laf.useScreenMenuBar", "true" );
```

Und schließlich können Sie die Properties auch in die Datei `Info.plist` im jeweiligen Programmpaket eintragen, und zwar im `Properties-Dictionary` innerhalb des `Java-Dictionary`:

```
<key>Java</key>
<dict>
  <!-- ... -->
  <key>Properties</key>
  <dict>
    <key>apple.laf.useScreenMenuBar</key>
    <string>true</string>
  </dict>
</dict>
```

Gerade weil es sich hier um Apple-spezifische Properties handelt, die auf anderen Systemen keinen Effekt haben und dort auch gar nicht gesetzt werden müssten, ist diese letzte Möglichkeit: das Apple-spezifische MacOS X-Programmpaket – sehr elegant.

Die folgenden Listen berücksichtigen Properties bis Java 1.4.2. Manche Properties stehen dabei nur für bestimmte Java-Versionen zur Verfügung, insbesondere wenn eine aktuelle Java-Version neue Property-Namen einführt, um alte Properties zu ersetzen. Sofern bei den Property-Beschreibungen nichts anderes angegeben ist, schadet es dann nicht, wenn Sie *beide* Properties angeben, um z.B. Java 1.3 und 1.4 innerhalb desselben Programmpaketes unterstützen zu können.

Konfiguration der Laufzeitumgebung

Wenn die Java-Laufzeitumgebung gestartet wird, müssen diverse Parameter bekannt sein, unter anderem die Hauptklasse und der Klassenpfad. Beim Aufruf von `java` geben Sie dies mit Umgebungsvariablen oder direkt in der Kommandozeile an. Bei MacOS X-Programmpaketen wird dies dagegen in speziellen Konfigurationsdateien festgelegt. MacOS X 10.0 verwendete dafür die folgenden Properties in der Datei `MRJApp.properties`, seit MacOS X 10.1 empfiehlt Apple die Verwendung der Datei `Info.plist` und der entsprechenden Schlüssel (siehe Kapitel 4, *Ausführbare Programme*).

- ▶ `com.apple.mrj.application.main`
Gibt den vollqualifizierten Namen der Hauptklasse an, also der Klasse mit der `main()`-Methode. Ist zwingend, wenn die Laufzeitumgebung mit diesen Properties konfiguriert wird. Veraltet, besser den `MainClass`-Schlüssel in `Info.plist` einsetzen!
- ▶ `com.apple.mrj.application.classpath`
Gibt den Klassenpfad an, d.h. eine Liste aller von der Anwendung benötigten JAR-Archive und Klassenverzeichnisse (durch Doppelpunkte getrennt). Ist zwingend, wenn die Laufzeitumgebung mit diesen Properties konfiguriert wird. Veraltet, besser den `ClassPath`-Schlüssel in `Info.plist` einsetzen!
- ▶ `com.apple.mrj.application.parameters`
Hiermit können Sie eine (mit Leerzeichen getrennte) Liste von Parametern angeben, die als String-Array an `main()` übergeben werden. Optional. Veraltet, besser den `Arguments`-Schlüssel in `Info.plist` einsetzen!
- ▶ `com.apple.mrj.application.workingdirectory`
Legt das Arbeitsverzeichnis der Applikation fest. Optional. Veraltet, besser den `WorkingDirectory`-Schlüssel in `Info.plist` einsetzen!

- ▶ `com.apple.mrj.application.vm.options`
Erlaubt das Setzen von VM-Optionen (siehe Anhang E, *VM-Optionen*). Wenn Sie mehr als eine Option angeben, werden diese durch Leerzeichen getrennt. Optional. Veraltet, besser den `VMOptions`-Schlüssel in `Info.plist` einsetzen!
- ▶ `com.apple.mrj.application.JVMVersion`
Legt die Java-Version fest, mit der die Applikation ausgeführt wird, z.B. »1.3*« (irgendeine 1.3er-Version) oder »1.4+« (Java 1.4 oder neuer). Wenn Sie nichts angeben, wird Ihr Programm mit Java 1.3.1 gestartet, aber verlassen Sie sich nicht darauf: Eine neue Mac OS X-Version könnte in diesem Fall eine andere Version verwenden. Ist erst mit dem Java 1.4.1-Update verfügbar, damit Sie alte Java-Anwendungen, die noch `MRJApp.properties` nutzen, einfach anpassen können. Setzen Sie bei aktuellen Applikationen stattdessen den `JVMVersion`-Schlüssel in `Info.plist` ein!

Anpassung an die Benutzungsoberfläche

Damit Java-Anwendungen wie »echte« Mac-Applikationen aussehen, sollten Sie die Benutzungsoberfläche (GUI) mit den folgenden Properties optimieren. Damit die Einstellungen korrekt übernommen werden, sollten Sie die Werte so früh wie möglich setzen – am besten am Anfang der `main()`-Methode oder sogar außerhalb des Programmcodes in der Datei `Info.plist`. Die genaue Verwendung der Properties innerhalb des Quelltextes ist in Kapitel 3, *Grafische Benutzungsoberflächen* ausführlich beschrieben.

- ▶ `apple.laf.useScreenMenuBar`
Setzen Sie diese Property auf `true`, damit die Swing-Menüleiste Mac-typisch oben am Bildschirmrand (und nicht innerhalb eines Fensters) angezeigt wird. Bei modalen `JDialogen` verbleibt die Menüleiste allerdings innerhalb des Dialogs. Nur beim Aqua-Look & Feel aktiv. Java 1.4; für Java 1.3 muss stattdessen die Property `com.apple.macos.useScreenMenuBar` verwendet werden.
- ▶ `apple.awt.showGrowBox`
Bei Java 1.4 zeigen Fenster, deren Größe verändert werden kann, dies *nicht* standardmäßig mit einem Größenänderungselement unten rechts im Fenster an, obwohl der Anwender dort mit dem Mauszeiger greifen und ziehen kann. Setzen Sie diese Property auf `true`, damit dem Benutzer diese Funktionalität auch grafisch angezeigt wird – dann sollten Sie in Ihrem Dialoglayout aber auch entsprechend Platz für das Größenänderungselement vorsehen, damit keine anderen Elemente verdeckt werden. Für Java 1.3 bietet `com.apple.mrj.application.growbox.intrudes` eine ähnliche Funktionalität.

► `apple.awt.brushMetalLook`

Wenn Sie diese Property auf `true` setzen, werden Fenster und Dialoge nicht im Aqua-Stil (mit einem Streifenmuster-Hintergrund), sondern im »Brush Metal Look« (gebürstetes Metall, wie bei Safari oder iTunes) dargestellt. Da Apple dieses Aussehen eigentlich nur für das Hauptfenster einer Applikation vorgesehen hat, das Aussehen aber leider nicht pro Fenster eingestellt werden kann, sollten Sie diese Property nur dann aktivieren, wenn Ihre Anwendung im Wesentlichen aus nur einem Fenster besteht. Ab Java 1.4.

Diese Property ist eigentlich nur für AWT-Applikationen gedacht, aber mit ein paar Tricks können Sie den »Brush Metal Look« auch bei Swing-Anwendungen verwenden – allerdings war die Darstellung unter Mac OS X 10.2 noch ziemlich problematisch, erst mit 10.3 hat Apple dies größtenteils korrigiert.

► `apple.awt.fileDialogForDirectories`

Wenn diese Property auf `true` gesetzt ist, können in einem `java.awt.FileDialog` Verzeichnisse anstelle von Dateien ausgewählt werden. Erst ab Java 1.4, bei Java 1.3 muss für dieses Verhalten der Modus des Dateiauswahl-Dialogs auf den (nicht portablen) Wert 3 gesetzt werden.

► `com.apple.mrj.application.apple.menu.about.name`

Mit dieser Property können Sie einen Programmnamen festlegen, der nicht länger als 16 Zeichen sein sollte. Diese Property sollte (wenn überhaupt) nur bei Java 1.3 eingesetzt werden, ab Java 1.4 gilt sie als veraltet! Die deutlich bessere Lösung ist der Schlüssel `CFBundleName` in der Datei `Info.plist` eines Programmpaketes (siehe Kapitel 4, *Ausführbare Programme*).

Bei Mac OS X 10.0 und 10.1 war diese Property allerdings zwingend notwendig, damit im Programm-Menü ein »Über ...«-Eintrag auftaucht, auf den Sie dann mit `com.apple.mrj.MRJAboutHandler` und `registerAboutHandler()` reagieren können. Ab Mac OS X 10.2 ist der »Über ...«-Eintrag auch dann vorhanden, wenn `CFBundleName` in `Info.plist` gesetzt ist.

► `com.apple.mrj.application.growbox.intrudes`

Standardmäßig ist diese Property aktiv und sorgt dafür, dass bei Fenstern, deren Größe änderbar ist, das Größenänderungselement unten rechts in das Fenster hineinragt. Wenn der Fensterinhalt dafür aber nicht ausgelegt ist, könnten dadurch Dialogelemente teilweise verdeckt werden. In diesem Fall setzen Sie diese Property auf `false`, was die Fensterhöhe um 15 Pixel vergrößert und das Größenänderungselement auf einer eigenen »Zeile« im Dialog darstellt. Am besten überarbeiten Sie jedoch das Dialoglayout, um unten rechts etwas Platz zu lassen. Nur Java 1.3, bei Java 1.4 bietet `apple.awt.showGrowBox` eine ähnliche Funktionalität.

- ▶ `com.apple.mrj.application.live-resize`
Mit dieser Property können Sie eine mitlaufende Darstellung des Fensterinhalts bei einer Größenänderung einschalten. Standardmäßig ist aber `false` voreingestellt, da das ständige Neuzeichnen zu Performanzproblemen führen kann. Nur Java 1.3.
- ▶ `com.apple.macos.smallTabs`
Tabs (Karteireiter) werden mit einer kleineren Zeichengröße dargestellt, wenn diese Property auf `true` gesetzt ist. Dadurch belegen die Tabs dann weniger Platz. Nur Java 1.3.
- ▶ `com.apple.macos.useScreenMenuBar`
Setzen Sie diese Property auf `true`, damit die Swing-Menüleiste Mac-typisch oben am Bildschirmrand (und nicht innerhalb eines Fensters) angezeigt wird. Bei modalen `JDialog`en verbleibt die Menüleiste allerdings innerhalb des Dialogs. Nur beim Aqua-Look & Feel aktiv. Nur Java 1.3, ab Java 1.4 sollte die Property `apple.laf.useScreenMenuBar` verwendet werden.
- ▶ `com.apple.macos.use-file-dialog-packages`
Wenn Sie diese Property auf `true` setzen, werden Programm- und Installationspakete (`.app` und `.pkg`) von `java.awt.FileDialog` als Dateien (und nicht als Verzeichnishierarchie) angezeigt – dies ist in den allermeisten Fällen die sinnvollere Variante. Sollten Sie innerhalb einer Applikation unterschiedliches Verhalten benötigen, können Sie die Property bei Bedarf mit `System.setProperty()` zur Laufzeit ändern. Nur Java 1.3, in Java 1.4 gilt diese Property als veraltet. Alternativ kann `javax.swing.JFileChooser` mit speziellen Eigenschaften konfiguriert werden.
- ▶ `com.apple.macos.useMDEF`
Bei den allerersten MacOS X-Versionen konnte man noch festlegen, ob Swing-Menüs beliebigen Inhalt (z.B. Bilder) in der Menüleiste darstellen durften, später war dies dann generell vorgesehen. Diese Property war standardmäßig eingeschaltet. Nur Mac OS X 10.0.
- ▶ `com.apple.backgroundOnly`
Wenn eine Applikation AWT-Komponenten benutzt, aber keine Benutzungsoberfläche anzeigt (z.B. Server-Applikationen zur Bildverarbeitung), können Sie diese Property auf `true` setzen und damit das Dock-Symbol und die Menüleiste unterdrücken. Dieser Workaround steht allerdings nur bei Apples Java 1.3 zur Verfügung, ab Java 1.4 wird das so genannte »Headless AWT« dann (mit einer anderen Property) allgemein unterstützt.

Exklusiver Vollbildmodus

Mit Java 1.4 hat Sun eine API für den so genannten »Full-Screen Exclusive Mode« eingeführt. Damit können Programme exklusiv den gesamten Bildschirm benutzen und das Zeichnen, die Farbtiefe und die Bildschirmauflösung aktiv kontrollieren (im Gegensatz zu »normalen« Anwendungen mit grafischen Benutzungsoberflächen, die nur passiv auf Zeichenanforderungen reagieren). Dafür wird ein `Window` (oder eine Unterklasse davon, wie hier im Beispiel ein `Frame`) mit `setFullScreenWindow()` zu dem Fenster gemacht, das den gesamten Bildschirm bedeckt:

```
import java.awt.*;
// ...
GraphicsEnvironment env;
GraphicsDevice dev;
Frame f;
env = GraphicsEnvironment.getLocalGraphicsEnvironment();
dev = env.getDefaultScreenDevice();
f = new Frame("Vollbild");
f.setResizable(false);
f.setUndecorated(true);
// ...
if ( dev.isFullScreenSupported() ) {
    try {
        dev.setFullScreenWindow( f );
        // "f" arbeitet nun im exklusiven Vollbildmodus ...
    }
    finally {
        dev.setFullScreenWindow( null );
    }
}
```

Zunächst wird die lokale Grafikumgebung ermittelt und dann mit `getDefaultScreenDevice()` der Hauptbildschirm abgefragt (wenn an Ihrem Rechner nur ein Bildschirm angeschlossen ist, ist dies der Hauptbildschirm). Vollbildfenster sollten nicht größenveränderbar sein und keine Umrandung (Titelzeile, Rand) besitzen, daher wird letzteres mit `setUndecorated()` abgeschaltet.

Sicherheitshalber wird dann mit `isFullScreenSupported()` getestet, ob der Vollbildmodus überhaupt unterstützt wird. Wenn nicht, könnte dies nämlich zu Performanzverlusten führen, weil der Modus dann emuliert werden muss. Schließlich wird der Vollbildmodus mit `setFullScreenWindow()` aktiviert

(und durch Übergabe der `null`-Referenz wieder deaktiviert). Das geschieht innerhalb von `try...finally`, damit der Vollbildmodus auf jeden Fall wieder beendet wird.

Ausführliche Hinweise zum exklusiven Vollbildmodus finden Sie bei Sun auf der Seite <http://java.sun.com/docs/books/tutorial/extra/fullscreen/>. Leider ist dieser Modus in Apples Java 1.4.2-Implementierung fehlerhaft (auch wenn dies mit den neuesten Java-Updates besser geworden ist) – manchmal klappt die Initialisierung nicht, und falls doch, werden dann keine Maus-Ereignisse angenommen. Aber auch Suns zugrunde liegende Implementierung hat zum Teil noch Probleme mit Threads, was wohl erst mit Java 1.5 behoben sein wird. Wenn Sie den Vollbildmodus dennoch verwenden wollen, können Sie ihn mit folgenden Properties konfigurieren:

- ▶ `apple.awt.fakefullscreen`
Mit dieser Property können Sie Vollbild-Applikationen zu Testzwecken in ganz gewöhnlichen Fenstern darstellen lassen. Der Standardwert ist `false`.
- ▶ `apple.awt.fullscreencapturealldisplays`
Wenn Sie mehrere Bildschirme an Ihrem Rechner betreiben, werden beim Vollbildmodus normalerweise alle Bildschirme bis auf einen (der das Vollbild darstellt) abgedunkelt. Wenn Sie diese Property auf `false` setzen, bleiben die Inhalte aller Bildschirme sichtbar, was beim Testen hilfreich sein kann.
- ▶ `apple.awt.fullscreenhidecursor`
Versteckt den Mauszeiger im Vollbildmodus. Mit dem Wert `true` ist dies standardmäßig aktiv.
- ▶ `apple.awt.fullscreenusefade`
Wenn Sie den Vollbildmodus aktivieren und dabei eine Änderung der Bildschirmauflösung durchführen, geschieht dies mit einem Aus- und Einblendeffekt. Wenn Sie diese Property auf `true` setzen, wird der Effekt immer eingesetzt, unabhängig von einem Auflösungswechsel.

Ein Problem bei MacOS X besteht darin, dass trotz Vollbildmodus die Menüleiste und das Dock sichtbar sind. Mit einer statischen Methode einer Cocoa-Klasse können Sie beide unsichtbar machen:

```
import com.apple.cocoa.application.NSMenu;
// ...
NSMenu.setMenuBarVisible(false);
```

Denken Sie daran, beide Elemente beim Beenden des Vollbildmodus wieder einzublenden – am besten im `finally`-Block, wo Sie auch `setFullScreenWindow(null)` aufrufen.

Wenn Ihnen der Java-Vollbildmodus noch zu fehleranfällig ist oder Sie auch mit Java 1.3 ein exklusives Vollbild-Fenster benötigen, kann die QTJava-Klasse `quicktime.app.display.FullScreenWindow` eine Alternative sein. Leider gilt diese Klasse seit QuickTime 6.4 (QTJava 6.1) als veraltet.

Fensterpositionierung

Ab Java 1.4 können Sie mit diesen Properties verhindern, dass die Fenster Ihrer Applikation außerhalb des Bildschirms liegen und somit nicht genutzt werden können. Dies kann beispielsweise passieren, wenn die in einer Konfigurationsdatei gespeicherten Fensterkoordinaten außerhalb der aktuellen Bildschirmauflösung liegen.

- ▶ `apple.awt.window.position.forceSafeCreation`
Stellt sicher, dass neue Fenster nicht außerhalb des Bildschirmbereichs geöffnet werden können, wo der Anwender keinen Zugriff auf die Fenster hätte. Mit dem Wert `false` ist diese Property standardmäßig deaktiviert.
- ▶ `apple.awt.window.position.forceSafeProgrammaticPositioning`
Mit dieser Property wird verhindert, dass ein Programm ein Fenster aus dem sichtbaren Bildschirmbereich herauschieben kann. Mit dem Wert `true` ist dies standardmäßig aktiv. Nach Möglichkeit sollten Sie diese Funktion nicht deaktivieren, da ansonsten die Fensterverwaltung des Systems durcheinander kommen könnte.
- ▶ `apple.awt.window.position.forceSafeUserPositioning`
Hiermit wird der Anwender daran gehindert, Fenster aus dem Bildschirmbereich herauszuschieben, weil er dann keinen Zugriff mehr auf die Fenster hätte. Diese Property ist mit dem Wert `false` standardmäßig ausgeschaltet.

Grafikdarstellung

Mit der Java2D-API können Sie bei der Grafikausgabe »Rendering Hints« (Darstellungshinweise) angeben, um die Darstellung zu beeinflussen. Beispielsweise können Sie »Antialiasing« (Weichzeichnen) aktivieren:

```
import java.awt.*;
import java.awt.geom.*;
// ...
public void paint(Graphics g) {
    Graphics2D g2d = (Graphics2D)g;
    g2d.setRenderingHint( RenderingHints.KEY_ANTIALIASING,
                          RenderingHints.VALUE_ANTIALIAS_ON );
    g2d.draw( new Line2D.Double(0,0,100,90) );
}
```


Mit den folgenden Properties können Sie diese Darstellungshinweise für Ihre gesamte Applikation festlegen. Innerhalb des Programms können Sie die Ausgabe dann immer noch ganz speziell nach obigem Beispiel mit den `KEY_XXX`-Konstanten aus der Klasse `java.awt.RenderingHints` konfigurieren.

► `apple.awt.antialiasing`

Schaltet für Grafik-Basiselemente (»Grafik-Primitive« – Linien, Rechtecke usw.) eine weichgezeichnete Ausgabe ein (»Antialiasing«). Die Textausgabe übernimmt diese Einstellung, allerdings können Sie das Text-Weichzeichnen auch unabhängig mit `apple.awt.textantialiasing` konfigurieren. Unabhängig von dieser globalen Einstellung können Sie das Weichzeichnen pro Ausgabeobjekt mit `RenderingHints.KEY_ANTIALIASING` beeinflussen.

Erlaubte Werte sind `on` und `off`. Wenn Ihre Anwendung das Aqua-Look & Feel verwendet, ist das Weichzeichnen standardmäßig eingeschaltet, ansonsten aus. Auch wenn Sie diese Property explizit auf `off` setzen, werden Aqua-Oberflächenelemente nach wie vor weichgezeichnet – die Einstellung betrifft also nur andere Look & Feels und Grafikfunktionen. Ab Java 1.4, bis Java 1.3 muss `com.apple.macosx.AntiAliasedGraphicsOn` verwendet werden.

► `apple.awt.textantialiasing`

Hiermit können Sie festlegen, ob Texte weichgezeichnet werden sollen. Mögliche Werte sind `on` und `off`. Wenn Sie diese Property nicht setzen, wird die Einstellung von `apple.awt.antialiasing` übernommen. Mit `RenderingHints.KEY_TEXT_ANTIALIASING` können Sie diese Einstellung innerhalb des Programms beeinflussen. Ab Java 1.4, bis Java 1.3 muss `com.apple.macosx.AntiAliasedTextOn` verwendet werden.

► `apple.awt.rendering`

Legt fest, ob `Graphics2D`-Ausgabekontexte auf Geschwindigkeit oder auf Qualität optimieren. Die erlaubten Werte sind entsprechend `speed` und `quality`. Sie können alternativ auch den Darstellungshinweis `RenderingHints.KEY_RENDERING` setzen. Ab Java 1.4.

► `apple.awt.interpolation`

Setzt `RenderingHints.KEY_INTERPOLATION`, um den Algorithmus für die Bildumwandlung festzulegen. Erlaubte Werte sind `nearestneighbor`, `bilinear` und `bicubic`.

► `apple.awt.fractionalmetrics`

Setzt `RenderingHints.KEY_FRACTIONALMETRICS`, um bei Font-Berechnungen Fließkomma-Angaben (`on`) anstelle der standardmäßigen Ganzzahl-Angaben (`off`) zu verwenden.

- ▶ `com.apple.macosx.AntiAliasedGraphicsOn`
Konfiguriert das Weichzeichnen von Grafikausgaben, das mit dem Wert `true` standardmäßig aktiv ist. Nur Java 1.3, ab Java 1.4 muss `apple.awt.antialiasing` verwendet werden.
- ▶ `com.apple.macosx.AntiAliasedTextOn`
Bestimmt das Weichzeichnen von Textausgaben, das standardmäßig eingeschaltet (`true`) ist. Nur Java 1.3, ab Java 1.4 muss `apple.awt.textantialiasing` verwendet werden.

Grafikperformanz

- ▶ `apple.awt.graphics.OptimizeShapes`
Normalerweise versucht Mac OS X, nach Möglichkeit einfache Grafikfunktionen anstelle der komplexeren Shape-Objekte zu verwenden. Beispielsweise wird ein Aufruf von `draw(new Rectangle2D.Float(0,0,10,10))` auf `drawRect(0,0,10,10)` abgebildet. Zum Deaktivieren dieser automatischen Optimierung setzen Sie diese Property auf `false`. Ab Java 1.4.
- ▶ `apple.awt.graphics.EnableLazyDrawing`
Grafikausgaben werden normalerweise in eine Warteschlange eingereicht, bevor sie zum Renderer geschickt werden – dadurch kann die Ausgabe optimiert werden. Wenn die Grafikausgaben direkt zum Renderer geschickt werden sollen, setzen Sie diese Property auf `false`. Ab Java 1.4.
- ▶ `apple.awt.graphics.EnableLazyDrawingQueueSize`
Wenn `EnableLazyDrawing` aktiv ist, bestimmt diese Property die Größe der Grafik-Warteschlange. Sie können mit einer Ganzzahl bestimmen, wie viele Grafik-Basiselemente (»Grafik-Primitive« – Linien, Rechtecke usw.) im Zwischenspeicher gehalten werden. Jedes Basiselement benötigt ungefähr zehn Einträge, jeder Eintrag belegt vier Bytes. Der Standardwert ist zwei. Ab Java 1.4.

Grafik-Hardware-Beschleunigung

Wenn es um Grafikdarstellung geht, ist die Java-Implementierung für Mac OS X leider oft langsamer als die entsprechenden Java-Versionen auf Windows oder Linux. Um Java2D-Grafik und Swing-Oberflächen zu beschleunigen, kann eine Hardware-Beschleunigung eingeschaltet werden, wodurch die Grafik dann nicht mehr vom System-Prozessor, sondern von der Grafikkarte berechnet wird.

Allerdings steht dieser Mechanismus **nur für Java 1.3** zur Verfügung. Die Beschleunigung wird dort durch die interne Verwendung von OpenGL erreicht – interessanterweise wird also 2D-Grafik durch eine 3D-API dargestellt.

Dadurch mussten die Apple-Ingenieure aber nicht unnötig viel Aufwand betreiben, denn OpenGL greift bereits automatisch auf die Grafikkarten-Hardware zurück. Java 1.4 baut dagegen auf das System-Framework »CoreGraphics« auf, das derzeit noch nicht Hardware-beschleunigt ist. Sobald CoreGraphics auf QuartzExtreme2D zurückgreift, wird die Grafikausgabe von Java 1.4 dann automatisch mit optimiert sein.

Bei Mac OS X 10.1 war die Java-Grafikbeschleunigung nur testweise eingebaut, daher war sie standardmäßig deaktiviert und musste explizit eingeschaltet werden. Mac OS X 10.2 schaltet die Beschleunigung für die meisten Grafikkarten mit mindestens 16 MByte Videospeicher automatisch ein, Sie müssen in diesem Fall also oft gar nichts tun. Bei Mac OS X 10.3 müssen Sie die Beschleunigung bei Bedarf wieder explizit einschalten.

Zur Konfiguration haben die Java-Versionen von Mac OS X 10.1 leider unterschiedliche System-Properties eingesetzt. Auch Mac OS X 10.2 hat das Konzept noch einmal abgeändert, seitdem ist die Art und Weise der Anwendung gleich geblieben. Den Konfigurationsmöglichkeiten gemeinsam ist aber, dass sie die Beschleunigung nur für bestimmte Grafikkarten aktivieren oder bestimmte Karten davon ausschließen können, denn einige Karten – vor allem die älteren – unterstützen die Hardware-Beschleunigung eher schlecht. Zu diesem Zweck müssen Sie die Namen der Grafikkarten (die »Kennung«) kennen, die Sie in folgender Tabelle finden:

Grafikkarte	Speicher	Kennung
ATI Rage Mobility 128	8 MByte	ATIRage128_8388608
ATI Rage 128	16 MByte	ATIRage128_16777216
ATI Radeon	16 MByte	ATIRadeon_16777216
ATI Rage 128	32 MByte	ATIRage128_33554432
ATI Radeon 7500	32 MByte	ATIRadeon_33554432
ATI Radeon 8500	32 MByte	ATIRadeon8500_67108864
NVidia GeForce2	32 MByte	NVidia11_33554432
NVidia GeForce3	64 MByte	NVidia20_67108864
NVidia GeForce4MX	64 MByte	NVidia20_134217728
NVidia GeForce4TNT	64 MByte	NVidia20_134217728

Tabelle D.2 Liste der für die Hardware-Beschleunigung relevanten Grafikkarten

Die Liste ist auf dem Stand zum Zeitpunkt der Auslieferung von MacOS X 10.2. Mittlerweile gibt es zwar zahlreiche weitere Grafikkarten, die neueren Kennungen sind aber in der Regel unnötig, da Sie diese eigentlich nur brauchen, um zu alte Karten von der Beschleunigung auszuschließen.

Falls Sie dennoch einmal die Kennung Ihrer Grafikkarte benötigen, können Sie zur Ermittlung des Namens das Kommandozeilenprogramm `hwaccel_info_tool` verwenden. Sie finden es in der Apple Developer Connection (ADC) als Archiv »HWAccInfoTool 1.1«. Nach der Installation können Sie sich mit `man` eine Beschreibung anzeigen lassen.

Mit folgenden Properties wird die Grafik-Hardware-Beschleunigung konfiguriert. Für MacOS X 10.2 und neuer sind dabei vor allem `com.apple.hwaccel` und `com.apple.hwaccelexclude` relevant:

- ▶ `com.apple.hwaccel`
Dient zur Aktivierung der Grafikbeschleunigung und steht seit MacOS X 10.1 zur Verfügung; dort war der Wert standardmäßig `false`. Das »Java 1.3.1 Update 1« machte diese Property kurzzeitig überflüssig, da nun ausschließlich `com.apple.hwaccellist` ausgewertet wurde. Seit MacOS X 10.2 wird nun wieder diese Property verwendet, wobei der Standardwert `true` (10.2) auf `false` (10.3) geändert wurde – setzen Sie diese Property also am besten immer explizit auf `true`.

Wenn Sie keine sehr spezielle (nachgerüstete) Grafikkarte einsetzen und MacOS X 10.2 oder 10.3 voraussetzen können, ist die einfachste und beste Möglichkeit zur Aktivierung der Hardware-Beschleunigung das alleinige Setzen dieser Property:

```
java -Dcom.apple.hwaccel=true -jar Applikation.jar
```

- ▶ `com.apple.hwaccellist`
Diese Property ist nur bei MacOS X 10.1 mit dem »Java 1.3.1 Update 1« relevant. Als Wert wurde eine kommasetrennte Liste von Grafikkarten-Kennungen übergeben, bei denen die Hardware-Beschleunigung aktiviert werden sollte. Die Liste möglicher Kennungen stand auch in der Datei `/System/Library/Frameworks/JavaVM.framework/Versions/1.3.1/Home/lib/glconfigurations.properties` zur Verfügung.
- ▶ `com.apple.hwaccelexclude`
Seit MacOS X 10.2 wird die Grafikbeschleunigung grundsätzlich mit `com.apple.hwaccel` aktiviert, wenn die Kennung der Grafikkarte nicht in

der Datei `/System/Library/Frameworks/JavaVM.framework/Versions/1.3.1/Home/lib/hwaccelextclude.properties` eingetragen ist. Sie können diese – meist allerdings vollkommen ausreichende – Liste ignorieren lassen, indem Sie diese Property verwenden und eine Komma-getrennte Liste von Grafikkarten-Kennungen übergeben, die dann nicht beschleunigt werden:

```
java -Dcom.apple.hwaccel=true
      -Dcom.apple.hwaccelextclude=ATI Rage128_8388608
      -jar Applikation.jar
```

Mit diesem Aufruf wird beispielsweise nur die Grafikkarte »ATI Rage Mobility 128« nicht beschleunigt (dies ist normalerweise auch die einzige Karte, die in der Datei `hwaccelextclude.properties` ausgeschlossen wird).

```
java -Dcom.apple.hwaccel=true -Dcom.apple.hwaccelextclude=
      -jar Applikation.jar
```

Hier wird kein Wert nach `hwaccelextclude` angegeben, die Hardware-Beschleunigung wird also bei allen Grafikkarten aktiviert.

► `com.apple.usedebugkeys`

Ab Mac OS X 10.2 können Sie diese Property setzen, um die Grafikbeschleunigung zu Testzwecken per Tastatur ein- und auszuschalten. Mit `⌘+F7` können Sie dann die Beschleunigung zur Laufzeit aktivieren, falls die Grafikkarte nicht von der Beschleunigung ausgeschlossen ist (`⌘+F7` schaltet die Beschleunigung wieder ab). `⌘+F8` erzwingt eine Hardware-Beschleunigung, aber Achtung: Wenn die Grafikkarte dies nicht unterstützt, kann die Anzeige arg durcheinander geraten. `⌘+F8` deaktiviert diesen Modus wieder.

Bei Mac OS X 10.1 mit installiertem »Java 1.3.1 Update 1« waren die Tastenkombinationen übrigens aktiv, obwohl es diese Property dort noch gar nicht gab.

► `com.apple.forcehwaccel`

Seit Mac OS X 10.2 können Sie die Grafik-Hardware-Beschleunigung erzwingen, ohne dass irgendwelche Ausschlusslisten beachtet werden. Setzen Sie dazu einfach diese Property auf `true`.

► `com.apple.hwaccelnogrowbox`

Ebenfalls seit Mac OS X 10.2 steht diese Property zur Verfügung, mit der Sie an jedem Fenster erkennen können, ob die Hardware-Beschleunigung aktiv ist. Normalerweise sehen Sie in einem Fenster unten rechts das Element zur Änderung der Fenstergröße. Wenn Sie diese Property auf `true` setzen, wird dieses Element *nicht* angezeigt, wenn die Beschleunigung aktiv ist.

Grafik-Debugging

Mit den folgenden Properties, die alle erst ab Java 1.4 zur Verfügung stehen, können Sie zu Testzwecken einzelne Grafikfunktionen deaktivieren, um Geschwindigkeitsengpässe besser zu erkennen. Wenn die Grafikausgabe Ihrer Applikation unerwartet langsam ist und Sie beispielsweise vermuten, dass dies vor allem an gefüllten Rechtecken liegt, können Sie der Laufzeitumgebung `apple.awt.graphics.RenderFillRect=false` übergeben und damit gezielt dieses Grafikelement nicht anzeigen lassen. Standardmäßig haben alle Properties den Wert `true`, d.h., es werden alle Grafikfunktionen ausgeführt.

- ▶ `apple.awt.graphics.RenderLine`
Legt fest, ob Linien dargestellt werden.
- ▶ `apple.awt.graphics.RenderDrawRect`
Legt fest, ob Rechtecke dargestellt werden.
- ▶ `apple.awt.graphics.RenderFillRect`
Legt fest, ob gefüllte Rechtecke dargestellt werden.
- ▶ `apple.awt.graphics.RenderDrawRoundRect`
Legt fest, ob abgerundete Rechtecke dargestellt werden.
- ▶ `apple.awt.graphics.RenderFillRoundRect`
Legt fest, ob gefüllte, abgerundete Rechtecke dargestellt werden.
- ▶ `apple.awt.graphics.RenderDrawOval`
Legt fest, ob Ellipsen dargestellt werden.
- ▶ `apple.awt.graphics.RenderFillOval`
Legt fest, ob gefüllte Ellipsen dargestellt werden.
- ▶ `apple.awt.graphics.RenderDrawArc`
Legt fest, ob Ellipsenbögen dargestellt werden.
- ▶ `apple.awt.graphics.RenderFillArc`
Legt fest, ob gefüllte Ellipsenausschnitte dargestellt werden.
- ▶ `apple.awt.graphics.RenderDrawPolygon`
Legt fest, ob Polygonzüge dargestellt werden.
- ▶ `apple.awt.graphics.RenderFillPolygon`
Legt fest, ob gefüllte Polygone dargestellt werden.
- ▶ `apple.awt.graphics.RenderDrawShape`
Legt fest, ob Shapes dargestellt werden.
- ▶ `apple.awt.graphics.RenderFillShape`
Legt fest, ob gefüllte Shapes dargestellt werden.
- ▶ `apple.awt.graphics.RenderImage`
Legt fest, ob Bilder angezeigt werden.

- ▶ `apple.awt.graphics.RenderString`
Legt fest, ob Zeichenketten dargestellt werden.
- ▶ `apple.awt.graphics.RenderGlyphs`
Legt fest, ob Glyphen dargestellt werden.
- ▶ `apple.awt.graphics.RenderUnicode`
Legt fest, ob Unicode-Zeichen dargestellt werden.

Bilddaten und Farbverwaltung

- ▶ `apple.awt.graphics.EnableLazyPixelConversion`
Apples Java 1.4-Implementation optimiert normalerweise die Umwandlung von Bilddaten, die nicht in einem vom System (bzw. vom CoreGraphics-Framework) unterstützten Format vorliegen. Wenn Sie diese Property auf `false` setzen, wird stattdessen eine genauere, aber langsamere Umwandlung durchgeführt. Die Umwandlung betrifft folgende `java.awt.image.BufferedImage`-Typen:
 - ▶ `TYPE_3BYTE_BGR`
 - ▶ `TYPE_4BYTE_ABGR`
 - ▶ `TYPE_4BYTE_ABGR_PRE`
 - ▶ `TYPE_BYTE_BINARY`
 - ▶ `TYPE_BYTE_INDEXED`
 - ▶ `TYPE_CUSTOM`
 - ▶ `TYPE_INT_ARGB`
 - ▶ `TYPE_INT_BGR`
 - ▶ `TYPE_USHORT_565_RGB`
 - ▶ `TYPE_USHORT_GRAY`
- ▶ `apple.awt.graphics.UseTwoImageLazyPixelConversion`
Wenn das Quellbild den Typ `BufferedImage.TYPE_INT_ARGB` hat, wird zur Optimierung je nach Bedarf entweder mit Java- oder mit System-Pixeln gerechnet. Diese Property hat standardmäßig den Wert `true`, was Sie normalerweise nicht ändern sollten. Ab Java 1.4.
- ▶ `apple.cmm.usecolorsync`
In Java 1.4 werden Farbraumberechnungen mit `java.awt.color.ICC_ColorSpace` normalerweise mit dem KodakCMM-Code durchgeführt. Ab Java 1.4.2 können Sie stattdessen die normalen Mac OS X-ColorSync-Profile verwenden, die der Anwender im System eingestellt hat. Setzen Sie dazu diese Property auf `true`.

E VM-Optionen

»Entdecke die Möglichkeiten!«

»Nichts ist unmöglich!«

(Aus der Werbung)

Beim Aufruf der Java Virtual Machine (JVM) können diverse Optionen für die Java-Laufzeitumgebung angegeben werden – beispielsweise im Terminal direkt nach dem `java`-Kommando im Format

```
java -Option1 -Option2 ... Klassenname Argument1 Argument2 ...
```

Alternativ tragen Sie die Optionen in einem Programmpaket in der Datei `Info.plist` im Java-Dictionary unter dem Schlüssel `VMOptions` ein. Einige der Standardoptionen, z.B. zum Setzen des Klassenpfades oder zum Aufruf eines JAR-Archivs, haben Sie in diesem Buch bereits kennen gelernt. Es gibt aber auch zahlreiche Nichtstandardoptionen, die mit dem Präfix `-X` oder `-XX` beginnen und die entweder ganz Apple-spezifisch sind oder zumindest Mac-Besonderheiten aufweisen. Der wichtige Unterschied zwischen den Standard- und den Nichtstandardoptionen ist, dass sich die Nichtstandardoptionen jederzeit ändern können – seien Sie also vorsichtig, dass die Ausführung Ihres Programms nicht zu sehr von letzteren abhängt. Die meisten Optionen werden Sie vermutlich niemals benötigen, aber wenn Sie ausführliche Debug-Informationen ausgeben oder ganz spezielle Performanzverbesserungen einstellen wollen, finden Sie im Folgenden Hilfe.

Auch beim Compiler `javac` können Optionen angegeben werden, allerdings sind diese nicht systemabhängig und werden daher hier nicht beschrieben. Eine Übersicht erhalten Sie, wenn Sie den Compiler ohne weitere Parameter aufrufen.

Die folgenden Listen berücksichtigen die VM-Optionen bis Java 1.4.2. Die Apple-spezifischen Optionen und MacOS X-Besonderheiten der Nichtstandardoptionen finden Sie aktuell auf der Seite <http://developer.apple.com/documentation/Java/Reference/Java14VMOptions/>. Die Standardoptionen sind für Suns Betriebssystem Solaris auf der Seite <http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/java.html> beschrieben.

Wenn Sie bei den Optionen eine Speicherangabe in Bytes finden, können Sie diese, sofern das nicht explizit ausgeschlossen ist, auch in Kilobyte (mit kleinem oder großem K nach der Zahl) oder Megabyte (kleines oder großes M)

angeben. Die folgenden Angaben sind also gleichbedeutend: 4194304, 4096k, 4096K, 4m und 4M.

Gerade die Optionen für die Speicherverwaltung setzen zum Teil voraus, dass Sie sich mit der internen Funktionsweise einer Garbage Collection (GC) auskennen. Falls Sie sich wirklich dafür interessieren, finden Sie bei Sun auf der Seite <http://java.sun.com/products/hotspot/> Informationen über den Aufbau der Virtual Machine; insbesondere erläutert das Dokument <http://java.sun.com/docs/hotspot/gc1.4.2/> die Möglichkeiten zur Beeinflussung der Garbage Collection. Details über Apples »Generational Garbage Collection« mit der »Eden«-Generation und der »Java Shared Archive« (JSA) Technologie stehen auf http://developer.apple.com/documentation/Java/Conceptual/Java141Development/Virtual_Machine/chapter_7_section_3.html zur Verfügung.

Die erweiterten Nichtstandardoptionen, die mit `-XX` beginnen, können Sie auch so setzen, dass diese für *alle* Java-Aufrufe gelten und nicht jedes Mal neu übergeben werden müssen. Dazu legen Sie die Datei `~/.hotspotrc` (also in Ihrem Benutzerverzeichnis) an und tragen dort alle gewünschten Optionen, eine pro Zeile, ohne das `-XX` ein. Beispielsweise muss für die Option `-XX:+PrintSharedSpaces` die Zeile `+PrintSharedSpaces` auftauchen.

Allgemeine Optionen

► "- ?"

`-help`

Zeigt Informationen zum Aufruf der Laufzeitumgebung sowie die wichtigsten Optionen an. Diesen Text sehen Sie auch, wenn Sie `java` ganz ohne Parameter aufrufen.

► `-client`

Wählt die Java HotSpot Client VM aus – dies ist der Standardfall, daher wird die Option üblicherweise weggelassen. Die Optionen `-jvm` und `-hotspot` haben die gleiche Wirkung, allerdings sind sie ab Java 1.4 veraltet und sollten nicht mehr genutzt werden.

► `-server`

Wählt die Java HotSpot Server VM zur Programmausführung aus. Mac OS X kennt allerdings keine separate Server-VM, es wird stattdessen die Client-VM mit speziellen Anpassungen gestartet:

- Es werden andere Klassen für die Shared-Archiv-Generation verwendet (insbesondere keine GUI-Klassen).

- ▶ Der Java-Heap wird vergrößert.
- ▶ Der Speicher für die Eden-Generation wird vergrößert.
- ▶ Es wird eine Thread-lokale Eden-Garbage Collection verwendet.
- ▶ Es erfolgt ein verstärktes Inlining von Methoden.

Die aggressiveren Optimierungen anderer Server-VMs, beispielsweise das Abschalten der Array-Index- und der `null`-Referenz-Prüfung, sind nicht implementiert.

- ▶ `-classpath Pfade`
`-cp Pfade`

Gibt eine Liste von durch Doppelpunkte (:) getrennten Verzeichnissen, JAR-Archiven und ZIP-Archiven an, in denen nach Klassen gesucht wird. Wenn Sie diese Option setzen, wird die Umgebungsvariable `CLASSPATH` ignoriert. Ist weder diese Option angegeben noch `CLASSPATH` gesetzt, wird das aktuelle Arbeitsverzeichnis (.) als Klassenpfad verwendet (siehe Kapitel 4, *Ausführbare Programme*).

- ▶ `-jar JAR-Archiv`

Führt ein JAR-Archiv aus, das ein Manifest mit einem `Main-Class`-Attribut besitzt. Der Klassenpfad ist dann automatisch auf dieses JAR-Archiv gesetzt, eine Klasse zum Starten wird nicht angegeben (siehe Kapitel 4).

- ▶ `-DSystemproperty=Wert`

Setzt eine System-Property auf den angegebenen Wert. Die `-D`-Option kann beliebig häufig bei einem Aufruf verwendet werden (siehe Kapitel 4).

- ▶ `-verbose`

`-verbose:class`

Gibt Informationen darüber aus, welche Archive und Klassen geladen werden.

- ▶ `-verbose:gc`

Zeigt Garbage Collection-Ereignisse an.

- ▶ `-verbose:jni`

Gibt Informationen über die Verwendung von nicht in Java programmierten (nativen) Methoden aus.

- ▶ `-version`

Zeigt Java-Versionsinformationen an und beendet danach die Programmausführung.

- ▶ `-showversion`

Zeigt Java-Versionsinformationen an und setzt die Programmausführung fort.

- ▶ `-fullversion`
Gibt die komplette Java-Versionsnummer aus und beendet danach die Programmausführung.
- ▶ `-X`
Zeigt eine Liste ausgewählter Nichtstandardoptionen an.
- ▶ `-Xbootclasspath:Pfade`
Hiermit können Sie eine durch Doppelpunkt getrennte Liste von Verzeichnissen, JAR-Archiven und ZIP-Archiven angeben, die anstelle der Standard-J2SE-Klassenbibliothek verwendet werden. Achtung, Sie verlieren hierdurch die J2SE-Kompatibilität (und verstoßen gegen Suns Lizenzbestimmungen, wenn Sie diese Option bei ausgelieferten Applikationen einsetzen)! Bei J2ME hingegen muss `bootclasspath` verwendet werden, allerdings beim Compiler und nicht bei der Laufzeitumgebung.
- ▶ `-Xbootclasspath/a:Pfade`
Hängt eine Liste von Verzeichnissen, ZIP- und JAR-Archiven an den Boot-Klassenpfad an. Verwenden Sie nach Möglichkeit besser `classpath` oder die Erweiterungsverzeichnisse.
- ▶ `-Xbootclasspath/p:Pfade`
Hängt eine Liste von Verzeichnissen, JAR-Archiven und ZIP-Archiven vor den bestehenden Boot-Klassenpfad. Wie bei `Xbootclasspath` verlieren Sie aber auch hiermit die J2SE-Kompatibilität und verstoßen gegen Suns Lizenzbestimmungen, wenn Sie damit Teile der Standard-Klassenbibliothek ersetzen.

Ab Java 1.4 steht ein neuer Mechanismus zur Verfügung, die so genannten »Endorsed Directories« (Verzeichnisse für Nachträge bzw. Zusätze). In diese Verzeichnisse können Sie JAR-Archive legen und damit bestimmte Teile der Standard-Klassenbibliothek durch neuere Versionen ersetzen, und zwar sich gerade entwickelnde Standards – beispielsweise CORBA, DOM und SAX. Welche Pakete genau ersetzt werden dürfen, hat Sun auf der Seite <http://java.sun.com/j2se/1.4.2/docs/guide/standards/> dokumentiert. Wo die entsprechenden JAR-Archive gesucht werden, können Sie mit der System-Property `java.endorsed.dirs` (und der Option `-D`) festlegen. Wenn Sie diese Property nicht setzen, wird bei Mac OS X im Verzeichnis `/Library/Java/Home/lib/endorsed/` gesucht.

- ▶ `-Xfuture`
Führt strengere Prüfungen bei den Klassendateien durch als eine Java 1.1 VM. Nach Möglichkeit sollten Sie diese Option immer verwenden.

- ▶ `-Xinternalversion`
Zeigt Build-Informationen über das Programm `java` an, bei Java 1.4.2 beispielsweise »Built on Jan 9 2004 23:07:30 by root with gcc 3.3 20030304 (build 1495)«.
- ▶ `-Xprof`
Führt ein Profiling der laufenden Applikation durch und zeigt die gemessenen Zeiten in der Standardausgabe an.
- ▶ `-Xrunhprof`
`-Xrunhprof:Unteroptionen`
Aktiviert ein ausführlicheres Profiling für den Speicherverbrauch, CPU-Nutzung und Threads. Was genau gemessen werden soll, können Sie in einer durch Komma getrennten Liste nach dieser Option angeben. Welche Möglichkeiten Sie dabei haben, können Sie sich mit der Option `Xrunhprof:help` anzeigen lassen. Wenn Sie nichts Spezielles konfigurieren, landen die Ausgaben in der Datei `java.hprof.txt` (die ziemlich schnell sehr groß wird).
- ▶ `-XrunBibliothek`
Startet vor dem Ausführen des Java-Codes die angegebene JNI-Bibliothek, die sich im Verzeichnis `/System/Library/Frameworks/JavaVM.framework/Libraries/` oder (ab Java 1.4.2 Update 1) in einem der Erweiterungsverzeichnisse befinden muss. Mit `-XrunShark` würde also `libShark.jnilib` aufgerufen werden.
- ▶ `-Xdebug`
Startet die Anwendung im Debug-Modus. Damit können Sie dann jederzeit den Java-Debugger `jdb` aufrufen und mit der JDB-Option `-attach` mit der laufenden Applikation verknüpfen.
- ▶ `-Xcheck:jni`
Führt eine aufwändigere Prüfung beim Aufruf von JNI-Routinen durch. Wenn ein Fehler bei den JNI-Parametern entdeckt wird, bricht die Virtual Machine die Programmausführung ab.
- ▶ `-XX:+JavaMonitorsInStackTrace`
Gibt bei einem Stack-Trace auch Informationen über den Zustand der Java-Monitore (d.h. über den Zustand der Thread-Synchronisation) aus.
- ▶ `-XX:+MaxFDLimit`
Erhöht die Zahl der erlaubten Datei-Deskriptoren auf das Maximum. Ab Java 1.4.2.
- ▶ `-XX:-PrintJavaStackAtFatalState`
Bei einem Absturz im nativen Code wird versucht, den Java-Stack zu ermitteln und auszugeben. Da dies aber zu weiteren Abstürzen in der Java-Fehler-

behandlung führen kann, können Sie die Stack-Ermittlung mit dieser Option ausschalten. Bei Java 1.3 ist die Option übrigens standardmäßig deaktiviert und muss mit `XX:+PrintJavaStackAtFatalState` explizit eingeschaltet werden.

- ▶ `-XX:+ReduceSignalUsage`
`-Xrs`

Damit die Java-Laufzeitumgebung auch bei abruptem Programmende (beispielsweise, wenn Sie in der zugehörigen Shell `Ctrl+C` drücken) Aufräumcode in so genannten »Shutdown Hooks« ausführen kann¹, reagiert die Virtual Machine auf die System-Signale `SIGHUP`, `SIGINT`, `SIGTERM` und `SIGQUIT`. Wenn Sie eine native Applikation schreiben, die Java über JNI einbindet und selbst auf die Signale reagieren muss, können Sie mit dieser Option die Installation der Java-Signal-Behandlungsroutinen verhindern. Allerdings ist dann auch wirklich Ihre Applikation für das korrekte Herunterfahren der Java-Laufzeitumgebung verantwortlich – beispielsweise mit `System.exit()`.

Java 1.3.1 verhält sich an dieser Stelle nicht ganz korrekt und führt die Shutdown Hooks leider nicht in jedem Fall aus, auch wenn Sie die Option `Xrs` *nicht* angeben – gerade bei `Ctrl+C` wird das Programm einfach beendet.

- ▶ `-XX:+UseAltSigs`
Normalerweise benutzt die Virtual Machine auch die Signale `SIGUSR1` und `SIGUSR2`. Falls dies zu Konflikten führt, können Sie mit dieser Option die Verwendung anderer Signale erzwingen.
- ▶ `-XX:ReservedCodeCacheSize=Größe_in_Bytes`
Legt die maximale Größe für den Code-Zwischenspeicher fest. Standardmäßig werden 32 MByte (»32M«) verwendet.

Assertion-Optionen

Seit Java 1.4 gibt es so genannte »Zusicherungen« (Assertions) mit dem `assert`-Statement. Damit können Sie Annahmen über Programmmzustände und Parameterwerte direkt im Programmcode ausdrücken. Die Annahmen können Sie dann im Testcode prüfen, im Produktionscode aus Performanzgründen jedoch ignorieren lassen. `assert` erwartet einen Ausdruck mit `boolean`-Ergebnis und wird wie folgt verwendet:

```
// Annahme: nenner ist an dieser Stelle immer ungleich Null
assert (nenner != 0);
```

¹ Mit `java.lang.Runtime.getRuntime().addShutdownHook()` können Sie solchen Aufräumcode ins Laufzeitsystem einhängen.

```
int ergebnis = (zaehler / nenner);
```

Eine zweite Form erlaubt es, nach dem Testausdruck und einem Doppelpunkt einen Fehlerwert anzugeben. Der Typ des Fehlerwertes ist beliebig und kann beispielsweise eine Zeichenkette sein:

```
assert (nenner != 0) : "Nenner ist wider Erwarten Null";
```

Wenn der Testausdruck während der Programmausführung zu `true` ausgewertet wird, passiert nichts Besonderes, das Programm läuft einfach weiter. Ergibt der Ausdruck jedoch `false`, wird von der Laufzeitumgebung ein `java.lang.AssertionError` geworfen, dem als Konstruktor-Parameter der optionale Fehlerwert übergeben wird. Sie sollten diesen Ausnahmefehler niemals abfangen, damit Sie nicht eingetretene Annahmen – also Programmierfehler – durch einen Programmabbruch erkennen können!

Da Sie »assert« vor Java 1.4 problemlos als Bezeichner verwenden konnten, müssen Sie es dem Compiler explizit sagen, wenn Sie einen Quelltext speziell für Java 1.4 geschrieben haben, in dem `assert` als Anweisung vorkommt. Dazu übergeben Sie beim Übersetzen die Option `-source`:

```
javac -source 1.4 KlasseMitAssertions.java
```

Wenn Sie so übersetzten Code ausführen lassen, sind Assertions standardmäßig deaktiviert. Mit den folgenden Optionen können Sie Assertions für bestimmte Klassen und Pakete gezielt ein- und ausschalten:

- ▶ `-enableassertions`
 - `-ea`
 - `-enableassertions:KlasseOderPaket`
 - `-ea:KlasseOderPaket`

Ohne Argumente aktiviert diese Option die Auswertung von Assertions für alle Klassen außer für Systemklassen. Wenn Sie einen Klassennamen übergeben, werden Assertions für genau diese Klasse aktiviert. Alles andere wird als Paket angesehen und muss mit drei Punkten enden – dann werden Assertions für dieses Paket und alle Unterpakete eingeschaltet. Wenn Sie nur »...« übergeben, werden Assertions bei Klassen im anonymen Paket (dem aktuellen Arbeitsverzeichnis) ausgewertet.

- ▶ `-disableassertions`
 - `-da`
 - `-disableassertions:KlasseOderPaket`
 - `-da:KlasseOderPaket`

Ohne Argumente deaktiviert diese Option die Auswertung von Assertions für alle Klassen (außer für Systemklassen, bei denen dies mit `-esa` und `-dsa`



gesteuert wird). Wenn Sie einen Klassennamen übergeben, werden Assertions für genau diese Klasse deaktiviert. Alles andere wird als Paket angesehen und muss mit drei Punkten enden – dann werden Assertions für dieses Paket und alle Unterpakete ausgeschaltet. Wenn Sie nur »...« übergeben, deaktivieren Sie die Auswertung von Assertions bei Klassen im anonymen Paket (dem aktuellen Arbeitsverzeichnis).

- ▶ `-enablesystemassertions`
`-esa`
Aktiviert Assertions für alle Systemklassen.
- ▶ `-disablesystemassertions`
`-dsa`
Deaktiviert Assertions für alle Systemklassen.

Mit folgendem Aufruf wird also die Klasse `Test` ausgeführt, wobei für alle Klassen im Paket `com.muchsoft.mac` (und in allen Unterpaketen) Assertions ausgewertet werden – abgesehen von der Klasse `com.muchsoft.mac.NoProblem`, für die Assertions explizit deaktiviert sind:

```
java -ea:com.muchsoft.mac... -da:com.muchsoft.mac.NoProblem Test
```

Falls Sie einmal innerhalb des Codes feststellen müssen, ob Assertions aktiviert sind, können Sie das mit folgenden beiden Zeilen abfragen. Beachten Sie, dass es sich in beiden Zeilen um Zuweisungen handelt! Dadurch gelingt die Assertion immer, und sie können anschließend die Variable `assertionsAktiviert` abfragen:

```
boolean assertionsAktiviert = false;  
assert assertionsAktiviert = true;
```

Spezielle Mac OS X-Optionen

- ▶ `-Xdock:icon=Pfad`
Wenn Sie mit dieser Option den Pfad einer Icon-Datei festlegen, verwendet Mac OS X dieses Icon als Programmsymbol. Ansonsten wird ein allgemeines Java-Programmsymbol dargestellt (siehe Kapitel 4, *Ausführbare Programme*).
- ▶ `-Xdock:name=Applikationsname`
Mac OS X zeigt als Programmnamen im Programm-Menü normalerweise den vollqualifizierten Klassennamen der Hauptklasse an. Mit dieser Option können Sie einen kürzeren, »schöneren« Namen setzen, der nicht länger als 16 Zeichen sein sollte (siehe Kapitel 4).

- ▶ `-Xdock:icon=Pfad:name=Applikationsname`

Hiermit können Sie die beiden obigen Optionen zu einer zusammenfassen. Achtung: Während dies bei Java 1.3.1 funktioniert, klappte das nicht mit allen 1.4.1-Versionen. Seit Java 1.4.2 können Sie diese zusammengesetzte Option wieder problemlos verwenden, allerdings sollten Sie den getrennten Angaben den Vorzug geben – dann läuft es mit allen Mac OS X-Java-Versionen (siehe Kapitel 4).

Die deutliche bessere Variante gegenüber allen drei obigen Möglichkeiten ist die Verwendung eines Programmpakets, in dem Sie dann den Programmnamen und das Applikationssymbol in der Datei `Info.plist` konfigurieren können. Auch dies ist in Kapitel 4 beschrieben.

- ▶ `-XX:+UseFileLocking`

Mit dieser Option können Sie das »Carbon File Locking« aktivieren, wenn Ihre Java-Programme Dateien verwenden, die zur selben Zeit von Carbon-Programmen (native Mac-Anwendungen, die die Carbon-Bibliothek einsetzen) benutzt werden. Hierdurch stellen Sie sicher, dass Ihre Anwendung eine Datei nicht modifiziert, während diese von einem anderen Programm gelesen wird. Obwohl noch recht viele Carbon-Applikationen im Einsatz sind, gibt es nicht so häufig Überschneidungen bei den verwendeten Dateien, daher ist die Option standardmäßig deaktiviert und wird (auch aus Performanzgründen) nur selten eingesetzt.

Heap und Stack

- ▶ `-XmsGröße_in_Bytes`

Setzt die anfängliche Größe für den Java-Heap. Standard sind 2 MByte (»2M«), gültige Werte müssen ein Vielfaches von (und größer als) 1.024 Bytes sein. Die `-server`-Option erhöht den Standardwert auf 32 MByte.

- ▶ `-XmnGröße_in_Bytes`

Setzt die anfängliche Heap-Größe für die »Eden«-Generation. Standardwert ist »640K«, die `-server`-Option erhöht diesen auf 2 MByte. Ab Java 1.4.

- ▶ `-XmxGröße_in_Bytes`

Legt die Maximalgröße fest, bis zu der der Java-Heap anwachsen kann. Standard sind 64 MByte (»64M«), die `-server`-Option erhöht diesen Wert auf 128 MByte. Der größte zulässige Wert liegt derzeit bei ca. 2 GByte (»2048M«).

- ▶ `-XssGröße_in_Bytes`

Legt die Größe des Stacks fest.

- ▶ `-XX:-UseSharedGen`

Bis Mac OS X 10.1 konnte der Heap nicht größer als 590 MByte sein. Mit

dieser Option können Sie den Speicher für gemeinsam genutzte Klassen deaktivieren, wodurch der Heap dann ca. 980 MByte groß werden kann. Wird nur bis Java 1.3 unterstützt.

Garbage Collection: Speicherverbrauch

Viele Garbage Collection-Optionen hängen von der Größe des Java-Heaps ab. Stellen Sie sicher, dass Sie erst die Speichergröße passend einstellen, bevor Sie dann mit den folgenden Optionen detailliert konfigurieren, wie dieser Speicher genutzt wird!

- ▶ `-XX:MinHeapFreeRatio=Prozent_als_Ganzzahl`
Legt die minimale Größe des Heap-Speichers in Prozent fest, die nach einer Garbage Collection frei sein muss. Standardwert ist 40, d.h., wenn nach dem Aufräumen des Speichers nicht mindestens 40% frei sind, wird der Heap vergrößert.
- ▶ `-XX:MaxHeapFreeRatio=Prozent_als_Ganzzahl`
Legt die maximale Größe des Heap-Speichers in Prozent fest, die nach einer Garbage Collection frei sein darf. Standardwert ist 70, d.h., wenn nach dem Aufräumen des Speichers mehr als 70% frei sind, wird der Heap verkleinert.
- ▶ `-XX:NewSize=Größe_in_Bytes`
Setzt die Standardgröße für die »Eden«-Generation. Standardwert ist »640K«, die `-server`-Option erhöht dies auf 2 MByte.
- ▶ `-XX:MaxNewSize=Größe_in_Bytes`
Legt die Maximalgröße für die »Eden«-Generation fest. Auch hier ist der Standardwert »640K«, was von der `-server`-Option wieder auf 2 MByte vergrößert wird.
- ▶ `-XX:NewRatio=Wert`
Konfiguriert das Verhältnis des »Eden«-Speichers für neue Objekte gegenüber älteren Objekten. Standardwert ist 8, d.h., der »Eden«-Speicher beträgt 1/8 der Größe des Speichers für ältere Objekte (der so genannten »Tenured«-Generation).
- ▶ `-XX:SurvivorRatio=Zahl`
Ändert das Verhältnis vom »Eden«-Speicher zu dem, der nach einer Garbage Collection für die »überlebenden« Objekte verwendet wird. Standardwert ist 10, wobei der »Eden«-Speicher »SurvivorRatio+2«-mal größer als der Speicher für die überlebenden Objekte ist.
- ▶ `-XX:TargetSurvivorRatio=Prozent`
Gewünschte Größe des Speichers für »überlebende« Objekte nach dem Umkopieren der Garbage Collection. Der Wert wird in Prozent angegeben, Standard ist 50.

- ▶ `-XX:MaxPermSize=Größe_in_Megabytes`
Legt die Größe der so genannten »permanenten« Generation fest. Standardgröße sind ab dem Java 1.4.2 Update 1 64 MByte (»64M«), davor 32 MByte.

Garbage Collection: allgemeine Einstellungen

- ▶ `-Xincgc`
Diese Option hat bei MacOS X keinen Effekt, da der so genannte »Train« Garbage Collector für eine inkrementelle Garbage Collection nicht unterstützt wird.
- ▶ `-Xnoclassgc`
Deaktiviert die Garbage Collection für Klassen.
- ▶ `-Xloggc:Datei`
Zeichnet jedes Garbage Collection-Ereignis in der angegebenen Datei auf. Zusätzlich zu den Informationen von `-verbose:gc` wird hierbei auch die relative Zeit (in Sekunden) zum ersten GC-Ereignis angegeben. Diese Datei wird schnell sehr groß, geben Sie daher immer eine lokale Datei (und keine auf einem Netzwerk-Volume) an. Wenn sowohl diese Option als auch `-verbose:gc` angegeben sind, hat `loggc` Vorrang. Ab Java 1.4.
- ▶ `-XX:+UseConcMarkSweepGC`
Aktiviert nebenläufige »Mark & Sweep« Garbage Collection. Wirkt sich nur bei Mehrprozessorsystemen aus. Ab Java 1.4.
- ▶ `-XX:+UseParallelGC`
Aktiviert parallele Garbage Collection. Wirkt sich nur bei Mehrprozessorsystemen aus. Ab Java 1.4.
- ▶ `-XX:-DisableExplicitGC`
Wenn diese Option gesetzt ist, werden Aufrufe von `System.gc()` ignoriert. Eine Garbage Collection wird natürlich immer noch von Zeit zu Zeit durchgeführt, aber Sie können dies nicht mehr explizit anfordern.
- ▶ `-XX:+PrintTenuringDistribution`
Gibt Informationen über das Alter der Objekte in der »Eden«-Generation aus. Wenn diese alt genug sind, können sie in die »Tenured«-Generation übernommen werden.

Übersetzung zur Laufzeit (JIT-Compiler)

- ▶ `-Xint`
Mit dieser Option läuft die Java Virtual Machine im reinen Interpreter-Modus, es wird also kein Bytecode mit dem adaptiven JIT-Compiler in Maschinencode übersetzt.

- ▶ `-Xmixed`
Führt die Java VM im gemischten Modus aus, d. h., wenn bestimmter Bytecode häufig genug aufgerufen wird, werden diese Stellen zur Laufzeit in Maschinencode übersetzt. Dies ist der Standardmodus, wenn keine Option angegeben ist.
- ▶ `-Xbatch`
Deaktiviert nebenläufige JIT-Übersetzung. Diese Option hat bei Mac OS X keinen Effekt.
- ▶ `-XX:CompileThreshold=Wert`
Verändert die Zahl der Aufrufe, ab der eine Methode vom JIT-Compiler übersetzt wird. Normalerweise passiert dies nach 1.000 Aufrufen.
- ▶ `-XX:+CITime`
Zeigt an, wieviel Zeit in kompilierten Methoden verbracht wird. Ab Java 1.4.
- ▶ `-XX:+PrintCompilation`
Zeigt an, wenn und welche Methoden kompiliert werden.

Thread-Verarbeitung

- ▶ `-XX:NewSizeThreadIncrease=Größe_in_Kilobytes`
Hiermit können Sie festlegen, in welchen Schritten der Speicher für neue Objekte (»Eden«-Speicher) pro Thread erhöht wird. Normalerweise sind dies 16 KByte (»16«).
- ▶ `-XX:ThreadStackSize=Größe_in_Kilobytes`
Verändert die Thread-Stack-Größe, die sonst vom Betriebssystem vorgegeben wird.
- ▶ `-XX:+UseTLAB`
Aktiviert den Thread-lokalen Speicheranforderungs-Puffer. Bei stark nebenläufigen Anwendungen kann dadurch die Speicherverwaltung besser skaliert werden, was zu deutlich besserer Performanz führt. Auf Mehrprozessorsystemen und bei Mac OS X Server ist diese Option standardmäßig eingeschaltet. Ab Java 1.4.

Gemeinsamer Speicher

- ▶ `-XX:+PrintSharedSpaces`
Zeigt Informationen über gemeinsam genutzten Speicher an. Dies betrifft die so genannte »Immortal«-Generation der Java Shared Archive (JSA)-Technologie. Ab Java 1.4.
- ▶ `-XX:-UseSharedSpaces`
Diese Option deaktiviert die Verwendung von gemeinsamem Speicher. Ab Java 1.4.

F Xcode- und Project Builder-Einstellungen

»Wenn du die Situation nicht ändern kannst,
dann ändere deine Einstellung.« (Chinesisches Sprichwort)

In diesem Kapitel sind die Einstellungen aufgelistet, mit denen Sie den Build-Prozess der Apple-Entwicklungsumgebungen Xcode und Project Builder beeinflussen können. Die angegebenen Variablen können Sie bei der »Expert View« in den »Build Settings« eines Targets setzen. Häufiger jedoch nehmen Sie die gewünschten Einstellungen in den anderen Dialogseiten der Target-Einstellungen vor, wodurch dann die entsprechenden Build-Variablen automatisch in die Liste aufgenommen werden (siehe Abbildung E.1).

Wichtiger als das Setzen der Variablen ist oft das Auslesen der Werte. So können Sie in einer »Shell Script Files«-Build-Phase aus den vordefinierten Projektverzeichnissen neue Pfade erstellen – beispielsweise `${TEMP_DIR}/preverified`. Je nach Kontext lesen Sie die Einstellungsvariable `TEMP_DIR` mit den Zeichenketten `$TEMP_DIR`, `$(TEMP_DIR)` oder `${TEMP_DIR}` aus. Wenn Sie für den Build-Prozess die Kommandozeilenbefehle `xcodebuild` oder `pbxbuild` einsetzen, werden Sie aber auch einige der Variablen setzen, indem Sie sie als Option übergeben.

Im Folgenden sind die wesentlichen Einstellungen für Java-Projekte aufgelistet. Eine komplette Übersicht über alle weiteren Einstellungen, die Cocoa- und Carbon-Projekte betreffen, erhalten Sie, wenn Sie in Xcode den Menüpunkt **Help** • **Show Build Settings Notes** aufrufen.

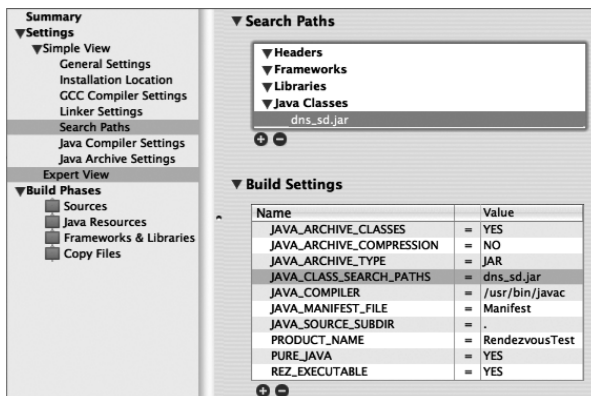


Abbildung F.1 Build-Einstellungen für ein Xcode-Target

F.1 Projekteinstellungen

Die folgenden Einstellungen legen die grundlegenden Werte und Pfade des Projekts fest. Die Wurzel- oder Basispfade enden auf `ROOT` und werden von den speziellen Pfaden verwendet, die auf `_DIR` oder `_PATH` enden. Die Werte gelten für alle Targets eines Projekts.

- ▶ `PROJECT_NAME`
Der Name des Projekts. Nur Lesen.
- ▶ `SYMROOT`
Dies ist das Wurzelverzeichnis für die fertigen Produkte und wird mit `<Projektverzeichnis>/build` vorbelegt.
- ▶ `BUILD_DIR`
In diesem Verzeichnis werden die temporären Dateien aller Targets abgelegt. Standardmäßig ist dieser Pfad mit `SYMROOT` identisch.
- ▶ `TARGET_BUILD_DIR`
In diesem Verzeichnis legt das jeweilige Target seine Zwischenprodukte ab. Bei einem Entwicklungs-Build ist der Pfad mit `BUILD_DIR` identisch. Bei einem Installations-Build wird bei tatsächlich installierten Produkten der Wert von `INSTALL_DIR` verwendet, ansonsten `$BUILD_DIR/Uninstalled-Products`.
- ▶ `BUILT_PRODUCTS_DIR`
Das Wurzelverzeichnis für sämtliche Produkte, die sich dort entweder als fertig erzeugtes Target oder als Link darauf befinden. Normalerweise ist dieser Pfad mit `BUILD_DIR` identisch. Wenn Dateien erzeugt werden, die von verschiedenen Produkten genutzt werden, sollten Sie sie in diesem Verzeichnis ablegen.

F.2 Installationseinstellungen

Hiermit werden der Installationspfad und die Zugriffsrechte des fertigen Produkts konfiguriert.

- ▶ `SKIP_INSTALL`
Wenn Sie diese Variable auf `YES` setzen, wird das erzeugte Produkt nicht in den Installationspfad kopiert. Dies ist nützlich, wenn ein Target ein Produkt erzeugt, das ausschließlich in andere Produkte eingebunden wird.
- ▶ `DEPLOYMENT_LOCATION`
Normalerweise werden die erzeugten Produkte in das Verzeichnis `BUILT_PRODUCTS_DIR` kopiert. Wenn Sie diese Variable auf `YES` setzen, werden stattdessen die folgenden Installationspfade beachtet.

- ▶ `INSTALL_OWNER`
`INSTALL_GROUP`
`INSTALL_MODE_FLAG`

Mit diesen Variablen legen Sie den Besitzer, die Gruppe und die Zugriffsrechte für das Produkt fest. Standardmäßig werden Besitzer und Gruppe vom installierenden Anwender ermittelt, und die Zugriffsrechte werden auf `ugo-w,o+rX` gesetzt. Wenn die Installation vom Benutzer `root` durchgeführt wird, ist der Besitzer `root` und die Gruppe `staff`.

- ▶ `DSTROOT`

Dies ist das Wurzelverzeichnis für installierte Produkte. Den Standardwert `/tmp/$(PROJECT_NAME).dst` müssen Sie für eine tatsächliche Installation auf einen geeigneten Wert setzen.

- ▶ `INSTALL_PATH`

Hiermit legen Sie den Pfad des installierten Produkts fest, beispielsweise `/MeinApplikationsPfad`. Standardmäßig ist diese Variable nicht definiert.

- ▶ `INSTALL_DIR`

Enthält den kompletten Pfad des installierten Produkts, der sich aus `DSTROOT` und `INSTALL_PATH` zusammensetzt. Nur Lesen.

F.3 Target-Einstellungen

- ▶ `TARGET_NAME`

Der Name des Targets. Nur Lesen.

- ▶ `PRODUCT_NAME`

Der Name des vom Target erzeugten Produkts. Nur Lesen.

- ▶ `ACTION`

Gibt an, mit welchem Befehl das Target ausgeführt wird. Project Builder und Xcode übergeben `build` (Standard) und `clean`, im Terminal können Sie aber auch `install` verwenden. Falls Sie den Installationsbefehl einsetzen, müssen Sie `DEPLOYMENT_LOCATION` auf `YES` setzen. Nur Lesen.

- ▶ `SRCROOT`

Dies ist das Wurzelverzeichnis für die Quelltexte des Projekts. Standardvorgabe ist der Wert von `PWD` oder `».«` (falls `PWD` leer ist).

- ▶ `OBJROOT`

Das Wurzelverzeichnis für Zwischenprodukte des Build-Prozesses, normalerweise `$(SRCROOT)/build`.

- ▶ `TEMP_DIR`

`TEMP_FILES_DIR`

Das Verzeichnis für die temporären Dateien eines Targets. Vorgabe ist

`$(OBJROOT)/$(PROJECT_NAME).build/$(TARGET_NAME).build`. Sie sollten diese Einstellung normalerweise nicht verändern.

F.4 Java-Einstellungen

Die folgenden Einstellungen betreffen die Bearbeitung von Java-Targets – man kann damit den Compiler und die diversen Java-Pfade konfigurieren. Entsprechend sind diese Einstellungen nur in Java-Targets verfügbar, nicht aber in den nativen Target-Typen.

- ▶ `JAVA_COMPILER`
Legt den Java-Compiler fest. Normalerweise ist dies `/usr/bin/javac`, aber `/usr/bin/jikes` kann ebenso verwendet werden.
- ▶ `JAVA_COMPILER_DEBUGGING_SYMBOLS`
Mit der Standardeinstellung `YES` wird der Compiler dazu veranlasst, Debug-Informationen zu erzeugen.
- ▶ `JAVA_COMPILER_DISABLE_WARNINGS`
Mit der Standardeinstellung `NO` werden Warnungen des Compilers unterdrückt.
- ▶ `JAVA_COMPILER_DEPRECATED_WARNINGS`
Mit `YES` gibt der Compiler Warnungen bei der Verwendung veralteter Methoden und Klassen aus. Standardeinstellung ist `NO`.
- ▶ `JAVA_COMPILER_TARGET_VM_VERSION`
Diese Einstellung legt fest, für welche JVM-Version die Klassen-Dateien erzeugt werden sollen. Gültige Werte sind »1.1«, »1.2«, »1.3«, »1.4« usw.
- ▶ `JAVA_COMPILER_SOURCE_VERSION`
Mit dem Wert »1.4« kann Java-Quelltext mit `assert`-Anweisungen übersetzt werden. Gültige Werte sind derzeit »1.3« und »1.4«. Nur bei Xcode.
- ▶ `JAVA_SOURCE_FILE_ENCODING`
Legt die Zeichenkodierung der Quelltexte fest. Standardwert ist `MACINTOSH`. Beachten Sie, dass sowohl `javac` als auch `jikes` dieselbe Kodierung bei allen zu übersetzenden Dateien voraussetzen.
- ▶ `JAVA_COMPILER_FLAGS`
Hiermit können Sie Compiler-Optionen definieren, die nicht von anderen Einstellungen abgedeckt sind (beispielsweise `-extdirs` oder `-bootclasspath`).
- ▶ `JAVAC_DEFAULT_FLAGS`
Legt die grundlegenden Optionen für alle `javac`-Aufrufe fest. Wenn Sie nichts anderes definieren, wird `-J-Xms64m -J-XX:NewSize=4M -J-Dfile.encoding=UTF8` verwendet. Sie sollten diese Einstellung normalerweise nicht verändern.

- ▶ `JIKES_DEFAULT_FLAGS`
Legt die grundlegenden Optionen für alle `jikes`-Aufrufe fest. Wenn Sie nichts anderes definieren, wird `+E +OLDCSO` verwendet. Sie sollten diese Einstellung normalerweise nicht verändern.
- ▶ `JAVA_ARCHIVE_CLASSES`
Der Standardwert `YES` legt fest, dass die Java-Klassen zu einem JAR-Archiv zusammengefasst werden. Mit `NO` werden die einzelnen Klassen ins fertige Produkt kopiert, mit `BOTH` (nur bei Xcode) sowohl die Klassen als auch das Archiv.
- ▶ `JAVA_ARCHIVE_COMPRESSION`
Wenn Sie diese Variable auf `YES` setzen, wird der Inhalt des JAR-Archivs komprimiert. Standardmäßig ist nichts vorgegeben, wodurch die Archive unkomprimiert sind.
- ▶ `JAVA_ARCHIVE_TYPE`
Der Standard-Archivtyp ist `JAR`, aber `ZIP` wird ebenso unterstützt.
- ▶ `CLASS_ARCHIVE_SUFFIX`
Die Dateinamenserweiterung des Programmarchivs ist standardmäßig `.jar` oder `.zip`, je nach Einstellung von `JAVA_ARCHIVE_TYPE`. Sie können hier aber auch `.war` oder `.ear` eintragen.
- ▶ `CLASS_ARCHIVE`
Dies ist ein produktrelativer Pfad zum erzeugten Java-Programmarchiv (inklusive der festgelegten Dateinamenserweiterung).
- ▶ `LINKED_CLASS_ARCHIVES`
Eine durch Leerzeichen getrennte Liste von Java-Archiven, die fest in das Produkt eingebunden werden.
- ▶ `JAVA_CLASS_SEARCH_PATHS`
Eine durch Leerzeichen getrennte Liste von Java-Archiven, die bei der Produkterzeugung vorhanden sein müssen (d.h., sie werden für den Compiler auf den Klassenpfad gesetzt, aber nicht fest in das Produkt eingebunden).
- ▶ `OTHER_JAVA_CLASS_PATH`
Eine durch Doppelpunkt getrennte Liste von Archiven, die auf den Compiler-Klassenpfad gesetzt werden.
- ▶ `JAVA_MANIFEST_FILE`
Ein projektrelativer Pfad auf eine Textdatei, die zum Standardmanifest `MANIFEST.MF` hinzugefügt wird.
- ▶ `CLASS_FILE_DIR`
Das Wurzelverzeichnis für übersetzte Java-Klassen, normalerweise `$TEMP_FILES_DIR/JavaClasses`. Sie sollten diese Einstellung nicht verändern.

- ▶ `CLASS_FILE_LIST`
Diese temporäre Datei listet alle Klassen-Dateien auf, die kopiert oder archiviert werden. Sie sollten diese Einstellung nicht verändern.
- ▶ `JAVA_SOURCE_SUBDIR`
Legt den projektrelativen Pfad für die Quelltexte fest. Standardmäßig ist hier nur ».« eingetragen, also das Projektverzeichnis.
- ▶ `JAVA_SOURCE_PATH`
Eine durch Doppelpunkt getrennte Liste von projektrelativen oder absoluten Pfadangaben, die für die `-sourcepath`-Option von `javac` verwendet wird. Standardvorgabe ist der Wert von `JAVA_SOURCE_SUBDIR`.
- ▶ `JAVA_USE_DEPENDENCIES`
Mit dem Wert `NO` können Sie den Projekt-Index abschalten, durch den Build-Abhängigkeiten erkannt werden. Sie sollten diese Option nicht verwenden. Nur bei Xcode.
- ▶ `JAVA_FORCE_FILE_LIST`
Mit dem Wert `YES` zwingen Sie die Entwicklungsumgebung, die Quelltexte an den Java-Compiler nicht als Kommandozeilen-Argumente zu übergeben, sondern als Liste in einer Textdatei. Nur bei Xcode.

F.5 Programmpaket-Einstellungen

Wenn ein Java-Target ein Mac OS X-Programmpaket (Bundle) erzeugt, können Sie die folgenden Variablen setzen oder auswerten.

- ▶ `PACKAGE_TYPE`
Der Pakettyp gibt die Art des erzeugten Produkts an. Für Java-Anwendungen sind die Werte `EXECUTABLE`, `JAR_FILE`, `ZIP_FILE` und `JAVA_CLASS_FOLDER` relevant.
- ▶ `INFO_PLIST_FILE`
Enthält den Pfad der Paket-Konfigurationsdatei, standardmäßig `Contents/Info.plist`. Sie sollten diese Einstellung nicht verändern.
- ▶ `DEVELOPMENT_PLIST_FILE`
Enthält den Pfad der Produkt-Konfigurationsdatei, standardmäßig `Contents/pbdevelopment.plist`. Sie sollten diese Einstellung nicht verändern.
- ▶ `PKGINFO_FILE`
Enthält den Pfad der Paket-Informationsdatei, standardmäßig `Contents/PkgInfo`. Sie sollten diese Einstellung nicht verändern.
- ▶ `JAVA_APP_STUB`
Gibt den Pfad des nativen Java-Programmstarters an, der in ein Programmpaket kopiert wird. Sie sollten diese Einstellung nicht verändern.

F.6 Standardpfade

Die folgenden Variablen enthalten die wichtigsten Standardverzeichnisse von MacOS X. Benutzen Sie nach Möglichkeit diese Variablen anstelle von fest kodierten Pfadangaben, da sich die Werte bei zukünftigen Systemversionen durchaus ändern können.

- ▶ SYSTEM_APPS_DIR
/Applications
- ▶ SYSTEM_ADMIN_APPS_DIR
/Applications/Utilities
- ▶ SYSTEM_DEMOS_DIR
/Applications/Extras
- ▶ SYSTEM_DEVELOPER_DIR
/Developer
- ▶ SYSTEM_DEVELOPER_APPS_DIR
/Developer/Applications
- ▶ SYSTEM_DEVELOPER_JAVA_TOOLS_DIR
/Developer/Applications/Java Tools
- ▶ SYSTEM_DEVELOPER_PERFORMANCE_TOOLS_DIR
/Developer/Applications/Performance Tools
- ▶ SYSTEM_DEVELOPER_GRAPHICS_TOOLS_DIR
/Developer/Applications/Graphics Tools
- ▶ SYSTEM_DEVELOPER_UTILITIES_DIR
/Developer/Applications/Utilities
- ▶ SYSTEM_DEVELOPER_DEMOS_DIR
/Developer/Applications/Utilities/Built Examples
- ▶ SYSTEM_DEVELOPER_DOC_DIR
/Developer/Documentation
- ▶ SYSTEM_LIBRARY_DIR
/System/Library
- ▶ SYSTEM_CORE_SERVICES_DIR
/System/Library/CoreServices
- ▶ SYSTEM_DOCUMENTATION_DIR
/Library/Documentation
- ▶ LOCAL_ADMIN_APPS_DIR
/Applications/Utilities

- ▶ LOCAL_APPS_DIR
/Applications
- ▶ LOCAL_DEVELOPER_DIR
/Library/Developer
- ▶ LOCAL_LIBRARY_DIR
/Library
- ▶ USER_APPS_DIR
\$(HOME)/Applications
- ▶ USER_LIBRARY_DIR
\$(HOME)/Library

G Mac OS X- und Java-Versionen

»Keine Atempause, Geschichte wird gemacht, es geht voran!«
(Fehlfarben)

Dieses Kapitel gibt Ihnen einen schnellen Überblick darüber, welche Java-Versionen von Apple für Mac OS X veröffentlicht wurden, wie Sie diese Versionsinformationen gezielt abfragen können und wofür Sie dies einsetzen sollten.

Mac OS X erkennen

Wenn Sie bestimmten Code nur auf Mac OS X ausführen möchten oder können, beispielsweise die Anpassung einer Menüzeile an die Mac-Funktionalität, müssen Sie feststellen, ob Ihre Java-Anwendung auf Mac OS X läuft. Wie Sie schon in Kapitel 1, *Grundlagen*, gesehen haben, erledigen Sie dies am besten – und kompatibel zu J2SE auf anderen Plattformen – mit der System-Property `os.name`:

```
String osname = System.getProperty("os.name").toLowerCase();
boolean ismacosx = osname.startsWith("mac os x");
```

Listing G.1 Abfrage auf Mac OS X

Am einfachsten können Sie dazu `com.muchsoft.util.Sys.isMacOSX()` aus dem Archiv `Sys.jar` verwenden, das Sie auf der Buch-CD im Verzeichnis `/utility/sys/` finden.

Java-Version ermitteln

Wenn Ihre Applikation bestimmte, erst ab Java 1.4 verfügbare APIs verwendet, können Sie die Java-Version ermitteln, um bei einem zu alten Java entweder den Anwender zu warnen oder um alternative APIs aufzurufen. Dazu verwenden Sie die System-Property `java.version`:

```
String javaVersion = System.getProperty("java.version");
if (javaVersion.startsWith("1.4")) {
    // neue 1.4-APIs aufrufen...
}
```

Listing G.2 Abfrage auf eine Java-Version

Version der Apple-Java-Implementierung

In seltenen Fällen müssen Sie die exakte Version der Apple-Java-Implementierung ermitteln – meistens dann, wenn Apples Implementierung einen Fehler enthielt, der mit einem folgenden Update beseitigt wurde. Dann können Sie beispielsweise den Anwender auffordern, die fehlerbereinigte Java-Version mit der Software-Aktualisierung einzuspielen.

Die normale Java-Version hilft Ihnen hier nicht weiter, denn dies sind Suns Versionsnummern für Java. Apple veröffentlicht aber nicht zwingend jedes kleine Sun-Java-Update auch für Mac OS X – die erste 1.4.2er Version war z. B. 1.4.2_03 und nicht 1.4.2_01. Außerdem könnte es durchaus vorkommen, dass Apple zur Fehlerkorrektur ein Update mit exakt gleich bleibender Java-Version veröffentlicht. Zur Abfrage verwenden Sie mehrere System-Properties, unter anderem `java.runtime.version`:

```
String jrv = System.getProperty("java.runtime.version");
if (ismacosx && jrv.startsWith("1.4.1") && jrv.endsWith("-69.1")) {
    // Mac OS X 10.2 mit Java 1.4.1 Update 1 erkannt (s.u.)
}
```

Listing G.3 Abfrage auf eine ganz bestimmte Apple-Java-Implementierung

Beachten Sie, dass Sie unbedingt auf Mac OS X und die passende Java-Version testen müssen, bevor Sie die sehr spezielle Version der Apple-Java-Implementierung abfragen!

Versionsübersicht

Die folgenden Tabellen zeigen Ihnen die bisherigen öffentlich verfügbaren Java-Versionen für Mac OS X. Eine aktuelle Liste finden Sie auf der Seite <http://developer.apple.com/technotes/tn2002/tn2110.html>. Nicht berücksichtigt sind dabei Entwickler-Vorabversionen.

Die Spalte für die System-Property `mrj.version` ist nur der Vollständigkeit halber aufgeführt. Sie sollten diese Property nicht mehr zur Abfrage auf Mac OS X oder zur Erkennung bestimmter Versionen einsetzen. Aus Kompatibilitätsgründen mit dem klassischen Mac OS ist diese Property zwar auch bei Mac OS X immer noch vorhanden, aber Apple gibt keine Garantie mehr für den Aufbau der Zeichenkette oder die Existenz der Property in zukünftigen Java-Versionen.

Installiertes Mac OS X und Apple-Java	java.version	java.runtime.version	mrj.version
10.0	1.3.0	1.3	3.0
10.1	1.3.1	1.3.1-root-010902-18:51	3.1
1.3.1 Update 1	1.3.1	1.3.1-root_1.3.1_020714-12:46	3.2
10.2	1.3.1	1.3.1-root-010902-18:51	3.3
1.4.1 Update 1	1.3.1	1.3.1-root_1.3.1_030709-15:51	3.3.2
10.3	1.3.1	1.3.1-root_1.3.1_030912-19:52	3.3.3
1.4.2	1.3.1	1.3.1-root_1.3.1_031103-17:49	3.3.3
1.4.2 Update 1	Keine Änderung		
1.4.2 Update 2			

Tabelle G.1 Java 1.3-Versionen für Mac OS X

Installiertes Mac OS X und Apple-Java	java.version	java.runtime.version	mrj.version
10.2	Mac OS X 10.2 wurde anfangs nur mit Java 1.3.1 ausgeliefert. Java 1.4 konnte dann später als Software-Aktualisierung eingespielt werden.		
1.4.1	1.4.1_01	1.4.1_01-39	4.1
1.4.1 Update 1	1.4.1_01	1.4.1_01-69.1	69.1
10.3	1.4.1_01	1.4.1_01-99	99
1.4.2	1.4.2_03	1.4.2_03-117.1	117.1
1.4.2 Update 1	1.4.2_05	1.4.2_05-141	141
1.4.2 Update 2	1.4.2_05	1.4.2_05-141.3	141.3

Tabelle G.2 Java 1.4-Versionen für Mac OS X

Der Aufbau der Zeichenketten in `java.version` und `java.runtime.version` ist von Sun für alle Java-Systeme auf der Seite http://java.sun.com/j2se/versioning_naming.html spezifiziert.

H Glossar

»Sprache ist, das wissen wir, das allerhöchste Gut, und ohne Klarheit in der Sprache ist der Mensch nur ein Gartenzweig.«
(*Element of Crime*)

Alias Ein Verweis innerhalb des Dateisystems, bei anderen Betriebssystemen oft *Link* genannt. Meistens ist damit ein symbolischer Link gemeint, ein Verweis auf eine andere Datei oder ein anderes Verzeichnis.

Antialiasing Weichzeichnen von Grafikelementen. Schriften und Linien haben keine abrupten Farbwechsel an den Elementgrenzen, sondern fließende Farbübergänge.

API »Application Programming Interface«, Programmierschnittstelle

Aqua ist der Name der grafischen Benutzungsoberfläche von Mac OS X.

ASCII Der »American Standard Code of Information Interchange« ist das typische Format reiner Textdateien.

AWT Das »Abstract Windowing Toolkit« ist die ältere der beiden Java-Standardbibliotheken für grafische Benutzungsoberflächen (die neuere heißt *Swing*).

BMP Bezeichnet entweder eine *EJB*-Komponente mit »Bean Managed Persistence« (die Komponente kümmert sich selbst um das Speichern ihrer Daten) oder das »Bitmap«-Grafikformat von Windows.

Bundle Ein Bundle ist eine spezielle Verzeichnisstruktur, die vom *Finder* nicht als Verzeichnis, sondern als eine Datei (ein *Paket*) angezeigt wird.

Bytecode nennt sich der Binärcode, der vom Java-Compiler erzeugt, in `.class`-Dateien gespeichert und dann von der Java-Laufzeitumgebung ausgeführt wird.

Carbon ist neben *Cocoa* und Java eine Programmierschnittstelle bzw. Bibliothek für Mac OS X, die für Anwendungen eingesetzt wird, die sowohl unter *Classic* als auch unter Mac OS X laufen sollen.

Cast Auch »Typecast« genannt, eine Typumwandlung eines Wertes von einem Datentyp in einen andere

CGI Das »Common Gateway Interface« bezeichnet eine ältere Schnittstelle, mit denen Programme auf einem Web-Server dynamische Daten an den Aufrufer liefern können.

Classic Die Classic-Umgebung ist die in Mac OS X integrierte Emulationssoftware für Apples altes Betriebssystem Mac OS.

CMP Bezeichnet eine *EJB*-Komponente mit »Container Managed Persistence«, d.h., der *EJB*-Container kümmert sich um die Speicherung der Komponenten-Daten.

Cocoa ist neben *Carbon* und Java eine Programmierschnittstelle bzw. Bibliothek, die für objektorientierte, in Objective-C geschriebene Mac OS X-Applikationen verwendet wird.

Compiler Ein Compiler übersetzt den Quelltext eines Programms in ein anderes Format, meistens in ausführbaren Maschinencode oder – im Falle von Java – in *Bytecode*. Wird in diesem Buch auch Übersetzer genannt.

CORBA Die »Common Object Request Broker Architecture« ist ein system- und programmiersprachenübergreifender Standard für verteilte Objektsysteme.

Wird bei Java im Rahmen von *J2EE* bei *EJB* eingesetzt.

Darwin heißt der *Kernel* von Mac OS X.

DB Datenbank

DBMS Datenbank-Management-System oder kürzer Datenbank-System. Meistens einfach Datenbank genannt.

Desktop Auf dem Schreibtisch (»Desktop«) von Mac OS X können Sie alle möglichen Dinge (Dateien, Ordner, Programme) ablegen – wie im richtigen Leben. Verwaltet wird der Desktop vom *Finder*.

Dock Das Dock ist die Programm- und Dokumentleiste von Mac OS X, die sich meist am unteren Bildschirmrand befindet.

DMG »Disk Mountable Image« ist ein Archiv, das bei Doppelklick ein neues *Volume* öffnet. Wird häufig zur Verteilung von Software eingesetzt.

DnD »Drag & Drop« bezeichnet das Anklippen, Ziehen und Fallenlassen von Objekten in grafischen Benutzungsoberflächen.

DNS Der »Domain Name Server« ist ein Rechner, der Internet-Adressnamen wie *www.galileocomputing.de* in das eigentliche *IP*-Zahlenformat 80.237.200.227 umwandelt.

Dylib Eine »dynamic library« oder »dynamic shared library« ist eine Form der Codebibliotheken, die von Mac OS X-Programmen verwendet werden.

EJB »Enterprise JavaBeans«, ein Komponentenmodell für *J2EE*-Server

Escape-Sequenz Eine Escape-Sequenz bezeichnet ein Sonderzeichen, das mit einem vorangestellten umgekehrten Schrägstrich (»Backslash«) eingegeben wird. Beispielsweise bezeichnet `\t` das

Tabulator-Zeichen und `\u20ac` das Euro-Symbol.

Exposé schafft Übersicht bei vielen offenen Fenstern. Mit den Tasten `F9`, `F10` und `F11` können Sie entweder alle Fenster gekachelt anordnen lassen oder kurzfristig ausblenden, um auf den *Desktop* zuzugreifen.

Finder Der Finder ist die zentrale Benutzerschnittstelle von Mac OS X, denn dieses Programm verwaltet den *Desktop* und die Dateifenster.

Framework Ein Framework ist eine Rahmenanwendung, in die man bestimmte Module einklinken kann, um daraus eine vollständige, funktionstüchtige Anwendung zu machen. Bei Mac OS X werden damit Bibliotheken bezeichnet, die mit einer bestimmten Verzeichnisstruktur gespeichert sind.

GNU Das GNU-Projekt (»GNU«s Not UNIX«) will ein komplettes Betriebssystem samt zugehöriger Werkzeuge als freie Software entwickeln. Während Linux das *Open Source*-System geworden ist (was den GNU-Gründern aber nicht frei genug ist), ist GNU vor allem bekannt für seine Compiler.

GPL Die »GNU Public Licence« definiert einen Lizenzvertrag für freie Software. Oft findet man aber auch abgewandelte Lizenzbestimmungen, beispielsweise die LGPL.

GUI »Graphic User Interface«, grafische Benutzungsschnittstelle

HFS+ Das »Hierarchical File System« ist das Standard-Dateisystem für Mac OS X.

Host Als Host wird ein Rechner im Netzwerk bezeichnet, der als Server verschiedene Dienste bereitstellt.

HTML Mit der »Hypertext Markup Language« kann die Struktur von Webseiten festgelegt werden. Meistens wird diese

Sprache auch für das Design und das Layout von Webseiten missbraucht.

HTTP Das »Hypertext Transfer Protocol« wird zur Übertragung von Dateien zwischen Web-Server und Web-Browser eingesetzt.

IB Der »Interface Builder« ist das Programm zur Gestaltung von Benutzungsoberflächen bei *Cocoa*-Anwendungen.

IDE »Integrated Development Environment«, Entwicklungsumgebung

IP »Internet Protocol«

J2EE »Java 2 Platform Enterprise Edition« – Java 1.2 und neuer für Server-Anwendungen

J2ME »Java 2 Platform Micro Edition« – Java 1.2 und neuer für mobile Geräte (z. B. Mobiltelefone)

J2SE »Java 2 Platform Standard Edition« – Java 1.2 und neuer für Desktop-Applikationen. Dies ist die mit Mac OS X ausgelieferte Java-Variante.

Jaguar Nicht nur eine Automarke und eine Spielekonsole von Atari, sondern auch der Codename für Mac OS X 10.2

JAWS »Java Web Start«

JDBC Die Standard-Programmierschnittstelle für *SQL*-Datenbankzugriffe aus Java.

JDK »Java Development Kit«, das Java-Entwicklerpaket. Mittlerweile Java *SDK* genannt.

JDO »Java Data Objects«, ein halbwegs neuer, aber nicht sehr akzeptierter Standard zum Speichern von Objekten.

JNDI »Java Naming & Directory Interface«. Mit dem Java-Namens- und -Verzeichnisdienst können Objekte und Dienste im Netzwerk gefunden werden. Wird unter anderem bei *EJB* eingesetzt.

JRE »Java Runtime Environment«, die Java-Laufzeitumgebung. Ist in Mac OS X fest integriert.

JTA/JTS Die »Java Transaction API« und der »Java Transaction Service« spezifizieren verteilte Transaktionen für Java-Systeme.

Kernel Der Kernel ist der zentrale Kern des Betriebssystems, der sich um die Verwaltung der laufenden Prozesse (Programme) kümmert.

LAF »Look & Feel«, siehe *PLAF*

Link Ein Verweis innerhalb des Dateisystems, bei Mac OS X *Alias* genannt

Lokalisierung Die Anpassung eines Programms an lokale Gegebenheiten, z. B. verschiedene Zahlen- und Währungsformate. Die Übersetzung von Texten ist nur ein Teil davon.

MAC Die MAC-Adresse identifiziert jede Netzwerkkarte weltweit eindeutig.

Mac Kurzform für »Macintosh«

MDI Das von Windows stammende »Multiple Document Interface«, bei dem sich innerhalb eines System-Fensters mehrere Dokument-Fenster befinden, gibt es bei Mac OS X nicht.

MIDP Das »Mobile Information Device Profile« beschreibt einen Anwendungsfall von *J2ME* für Mobiltelefone.

MRJ »Macintosh Runtime for Java«, Apples Java-Laufzeitumgebung für das alte Mac OS (*Classic*).

MVC Das MVC-Konzept teilt eine Anwendung in »Model, View, Controller« auf. Durch dieses Entwurfsmuster werden Darstellung und Daten getrennt.

NetInfo nennt sich bei Mac OS X die Komponente, welche die grundlegenden Systemeinstellungen wie Benutzer, Grup-

pen und Netzwerkeinstellungen verwaltet.

NeXTSTEP war das Betriebssystem der von Apple aufgekauften Firma NeXT und bildete die Grundlage für Mac OS X.

NIB In einer NIB-Datei («*NeXTSTEP* Interface Builder») ist die Benutzungsoberfläche einer *Cocoa*-Applikation gespeichert.

ODBC »Open Database Connectivity« ist ein Standard zum Zugriff auf Datenbanken. Bei Java verwenden Sie stattdessen *JDBC*, das auch auf ODBC zurückgreifen kann.

ODBMS Objekt-Datenbank, siehe *DBMS*

OO Objektorientierung

OOA Objektorientierte Analyse

OOD Objektorientiertes Design

OOP Objektorientierte Programmierung

Open Source bezeichnet Software, deren Quelltexte öffentlich zugänglich sind – beispielsweise auch *Darwin*. Es ist damit aber nichts darüber gesagt, was Sie mit diesem Quelltext machen dürfen, z. B. kann das Verändern oder Weitergeben untersagt sein.

Ordner Ein Verzeichnis bzw. Unterverzeichnis im Dateisystem wird beim Mac auch Ordner genannt.

Package Siehe *Paket*

Paket Ein Paket kann zum einen ein *Bundle* bezeichnen, d. h. ein Mac OS X-Programm- oder -Installationspaket, bei dem Daten in einer bestimmten Verzeichnisstruktur abgelegt werden. Zum anderen kann damit ein Java-Paket gemeint sein, mit dem Klassen gruppiert werden (beispielsweise `java.lang`).

Panther Der Codename von Mac OS X 10.3

Partition Eine Partition bezeichnet einen Teilbereich einer Festplatte. Beim Mac wird dies üblicherweise *Volume* genannt.

PB Der »Project Builder« ist Apples Entwicklungsumgebung für Mac OS X 10.2 und älter. Ab Mac OS X 10.3 kommt *Xcode* zum Einsatz.

Persistenz bezeichnet die Fähigkeit, Daten dauerhaft auf nicht flüchtigem Speicher sicher abzulegen.

PLAF »Pluggable Look & Feel« bezeichnet das austauschbare Aussehen von *Swing*-Anwendungen.

POSIX ist ein Standard für *UNIX*-Systeme, der unter anderem bestimmte System-Routinen festlegt.

Property List Eine Property List ist eine Mac OS X-Konfigurationsdatei für System- und Anwendungseinstellungen, die normalerweise im *XML*-Format gespeichert wird.

RDBMS Relationales Datenbank-System, das Daten in Tabellen und Beziehungen dazwischen abbildet (siehe *DBMS*)

Rendezvous Mit diesem Netzwerkprotokoll von Apple können sich kleine Netze (d. h. die darin enthaltenen Geräte wie Computer und Drucker) automatisch konfigurieren.

root ist der so genannte *Superuser*, der für die Verwaltung des Systems zuständig ist. Da dieser Benutzer wirklich alles darf, können Sie damit beliebig viel kaputt machen – passen Sie also auf, wenn Sie sich als root-Benutzer anmelden! Normalerweise ist das aber überhaupt nicht nötig, da Sie die Superuser-Rechte auch kurzfristig mit `sudo` erlangen können.

SCM »Source Code Management« ist der Oberbegriff von Apples Entwicklungs-

umgebung *Xcode* für die diversen Versionsverwaltungssysteme wie CVS oder Subversion.

SDK »Software Development Kit«, Software-Entwicklungspaket. Wenn Sie Java-Programme entwickeln wollen, müssen Sie normalerweise das Java-SDK herunterladen. Bei Mac OS X ist das SDK fest ins System integriert, Sie müssen sich bei Bedarf nur noch zusätzliche Dokumentationen installieren.

Shell In einer Shell können *UNIX*-Kommandos und *-Skripte* ausgeführt werden. Andere Betriebssysteme nennen dies auch Kommandozeile oder Eingabeaufforderung.

Skript Ein Skript ist eine Zusammenfassung von einzelnen Kommandos, die dann als ein Kommando ausgeführt werden können. Obwohl viele Programmiersprachen dieses Konzept bieten, ist hiermit meistens ein *UNIX*- bzw. *Shell*-Skript gemeint.

SQL »Structured Query Language«, die Standard-Abfragesprache für Datenbanken

SQL/J Während mit *JDBC* dynamisches *SQL* programmiert wird, ist *SQL/J* auf statisches, eingebettetes *SQL* spezialisiert. Dieser Standard hat sich nicht durchgesetzt.

Startvolume Das Startvolume bezeichnet das *Volume*, von dem aus Mac OS X gestartet (»gebootet«) wurde.

Suffix Die Extension eines Dateinamens (z. B. *.gif*) wird in diesem Buch meistens Dateinamenserweiterung genannt.

Superuser Siehe *root*

Swing ist die neuere und deutlich umfassendere Java-Standardbibliothek für grafische Benutzungsoberflächen (die ältere heißt *AWT*).

Terminal Das Terminal ist die grafische Anwendung von Mac OS X, mit der Sie in beliebig vielen Fenstern jeweils eine *Shell* laufen lassen können.

Tiger Der Codename sowohl von Java 1.5 bzw. J2SE 5.0 als auch vom kommenden Mac OS X 10.4. Zufall?

UI »User Interface«, siehe *GUI*

UML Mit der »Unified Modelling Language« können (vor allem objektorientierte) Systeme und Prozesse grafisch entworfen werden.

UNIX ist zum einen die Urversion, zum anderen der Sammelbegriff für diverse Betriebssysteme, darunter Linux, BSD und Darwin bzw. Mac OS X.

URI Ein »Uniform Resource Identifier« ist ein eindeutiger Name für eine beliebige Ressource.

URL Ein »Uniform Resource Locator« ist ein Spezialfall einer *URI*. Mit einer *URL* wird eine Ressource nicht nur eindeutig identifiziert, im Namen sind auch Angaben enthalten, wie und wo diese Ressource gefunden werden kann. Eine Webadresse ist beispielsweise eine *URL*.

Volume Ein Volume bezeichnet beim Mac einen Teil einer Festplatte. Andere Betriebssysteme nennen dies *Partition*.

Xcode ist Apples Entwicklungsumgebung seit Mac OS X 10.3. Davor wurde der »Project Builder« (*PB*) verwendet.

XML Mit der »Extensible Markup Language« werden Textdateien strukturiert. Im Gegensatz zu *HTML* sind diese Dateien nicht zwingend zur Darstellung vorgesehen. Außerdem geht *XML* deutlich strenger mit Fehlern im Dateiformat um.

I Die Buch-CD

»Die Welt ist eine Scheibe.«
(*Modernes Gerücht über das Mittelalter*)¹
»Und sie bewegt sich doch!«
(*Galileo Galilei zugesprochen*)

Die im Buch vorgestellte Software müssen Sie nicht selbst herunterladen, gerade wenn Sie nur einen langsamen Internet-Zugang besitzen. Die wichtigsten Programme und Archive finden Sie auch auf der beiliegenden CD – vielleicht nicht immer in der topaktuellen, dafür aber in einer stabilen, brauchbaren Version. Falls Sie dabei einige Programme vermissen, liegt das oft an der fehlenden Erlaubnis des Herstellers, die Software auf CD weiterzugeben – dies betrifft insbesondere leider jegliche Software von Apple, die Sie von Apples Webseiten herunterladen oder durch die Software-Aktualisierung von MacOS X installieren lassen müssen. Die Verzeichnisstruktur auf der Buch-CD sieht wie folgt aus:

index.html	Diese Webseite enthält das Inhaltsverzeichnis der CD und zahlreiche Links ins Internet.
examples/ ch01/ ... ch17/	Die Unterverzeichnisse enthalten die Beispielprogramme der jeweiligen Kapitel (das »ch« steht für »chapter«), normalerweise als Xcode-Projekte (falls es sich nicht um reine Kommandozeilenbeispiele handelt). Sie sollten mindestens Xcode 1.2 verwenden.
software/ ch02/ eclipse/ netbeans/ ...	Das Software-Verzeichnis zu Kapitel 2 enthält unter anderem aktuelle Versionen von Eclipse und NetBeans, dazu noch einige Programmiereditoren.
ch04/ izpack/	IzPack ist ein systemübergreifendes Installationsprogramm.
ch08/ ant/ junit/	Ant – ein Build-System für Java JUnit – eine Suite für Unit-Tests
ch09/ mysql/ postgresql/	Hier finden Sie die Datenbank-Systeme MySQL und PostgreSQL samt passender JDBC-Treiber.

¹ <http://www.wissenschaft-online.de/artikel/604748>

ch10/ tomcat/	Der Servlet- und JSP-Container Tomcat in einer stabilen 5.x-Version
ch11/ jboss/	Der Applikations-Server JBoss in der aktuellen Version 4.0
urls/	Dieses Verzeichnis enthält zahlreiche »URL-Clips«: Webadressen, die Sie unter Mac OS X einfach durch einen Doppelklick auf die Clip-Datei anzeigen lassen können. Andere Systeme können diese Dateien, die dort mit der Dateinamenserweiterung <code>.webloc</code> angezeigt werden, normalerweise nicht verwenden.
utility/	
sys/	Das Bibliothek »Sys« mit der gleichnamigen Hauptklasse wird in vielen Kapiteln eingesetzt, um systemabhängige Funktionalität zu behandeln. Sie dürfen die Bibliothek bei eigenen Projekten frei einsetzen.
Minimal.app/	In Kapitel 4 werden Mac OS X-Programmpakete vorgestellt, mit denen sich Java-Programme als »richtige« Mac OS X-Anwendungen verpacken lassen. Um auch unter fremden Systemen solche Pakete erzeugen zu können, nehmen Sie einfach dieses Minimal-Paket und passen es an.

Wenn im Fließtext Pfade zu Dateien auf der CD angegeben sind, beginnen sie immer mit `CD`, also beispielsweise `CD/examples/ch01/`. Innerhalb von Quelltexten werden die Pfadangaben in die jeweils gültigen Kommentarzeichen eingeschlossen, also z.B.

```
//CD/examples/ch01/
```

für Java-Quelltexte und

```
<!-- CD/examples/ch01/ -->
```

für XML- und HTML-Dokumente. Weitere Informationen zum Buch sowie aktualisierte Quelltexte und Webadressen finden Sie beim Verlag unter der Adresse

<http://www.galileocomputing.de/683>

und direkt beim Autor auf der Seite

<http://www.muchsoft.com/java/>.

Index

\$APP_PACKAGE 197, 213, 214
\$JAVAROOT 197, 212, 214
.app 166
.hotspotrc 566
.login 65, 414
.MacOSX 65
.mov 366
.pbHelpIndexerList 85
.pbproj 79
.pkg 166
.webloc 596
.xcode 79
/etc/hostconfig 434, 439
/usr/bin/cvs 96
/usr/lib 320
/usr/lib/java 317
/usr/local/ 448
/usr/local/bin 403
/usr/local/mysql/ 433
/usr/local/pgsql/ 438
\n 294
\r 294
_blank 273
127.0.0.1 449
3D-grafik 356
68k 270, 531

A

AAPL 199
Abhängigkeiten von Targets 90, 314
Ablage-Menü 28
Abstand zum Fensterrand 129
abstract 524
Abstract Windowing Toolkit 119, 121
abstrakte Klasse 524
Abstürze 30
ACTION 579
Action 334, 336, 337
action 456
ACTION_COPY 169
ACTION_LINK 169
ACTION_MOVE 169
ActionEvent 126, 153
ActionListener 126
actionPerformed() 126

ADB 301
ADC 46
 Reference Library 43
add() 126
addActionListener() 126
addApplicationListener() 155
addDocumentListener() 159
addMouseListener() 135
addSeparator() 128, 149
addWindowListener() 123
Administrator-Rechte 67, 348
Aktivitäts-Anzeige 32
Aktualisierung 44, → s. Update
Alias 31, 589, → s. Link,
 → s. Verknüpfung
Alpha-Kanal 189
Alpha-Wert 134
ALT_MASK 150
Alt-Taste 16
always 166
anonyme Klasse 126, 527
anonymes Paket 511
Ant 62, 77, 78, 385, 465, 468, 474
 ear 475
 Eclipse 394
 Filter 395
 jar 475
 JAR-Archiv 391
 jarbundler 396
 Manifest 391
 metainf 475
 Programmpaket erzeugen 394
 Property 389
 Umgebungsvariablen 389
 war 475
 Xcode 388
 Xcode 1.5 393
 XDoclet 477
ANT_HOME 386
AntiAliasedGraphicsOn 558
AntiAliasedTextOn 558
Antialiasing 556, 557, 589
Ant-Task 387
Anweisung 514
Apache 242, 257, 445, 448, 503

- Ant 385
 - Maven 396
- Apfel-Menü 27
- Apfel-Taste 16
- API 589
- API-Dokumentation 43, 62, 512
- app 187
- appBundlesTraversable 166
- AppKit 324, 336
- APPL???? 192
- Apple 70
 - Applet Runner 59, 535
 - Application Server 453
 - Application Servers-Paket 465
 - Developer Connection 46
 - Gestaltungsrichtlinien 171
 - Human Interface Guidelines 171
 - J2EE 479
 - OSXAdapter 164
 - WebObjects 464
- apple.awt.antialiasing 557
- apple.awt.brushMetalLook 168, 552
- apple.awt.fakefullscreen 555
- apple.awt.fileDialogForDirectories 138, 552
- apple.awt.fractionalmetrics 557
- apple.awt.fullscreencapturealldisplays 555
- apple.awt.fullscreenhidecursor 555
- apple.awt.fullscreenusefade 555
- apple.awt.graphics.EnableLazyDrawing 558
- apple.awt.graphics.EnableLazyDrawingQueueSize 558
- apple.awt.graphics.EnableLazyPixelConversion 563
- apple.awt.graphics.OptimizeShapes 558
- apple.awt.graphics.RenderDrawArc 562
- apple.awt.graphics.RenderDrawOval 562
- apple.awt.graphics.RenderDrawPolygon 562
- apple.awt.graphics.RenderDrawRect 562
- apple.awt.graphics.RenderDrawRoundRect 562
- apple.awt.graphics.RenderDrawShape 562
- apple.awt.graphics.RenderFillArc 562
- apple.awt.graphics.RenderFillOval 562
- apple.awt.graphics.RenderFillPolygon 562
- apple.awt.graphics.RenderFillRect 562
- apple.awt.graphics.RenderFillRoundRect 562
- apple.awt.graphics.RenderFillShape 562
- apple.awt.graphics.RenderGlyphs 563
- apple.awt.graphics.RenderImage 562
- apple.awt.graphics.RenderLine 562
- apple.awt.graphics.RenderString 563
- apple.awt.graphics.RenderUnicode 563
- apple.awt.graphics.UseTwoImageLazyPixelConversion 563
- apple.awt.interpolation 557
- apple.awt.rendering 557
- apple.awt.showGrowBox 168, 551
- apple.awt.textantialiasing 557
- apple.awt.window.position.forceSafeCreation 556
- apple.awt.window.position.forceSafeProgrammaticPositioning 556
- apple.awt.window.position.forceSafeUserPositioning 556
- apple.cmm.usecolorsync 563
- apple.laf.AquaLookAndFeel 144
- apple.laf.useScreenMenuBar 162, 181, 551
- AppleEvent-Deskriptor 341
- AppleJavaExtensions 163, 329
- AppleScript 340, 350
 - Cocoa Java 340
 - osascript 342
- APPLET 254
- Applet 59, 60, 64, 241, 250, 536
 - Cache 260
 - Lebenszyklus 253
 - Sicherheit 262
- Applet Launcher 61, 257
- Applet Runner 535
- AppleTalk 302
- AppletContext 271, 281

- appletviewer 61, 257, 264, 273
- Application 154, 203, 289
- Application Kit 336
- Application Support 68
- application.xml 473, 475
- ApplicationEvent 155
- ApplicationListener 154, 205
- Applications 31, 225
- Applications (Mac OS 9) 533
- Applikation 187
 - aktive 27
- Applikations-Server 453, 463, 464
- Aqua 31, 56, 144, 589
 - Gestaltungsrichtlinien und Layout-Manager 166
 - Größe von Komponenten 166
 - Hintergrund 134
 - Human Interface Guidelines 332
- Aqua-Look & Feel 142, 144
- Arbeitsverzeichnis 213
- ARCHIVE 254, 255, 261
- Archivnamen 288
- ArgoUML 417
- Argumentliste, variable 545
- Arguments 213, 216, 550
- Array 545
- ASCII 589
- ASP 445
- assert 570, 580
- assertTrue 399
- Assoziation 522
- Attribute 327
- attributorientiertes Programmieren 477
- Audio 366, 374
- Audiodaten 362
- Auflösungsunabhängigkeit 120
- Aufrufhierarchie 88
- Aufzählungen 545
- Ausdruck 515
- ausführen 87
- Ausführungsbit → s. Execution-Bit
- Ausnahmefehler 528
- Aussehen 142, 144
 - im Programmcode setzen 145
- Auswerfen 221
- Authentisierung 463
- Autoboxing 544

- Auto-Commit 431
- automatische Variable 515
- Autorisation 348, 463
- Autorisation Toolkit 348
- Autounboxing 545
- AVI 362
- awakeFromNib() 337
- AWT 119, 121, 251, 291, 589
 - Anpassung an andere Systeme 128
 - Brushed Metal 167
 - Cocoa 339
 - Xcode 122

B

- backgroundOnly 171, 553
- Back-in-Time-Debugging 106
- Backslash 590
- bash 39
- Basisklasse 520
- batch 576
- Batch-Datei 177
- BBEdit 36, 177
- Bea WebLogic 464
- Bean 463
- Bean Managed Persistence 468
- Bearbeiten-Menü 28
- beenden 176
- Beenden-Menüpunkt 132
- Befehl-Taste 16
- Benutzer 31, 432
 - Passwort ändern 433
- Benutzer-Prozesse 65
- Benutzerverwaltung 26, 347, 433
- Benutzer-Verzeichnis 37, → s. Home-Verzeichnis
- Benutzungsoberfläche 25, 119, 551
- Berechtigung 263
- Berkeley DB 441
- Betriebssystem 48, 297
- Bezeichner 510
- bg 435
- Bibliothek 51, 66, 182, 318
 - Abhängigkeiten 319
 - Installationspfad 321
 - Suchreihenfolge 68
- bicubic 557
- Big Endian 548
- BigDecimal 516

- BigInteger 516
- Bildbearbeitung 188, 353
- Bildlaufleiste 159
- bilinear 557
- bin 30, 188
- Bitmap 589
- Block 509
- BlueJ 113
- Bluetooth 497
 - MIDP 500
- Bluetooth Assistent 498
- Bluetooth Datenaustausch 498
- BMP 354, 468, 589
- Boolean 161
- bootclasspath 490, 568
- Booten 31
- Boot-Manager 31, 531
- bootstrap 543
- BorderLayout 120, 125, 274
- Borland
 - JBuilder 107
 - Together Control Center 108
 - Together Solo 108
- Bourne-Shell 177
- Boxing 544
- BoxLayout 120
- BranchGroup 356
- Breakpoint 88, 102
- BrowseListener 303
- Browser 60, 64, 234, 250, 255, 265, 299, 324, 325
 - AWT-Komponente 339
 - Java 1.4 266
- Browser → s. Web-Browser
- Browser Launcher 300
- Browser Opener 300
- BrowserControl 301
- Brushed Metal 167
- brushMetalLook 552
- BSD 56
- Buch-CD 595
- BuchUpdate 47
- Buch-Webseite 596
- BufferedImage 563
- Bug Parade 292
- Bug Reporter 548
- Build 86, 212, 391
 - Einstellungen 91
 - Phasen 92, 313, 491
 - Protokoll 86, 494
 - Prozess 385, 577
 - Settings 91
 - System 90
 - transcript 86
 - Variablen 92
- Build Settings 577
- Build Transcript 494
- build.xml 386, 474
- BUILD_DIR 578
- buildDMG 222
- Buildfile 386, 474
- BUILT_PRODUCTS_DIR 578
- Bundle 187, 219, 319, 394, 582, 589
 - im Dateiauswahldialog 138
- Bundle Resources 92
- Bundle-Bit 207
- Business Logic 470
- Button 120, 125
- Bytecode 22, 38, 403, 404, 414, 512, 589
 - Viewer 406

C

- C, C++ 24, 309, 519, 521
- Cache 238, 260
- CAD 359
- call 276
- CallStaticVoidMethod() 323
- Camino 259, 268
- Canvas 506
- Canvas3D 356
- Carbon 57, 589
 - Locking 326
- Carbon File Locking 573
- CarbonLock 326
- Carriage Return 294, 548
- CASE 108
- Cast 516, 589
- CATALINA_HOME 449
- catch 528
- cc 318
- CD 595
- ceil() 516
- CENTER 125, 129
- Certificate Authority 247
- CFBundleAllowMixedLocalizations 210

CFBundleDevelopmentRegion 210
 CFBundleDisplayName 210, 219
 CFBundleDocumentTypes 216
 CFBundleExecutable 210
 CFBundleGetInfoHTML 210, 219
 CFBundleGetInfoString 210, 219
 CFBundleIconFile 211
 CFBundleIdentifier 211
 CFBundleInfoDictionaryVersion 211
 CFBundleName 130, 211, 219, 552
 CFBundlePackageType 192, 211
 CFBundleShortVersionString 211, 219
 CFBundleSignature 192, 211
 CFBundleTypeExtensions 217
 CFBundleTypeIconFile 217
 CFBundleTypeMIMETypes 217
 CFBundleTypeName 217
 CFBundleTypeOSTypes 217
 CFBundleTypeRole 217
 CFBundleVersion 212
 CGI 445, 589
 check, jni 569
 Checkbox 120, 125
 CheckboxGroup 125
 checkSpelling() 346
 chmod 178, 207
 chown 438
 CHUD-Tools 409
 CTime 576
 Class 140
 Class.forName() 140, 288, 289, 291,
 323, 349
 CLASS_ARCHIVE 581
 CLASS_ARCHIVE_SUFFIX 581
 CLASS_FILE_DIR 491, 581
 CLASS_FILE_LIST 582
 ClassCastException 153
 class-Datei 39, 509
 Classes 63, 475
 classes.jar 288, 405, 413
 Classic 324, 589
 Speicherprobleme 537
 Umgebung 24, 533
 ClassLoader 244, 289
 ClassNotFoundException 289
 CLASSPATH 51, 66, 68, 176, 180, 212,
 216, 413, 550, 567, → s. cp
 CLDC 483
 Clean 390
 Client-VM 566
 close() 428, 429, 519
 cmd 298
 CMM 563
 CMP 468, 589
 Cocoa 57, 330, 349, 589
 Action 334, 337
 API 336, 339
 Applikation Kit 336
 AWT 339
 Foundation 336
 id 334
 Java und Objective-C 339
 NSObject 333
 Outlet 334
 Swing 339
 Cocoa Java 300, 330
 AppleScript 340
 Bridge 339
 Referenzen 335
 Speech 344
 Spelling 345
 Xcode 331
 Cocoa Java, Bridge 330
 CocoaComponent 339
 CocoaMySQL 437
 Cocoon 299
 CODE 255
 Code Completion 83
 Code Conventions 530
 Code Sense 83
 CODEBASE 239, 255, 261
 Codec 354
 CodeGuide 106
 Code-Optimierung 405
 Codepage 295
 CodeWarrior 110
 Collections API 22
 Color 134, 523
 ColorSync 563
 com.apple.audio 375
 com.apple.backgroundOnly 171, 553
 com.apple.cocoa.
 application.NSMenu 555
 com.apple.dnssd 303
 com.apple.eawt 146, 154
 com.apple.eio 329

- com.apple.forcehwaccel 561
- com.apple.hwaccel 560
- com.apple.hwacceleexclude 560
- com.apple.hwaccelist 560
- com.apple.hwaccelnogrowbox 561
- com.apple.macos.smallTabs 553
- com.apple.macos.use-file-dialog-
packages 137, 553
- com.apple.macos.useMDEF 553
- com.apple.macos.useScreenMenuBar
162, 553
- com.apple.macosx.AntiAliased-
GraphicsOn 558
- com.apple.macosx.AntiAliased-
TextOn 558
- com.apple.mrj 121, 122, 328
auf anderen Systemen 139
- com.apple.mrj.appli-
cation.apple.menu.about.name
130, 181, 552
- com.apple.mrj.application.classpath
216, 550
- com.apple.mrj.appli-
cation.growbox.intrudes 552
- com.apple.mrj.appli-
cation.JVMVersion 216, 551
- com.apple.mrj.application.live-resize
553
- com.apple.mrj.application.main 216,
550
- com.apple.mrj.application.para-
meters 216, 550
- com.apple.mrj.appli-
cation.vm.options 216, 551
- com.apple.mrj.application.
workingdirectory 216, 550
- com.apple.mrj.MRJAboutHandler 552
- com.apple.mrj.swing.MacLoo-
kAndFeel 144
- com.apple.usedebugkeys 561
- com.muchsoft.util.Mac 139, 163
- com.sun.j3d.utils 356
- com.sun.jimi.core 355
- com.sun.media.jai.codec 354
- Combo Box 166
- Command 503
- command 178
- CommandListener 503
- Commands 63
- Command-Taste 16
- commit() 430
- Communicator 267
- Compiler 38, 412, 512, 589
- CompileThreshold 576
- Component 120
- Concurrent Versions System 94
- Connection 335, 428, 429, 430
- Connection-URL 428
- Connector/J 437
- Console 41
- Constructor 141
- Container 120
- Container Managed Persistence 468
- Content Pane 159
- Contents 190, 582
- context-root 473
- Controller 467
- Copy Files 93
Programm 313
- Copyright 198, 211
Zeichen 129
- CORBA 479, 589
- CoreAudio 375
- CoreFoundation 316
- CoreGraphics 559
- cp 180, 491
- Cp1252 296
- CR 294
- Crash-Log 41
- create() 470, 472
- createCompatibleImage() 355
- createdb 438
- CreateException 469
- createImage() 374
- createStatement() 428, 430
- Creator-Code 191, 199, 211, 296, 327
- Cross-Platform 305
- CTRL_MASK 150
- Ctrl-Taste 16
- Current 64
- CurrentJDK 63
- currentTimeMillis() 301
- Custom Build Command 389
- CustomView 336
- CVS 94, 593
aktivieren 96

- checkout 95
- commit 95
- Compare 97
- Diff 97
- import 95
- init 94
- Projekt importieren 95
- CVSROOT 94
- cvs wrappers 94
- Cyberduck 244

D

- da 571
- Darstellungshinweis 556
- Darwin 56, 70, 318, 590
- Data Fork 188
- Database 423
- DataFlavor 169
- DataRef 371
 - file-URLs 372
- DateFormat 446
- Datei
 - Attribute 327
 - Information 198, 210
 - Metadaten 327
 - umbenennen 179
- Dateiauswahldialog 552, 553
 - AWT 136
 - Bundles / Pakete 138, 166
 - Swing 165
 - Verzeichnisse auswählen 138
- Dateiname 297, 537
 - Länge 294
 - Sonderzeichen 294
- Dateinamenerweiterung 328, 593
- Dateisystem 29, 69
- Dateitrennzeichen 293
- Dateiverwaltung 347
- Datenbank 423
 - Treiber laden 427
 - Verbindung abbauen 430
 - Verbindung aufbauen 428
- Datenbankentwurf 426
- Datenbank-System 424
- Datenbank-Verwaltungs-System 423
- Datenbasis 423
- Datensatz 424
- Datentyp 515

- DB 423, 590
- DBMS 424, 590
- dclj 530
- de.comp.lang.java 530
- debug 569
- Debugger 88, 102
- Debugging 88
 - Back-in-Time 106
 - Grafikausgabe 562
- Debug-Information 409
- DecimalFormat 514
- Decompiler 403, 404
- Decoration 134
- DefaultClasspath 68
- Deinstallieren 187
- Deklaration 515
- Deployment 476
- DEPLOYMENT_LOCATION 578
- Deployment-Deskriptor 458, 471, 473, 477
- deprecated 328, 370
- Desktop 29, 590
- destroy 253
- DestroyJavaVM() 324
- Destruktor 519
- Developer Tools 42, 76
- DEVELOPMENT_PLIST_FILE 582
- Dialog 120
 - Elemente 28
 - zentrieren 129
- Dictionary 209
 - Java 212, 237
- Dienst 450
- Dimension 129
- disableassertions 571
- DisableExplicitGC 575
- disablesystemassertions 572
- Disassembler 406
- Disk Copy 220
- Disk Image 220, 231
- Display 490
- display-name 471, 473
- displayURL() 300
- dispose() 124, 519
- DLL 311
- DMG 220, 590
 - Hintergrundbild 222
 - Nur lesen 223

- Segmente 42
- DMG Maker 222
- DMG Tool 222
- dmgpart 42
- DnD 169, 590
- DnDConstants 169
- DNS 303, 590
- dns_sd.jar 303
- DNSSD 303
- DNSSDService 303
- do..while 517
- Dock 26, 535, 590
- dock
 - icon 572
 - name 572
 - Symbol 180
 - Symbol anklicken 156
- Document Object Model 279
- DocumentListener 159
- doGet() 446, 472
- Dokumentation 43, 46, 410, 512
- Dokumentsymbol 217
- Dokumenttyp 199, 204, 209, 216
- Dokumenttypen 131
- DOM 279
- Dongle 404
- doPost() 456
- Doppelpunkt 69
- DOS 293, 295
- Drag & Drop 29, 51, 169, 220, 590
- DragSourceDragEvent 170
- Drawable 367
- Drei-Schicht-Architektur 463
- Driver/Manager 428, 430
- DrJava 115
- Druckdialog 299, 377
- Drucken 375
- dsa 572
- DSTROOT 579
- DTD 214
- Duke 535
- dylD 320
- DYLD_FALLBACK_FRAMEWORK_PATH 320
- DYLD_FALLBACK_LIBRARY_PATH 320
- DYLD_LIBRARY_PATH 320

- Dylib 318, 319, 326, 590
 - Suchpfade 320
- Dynamic Shared Library 318
- dynamischer Methodenaufruf 524
- dynamisches Laden von Klassen 289

E

- ea 571
- EAR 385, 474
- ear 475, 581
- Early Access 541
- EAST 125
- Eclipse 98, 205
 - Ant 394
 - Debugger 102
 - Eingabevollständigung 100
 - Fehler 100
 - JUnit 402
 - kompilieren 100
 - SDK 98
 - Versionen 2.1 und 3.0 99
 - XML-Plugin 417
- Eden-Generation 566, 574
- Editor 204, 217
- einfacher Datentyp 515
- Eingabeaufforderung 38
- Eingabevollständigung 83, 100
- Einstellungen...-Menüpunkt 52, 131, 155, 164
- eio 329
- EJB 78, 463, 467, 479, 590
 - Container 463, 468, 471
 - Deployment-Deskriptor 471
 - Komponente 463, 477
- ejb 473
- ejbActivate() 470
- ejb-class 471
- ejbCreate() 470
- EJBHome 470
- ejb-jar.xml 471, 473, 475
- ejb-name 471, 473
- EJBObject 469
- ejbPassivate() 470
- ejbRemove() 470
- Element 254
- else 516
- emacs 37

- E-Mail 295
- EMBED 255
- Embedding Framework 325
- emulator 494
- enableassertions 571
- EnableLazyDrawing 558
- EnableLazyDrawingQueueSize 558
- EnableLazyPixelConversion 563
- enablesystemassertions 572
- Ende-Tag 209, 254
- Endorsed Directory 568
- English.lproj 192, 218
- Enterprise JavaBeans 463, 467
- Enterprise-Applikation 78, 468
- Enterprise-Applikations-Archiv 474
- Enterprise-Applikations-Deployment-Deskriptor 473
- Entity-Bean 467
- Entwicklerpaket 591
- Entwicklertools 42
- Entwicklungsumgebung 75
- Entwurfsmuster 591
- Enumeration 152, 545
- environment.plist 65
- Environment-Variable 94
- erben 520
- Ereignis-Behandlung 120
- Ereignis-Verarbeitungs-Thread 157, 274
- Ergebnistabelle 426, 429
- Erweiterungs-Bibliotheken, Suchreihenfolge 68
- Erweiterungsverzeichnisse 66
- esa 572
- Escape-Sequenz 514, 590
- etc 30, 53
- Ethernet-Hardware-Adresse 310
- Euro-Symbol 590
- eval 277
- Event Handling 120
- EventObject 164
- EventQueue 291
- examples 595
- Exception 528
- ExceptionDescribe() 324
- ExceptionOccurred() 324
- exec() 297, 342
- execPrivileged() 349

- Executable 80
 - ausführen 87
- executeQuery() 428
- executeUpdate() 430
- Execution-Bit 178
- Expert View 92, 577
- Explorer 27
- Exposé 30, 590
- Expression 515
- ext 67
- extends 122, 520, 526
- Extension 593
- Extensions 66, 131, 401
- extern C 315
- External Target 90, 389
- Extreme Programming 398

F

- fail 400
- fakefullscreen 555
- Fallunterscheidung 516
- FALSE 161
- FAQ 46
- Farbverwaltung 563
- Feld 424, 429
- Fenster 120, 535
 - Änderungssymbol 160
 - Hintergrund 134
 - unsichtbar 155
 - zentrieren 129
- Fenster-Menü 29, 148
- Fensterrand 129
- Festplatte 31
- Festplatten-Dienstprogramm 32, 220
- Festschreiben 431
- File 293, 297
 - pathSeparator 294
 - pathSeparatorChar 294
 - separator 293, 294
 - separatorChar 294
- File Transfer Protocol 244
- file.encoding 296, 549
- file.separator 294, 549
- FileDialog 136, 137, 165, 552, 553
- FileManager 296, 300, 329, 330
- FileMerge 97
- FilenameFilter 138

- FileStorm 231
- FileType 199
- Filme 366
- Filter 138
- final 522
- finally 528
- FindClass() 323
- Finder 27, 590
- findFirstMisspelledWord() 347
- findFolder() 330
- findMisspelledWordInString() 347
- Firebird 439
- Firefox 259, 269
- Fix and Continue 89
- floor() 516
- FlowLayout 120, 125, 253
- for 517, 544
- for each 544
- forcehwaccel 561
- forceSafeCreation 556
- forceSafeProgrammaticPositioning 556
- forceSafeUserPositioning 556
- FORM 456
- Form 488
- formale Parameter 513
- forName() 140, 289
- Foundation (Cocoa) 336
- Foundation Classes 119
- fractionalmetrics 557
- Frame 120, 122
 - Rahmen 134
- Frame.ICONIFIED 128, 153
- Frame.NORMAL 154
- Framework 57, 62, 316, 590
- Frameworks & Libraries 92
- FreeBSD 56, 302
- freie Software 590
- Fremdschlüssel 425
- FTP 244
- Full-Screen Exclusive Mode 554
- fullscreencapturealldisplays 555
- fullscreenhidecursor 555
- fullscreenusefade 555
- FullScreenWindow 556
- fullversion 568
- Funktion 513
- future 568

G

- G3, G4, G5 548
- Garbage Collection 326, 519, 566, 574
- gcc 312, 318, 412
- gcc3 318
- GDB 89
- gebürstetes Metall 167
- GeekTool 42
- Generics 24, 544
- GenericServlet 446
- Generische Typen 544
- German.lproj 218
- Geschäftslogik 467, 470
- Geschwindigkeit 13
- Gestaltungsrichtlinien 171
- GET 456
- getAppletContext 273
- getBundle() 157
- getBytes() 296
- getConnection() 428, 430
- getConstructor() 141, 290
- getContentPane 274
- getContentPane() 159
- getCrossPlatformLookAndFeelClassName() 145
- getDefault() 157
- getDefaultToolkit() 149, 374, 376
- getDirectory() 137
- getDouble() 428
- getExtendedState() 153
- getFile() 137
- getFileCreator() 328
- GetFileInfo 199, 327
- getFileType() 328
- getGraphics() 376
- getInsets() 129
- getKeyStroke() 149
- getMember 277
- getMenuComponentCount() 152
- getMenuShortcutKeyMask() 149
- getMethod() 289
- getMouseLocationOnScreen() 289
- getParameter 255
- getParameter() 456
- getPrinterJob() 379
- getPrintJob() 376
- getProperty 547
- getRequestURI() 456

- getRootPane() 134, 160
- getScreenSize() 129
- getSize() 129
- getSource() 153
- GetStaticMethodID() 323
- getString() 148, 428
- getSystemLookAndFeelClassName() 145
- Getter 519
- getTitle() 153
- getToolkit() 129, 376
- getUserAction() 170
- getWindow 276
- GIF 189, 354, 492
- GL4Java 359
- GLCanvas 359
- glconfigurations.properties 560
- globaler Code 510
- GNU 590
 - GPL 300
- gnumake 310
- Goal 397
- Gosling, James 11
- GPL 300, 590
- Grafikausgabe 353
- Grafik-Hardware-Beschleunigung 196, 558
- Grafikkarte 559
- Grafik-Primitiv 557
- grafische Benutzeroberfläche 119
- grant 263
- Graphic User Interface → s. GUI
- GraphicConverter 188, 327
- Graphics 253, 353, 376, 380
 - dispose() 376
- Graphics2D 253, 353, 380
- GraphicsEnvironment 355
- GraphicsExporter 374
- GraphicsImporter 374
- GraphicsImporterDrawer 374
- GridLayout 125
- Groovy 414
 - groovy 415
 - Groovy, Compiler 415
 - GROOVY_HOME 414
 - groovyc 415
 - groovyConsole 415
 - groovysh 414

- Growbox 551, 552
- Grundlagen 509
- Gruppenverwaltung 347
- GUI 25, 48, 119, 340, 551, 590

H

- Häkchen vor einem Menüpunkt 153
- Handheld 483, 500
- handleAbout() 130, 140, 155, 164
- handleOpenApplication() 131, 140, 156, 164
- handleOpenFile() 131, 132, 140, 156, 164, 205, 218
- handlePreferences() 156
- handlePrefs() 130, 140, 164
- handlePrintFile() 140, 156, 164
- handleQuit() 132, 133, 140, 156, 164
- handleReOpenApplication() 156, 164
- Handy 483, 497
- Hardware-Beschleunigung 196, 558
- Hardware-Kopierschutz 404
- Hardware-Zugriff 309
- Hauptklasse 512
- Hauptversion 195, 259
- hdiutil 222
- Header 312, 315
- Headers (Verzeichnis) 325
- Headless AWT 553
- HeadlessException 170
- Headless-Modus 170
- Heap 573
- HFS+ 69, 297, 590
- Hilfe 29, 33, 42
- Hintergrund 134
- Hintergrundbild 222
- HiRes Timer 301
- Hit Mask 189
- Hochauflösende Zeitmessung 301
- Home 37, 53, 63, 543
- home 471
- Home-Interface 470
- HORIZONTAL_SCROLLBAR_ALWAYS 159
- Host 590
- hostconfig 434, 439
- Hot Deployment 476
- HotSpot 22, 566
- Hotsync Manager 500

- hqx 188
- HSQLDB 440
- HTML 234, 241, 250, 253, 265, 274, 278,
 - 445, 453, 590
 - anzeigen 299
 - Entity 296
 - in Swing-Komponenten 161
 - Validator 271
- HTML Converter 256
- HTTP 446, 456, 591
- httpd 242
- HTTP-Proxy 259
- HttpServlet 446, 472
- HttpServletRequest 446
- HttpServletResponse 446
- HTTPS-Proxy 259
- Hüllklasse 276, 311, 330, 343
- Hüllobjekt 544
- Human Interface Guidelines 31, 48,
 - 171, 332
- Humble Narrator 344
- hwaccel 560
- hwaccel_info_tool 560
- hwaccelextclude 560
- hwaccelextclude.properties 561
- hwacclist 560
- hwaccelnogrowbox 561
- Hypersonic SQL 440

I

- IB 331, 591
- IBAction 336
- IBOutlet 336
- iCab 270, 281
- ICC_ColorSpace 563
- icns Browser 189
- icns-Datei 180, 188
- Icon Composer 188
- Icon-Datei 28, 188, 191, 207, 217
- ICONIFIED 128, 153
- id 334
- ID → s. Schlüssel
- IDE 75, 591
- IDEA 105
- Identifier 200, 211
- Identifikation 348
- if 516
- IllegalStateException 132
- Im Dock ablegen 128
- Image 374
- Image I/O 353
- Immortal-Generation 576
- implements 122, 525
- import 288, 511
- import static 545
- incgc 575
- Index 83
- Index Templates 84
- Info.plist 130, 171, 191, 207, 208, 214,
 - 216, 237, 339, 395, 550, 565, 582
- INFO_PLIST_FILE 582
- InfoPlist.strings 192, 219
- Information Property List 208
- Inhalts-Ebene 159
- init 252
- initdb 438
- InitialContext 473
- Inkrement 517
- innere Klasse 124, 527
- Input Files 93
- InputEvent 149
- InputStreamReader 296
- INSERT INTO 430
- Insets 129
- INSTALL_DIR 579
- INSTALL_GROUP 579
- INSTALL_MODE_FLAG 579
- install_name_tool 321
- INSTALL_OWNER 579
- INSTALL_PATH 579
- install_templates 84
- install4j 229
- InstallAnywhere 227
- Installationspaket 166, 223, 592
- Installationsprogramm 192, 224
- InstallerMaker 232
- InstallShield 228
- int 575
- Integrated Development Environment
 - s. IDE
- IntelliJ IDEA 105
- InterBase 439
- Interface 123, 290, 525
- interface 525
- Interface Builder 331, 340
 - Action 336

- Connection 335
- Klassen ableiten 333
- Objekte erzeugen 335
- Outlet 335
- Quelltext generieren 336
- Swing 340
- Target/Action 336
- internalversion 569
- Internet Explorer 267
- Internet-Protokolle 295
- interpolation 557
- Interpreter 22
- invoke() 289
- invokeAndWait() 274, 291
- invokeLater() 133, 274, 291
- Invoker-Servlet 452
- iODBC 441
- IOKit 316
- IP 590, 591
- IPv4 445
- IPv6 445
- isPopupTrigger() 135
- iText 299
- IzPack 233

J

- J/Direct 324
- J2EE 78, 447, 463, 591
 - Apple 479
 - Applikations-Server 463
 - SDK 463
 - Server 463
- J2ME 483, 591
 - Konfiguration 483
 - MIDlets 485
 - MIDP 484
 - Profil 484
 - System-Properties 488
 - Tipps 508
 - Wireless Toolkit 485
- J2SE 447, 591
 - Version 5.0 24, 541
- j2se 240
- jad 403
- JAD-Datei 493, 502
- Jaguar 23, 591
- JAI 24, 353
 - Image I/O Tools 354
- jai_imageio.jar 354
- Jakarta Struts 459
- Jakarta Tomcat 448
- Jam 90
- JApplet 272
- jar 182, 475, 567
 - Option 185
- Jar Bundler 62, 193, 214
- Jar Bundler Ant Task 396
- JAR Caching 260
- Jar Launcher 185
- JAR-Archiv 51, 67, 182, 197, 207, 241, 244, 288, 385, 491
 - in Xcode 186
 - per Doppelklick starten 184
 - statisch einbinden 92
- jarsigner 247
- Java 57
 - 2D 353, 378, 380
 - 3D 24, 356, 381
 - Advanced Imaging 24, 353
 - API-Dokumentation 410, 512
 - Application Stub 191, 582
 - Archive Settings 92
 - Bytecode 22, 38
 - Code Conventions 530
 - Collection Framework 544
 - Communications API 301
 - Compiler 38, 412, 512
 - Compiler Settings 91
 - Data Objects 423, 591
 - Debugger 89
 - Developer Package 312
 - Developer Tools 43, 76
 - Development Kit 23
 - Einführung 530
 - Embedding API 536
 - Embedding Framework 64, 258, 325
 - Embedding Plugin 259
 - Enterprise Edition 463
 - Erweiterungen 66
 - Erweiterungsverzeichnis 401
 - Foundation Classes 119
 - Grundlagen 509
 - Home 65
 - Image I/O 354
 - Image Management Interface 354
 - Kompatibilität 292

- Konsole 258
- Language Specification 412
- Laufzeitparameter 259
- Laufzeitumgebung 321, 513
- Look & Feel 142, 146
- Media Framework 362, 374
- Micro Edition 483
- Native Interface 309
- Network Launching Protocol 234
- Plugin 60, 255, 258, 266, 277, 325, 536
- Plugin Einstellungen 259
- Preferences API 54
- Programmierrichtlinien 510, 530
- Programmiersprache 22
- Quelltext 411
- QuickTime 366
- Reflection 139, 163
- Resources 92
- Runtime Environment 23, 176
- Runtime-Version 586
- Scripting 280
- Server Faces 104
- Servlets 445, 446
- Shared Archive 566, 576
- Sound 374
- Speech Framework 343
- Spelling Framework 345
- Studio Creator 104
- User Group 70
- Version 63, 194, 292, 551, 585, 586
- Version 1.1 91
- Version 1.5 24, 541
- Version der Apple-Implementierung 586
- Versionsproblem 407
- Virtual Machine 22, 64, 321, 536, 565
- VM 22, 64
- VM-Framework 63
- Web Start 60, 112, 234
- java 39, 63, 176, 263, 513, 543, 565
- Java 1.3.1 Plugin Einstellungen 259
- Java 1.4.2 Plugin Einstellungen 259
- Java 2 Plattform 22
- Java Applet Plugin Enabler 268
- Java Applet.plugin 266
- Java Web Start 236
- java.applet 251
- java.awt 121, 375
- java.awt.color.ICC_ColorSpace 563
- java.awt.datatransfer 169
- java.awt.dnd 169
- java.awt.event 124
- java.awt.FileDialog 552
- java.awt.headless 171
- java.awt.image.BufferedImage 563
- java.awt.print 375, 378
- java.awt.RenderingHints 557
- java.class.path 288
- java.endorsed.dirs 568
- java.ext.dirs 68
- java.home 66
- java.io.File 297
- java.io.FileNameFilter 138
- java.io.Serializable 169
- java.lang.reflect 141
- java.library.path 294, 318
- java.math 516
- java.net.preferIPv4Stack 445
- java.policy 263
- java.rmi 469
- java.runtime.version 548, 586
- java.sql 426
- java.text 514
- java.vendor 548
- java.vendor.url 548
- java.vendor.url.bug 548
- java.version 547, 585
- java.vm.vendor 548
- java.vm.version 547
- JAVA_APP_STUB 582
- JAVA_ARCHIVE_CLASSES 581
- JAVA_ARCHIVE_COMPRESSION 581
- JAVA_ARCHIVE_TYPE 581
- JAVA_CLASS_SEARCH_PATHS 581
- JAVA_COMPILER 580
- JAVA_COMPILER_DEBUGGING_SYMBOLS 580
- JAVA_COMPILER_DEPRECATED_WARNINGS 580
- JAVA_COMPILER_DISABLE_WARNINGS 580
- JAVA_COMPILER_FLAGS 580
- JAVA_COMPILER_SOURCE_VERSION 580

JAVA_COMPILER_TARGET_VM_VERSION 580
 JAVA_FORCE_FILE_LIST 582
 JAVA_HOME 65, 414, 543
 JAVA_LAUNCHER_VERBOSE 207
 JAVA_MANIFEST_FILE 581
 JAVA_SOURCE_FILE_ENCODING 580
 JAVA_SOURCE_PATH 582
 JAVA_SOURCE_SUBDIR 582
 JAVA_USE_DEPENDENCIES 582
 Java13Adapter 140
 Java13Handler 139, 140
 Java14Adapter 163
 Java14Handler 163
 Java2D 556
 JavaApplet.h 325
 JavaApplicationStub 206, 210, 395
 JavaBrowser 62, 410
 javac 38, 63, 370, 512, 543, 580
 JAVAC_DEFAULT_FLAGS 580
 JavaClasses 581
 JavaComm 301
 javaconfig 68
 JavaConfig.plist 68
 JavaConnect 281
 JavaControl.h 325
 Javadoc 410, 477
 JavaEmbedding.h 325
 javah 310, 312
 JavaMonitorsInStackTrace 569
 javap 406
 JavaPluginCocoa.bundle 266
 JavaScript 242, 256, 274
 globales Objekt 276
 window 276
 JavaScriptCore 266
 JavaScript-URL 281
 JavaServer Pages 445, 453
 JavaSound 374
 Audio-Eingabe 375
 JavaSpeechFramework.jar 343
 JavaSpellingFramework.jar 345
 JavaVM 322
 JavaVM.framework 312
 JavaVMInitArgs 322
 JavaVMOption 322
 javaw.exe 186
 javax.ejb 469
 javax.imageio 354
 javax.jnlp 248
 javax.media 362
 javax.media.j3d 356
 javax.media.jai 353
 javax.microedition 485
 javax.naming 472
 javax.print 375
 javax.rmi 472
 javax.servlet 446
 javax.servlet.http 446
 javax.sound 375
 javax.sql 426
 javax.swing 147, 251, 272
 javax.swing.plaf.metal.
 MetalLookAndFeel 144
 JAWS 234, 591
 Desktop-Integration 236
 Programm-Manager 235
 Programmname 239
 Sicherheit 245
 JayBird 440
 JBindery 537, 539
 JBoss 78, 464
 beenden 467
 server/default/deploy 476
 starten 465
 Workbook 478
 jboss-j2ee.jar 474, 475
 JBuilder 107
 Versionen 6,7, X und 2005 107
 Versionen 8 und 9 108
 JCavaj 403
 JCheckBoxMenuItem 153
 jclasslib 407
 JComboBox 167
 JComponent 120
 JDBC 423, 426, 428, 591
 Adresse 428
 Treiber 427, 437, 439
 URL 428
 JDBC-ODBC-Bridge 441
 JdbcOdbcDriver 441
 JDesktopPane 143
 JDeveloper 108
 JDialog 120
 Menüzeile 163

- JDirect 324, 325, 349, 536
 - Version 2 und 3 326
- JDirect_MacOSX 325
- JDK 23, 57, 58, 531, 591
- JDKClasses.zip 91
- JDO 423, 591
- jEdit 111
- Jelly 396
- Jetty 445, 465
- Jext 112
- JFC 119, 537
- JFileChooser 165
- JFileChooser.appBundlesTraversable 166
- JFileChooser.packagelsTraversable 166
- JFrame 120, 147
 - Rahmen 134
- JGoodies 171
- JIO 354
- Jikes 412, 580, 581
 - Xcode 413
- JIKES_DEFAULT_FLAGS 581
- JKESPATH 413
- JIMI 354
- JInternalFrame 143
- JIT-Compiler 22, 575
- JJEdit 113
- JLabel 161
- JManager 324, 325, 536
- JmDNS 305
- JMenu 147, 148
- JMenuBar 148
- JMenuItem 147, 149, 153
- JMF 362
- jmf.jar 362
- JMFRegistry 362
- JMX 465
- JNDI 591
- JNI 309, 339, 349, 536, 569
 - Abhängigkeiten von Bibliotheken 319
 - Bibliothek 67
 - Bundle 319
 - Mac OS X 10.0 319
 - Pfad für Bibliotheken 317
 - Suchpfade 317
 - Xcode 310
- JNI, Bibliothek 246
- jni.h 312, 316, 322
- JNI_CreateJavaVM() 322
- JNI_GetDefaultJavaVMInitArgs() 322
- JNI_VERSION_1_2 322
- JNI_VERSION_1_4 322
- JNIDirect 326
- JNIEnv 316, 322
- jnilib 311
- JNLP 234
 - Developer's Pack 248
- jnlp.jar 248
- JNLP-API 248
- JNLP-Datei 238
- JobAttributes 375
- JOGL 359
- jogl.jar 359
- JOptionPane 120, 133
- JPanel 120
- JPEG 354
- JPEG2000 354
- JProfiler 408
- JRE 23, 58, 176, 249, 260, 591
- JRendezvous 305
- JRootPane 134
- JRun 464
- JSA 566, 576
- JScrollPane 159
- JSException 276
- JObject 275
 - call 276
 - eval 277
 - getMember 277
 - getWindow 276
- JSP 445, 453
 - Ausdruck 454
 - Direktive 454
 - Element-Bibliothek 455
 - Expression 454
 - Scriptlet 454
 - Standard Tag Library 455
 - Tag Library 455
- JSP-Container 445, 454
- JSR 541
- JSTL 455
- JTA 591
- JTabbedPane 166
- JTextArea 159
- JTS 591

JTxtCmpontDrvr 345
JUG 70
JUnit 398
 Eclipse 402
 Xcode 401
Just-in-time-Compiler 22
JVM 22, 38, 64, 321, 536
 Optionen 565
 Version 194
JVMVersion 212, 216, 551
JWS 60

K

Kapselung 519
Karteireiter 166, 553
KDE 266
Kennung 559
Kernel 591
KEY_ANTIALIASING 557
KEY_FRACTIONALMETRICS 557
KEY_INTERPOLATION 557
KEY_RENDERING 557
KEY_TEXT_ANTIALIASING 557
Key4J 404
KeyEvent 127, 149
Keystore 246
KeyStroke 149
keytool 246
KHTML 266
Kilobyte Virtual Machine 484
KJS 266
Klasse 275, 509
 abstrakte 524
 anonyme 126, 527
 dynamisch laden 289
 innere 124, 527
 lokale 527
 Typ 289
Klassenbibliothek 22
Klassendatei 39, 512
Klassendatei → s. class-Datei
Klassendiagramm 416
Klassenhierarchie 62
Klassenlader 139, 289
Klassenmethode 513
Klassenname 510
 voll qualifiziert 176

Klassenpfad 51, 66, 68, 92, 176, 197,
 212, 244, 288, 387, 401, 413, 567,
 → s. Classpath
KodakCMM 563
Kodierung 209, 549
Kommandozeile 38, 176
Kommandozeilenparameter 514
Kommentar 510
Kompatibilität 292
Komponente 120
Komponentenmodell 463
Konfiguration 483
Konfigurationsdatei 52
Konsole 41, 87, 185, 258, → s. Console
Konstante 522
Konstruktor 519
Konstruktorverkettung 521
Kontakt 17
Kontext 474
Kontext-Klick 136
Kontext-Popup-Menü 80, 135
Kontrollfelder 532
Kontrollkästchen 120
kooperativ 537
Kopierschutz 404
kTemporaryFolderType 330
KVM 484, 490

L

L&F 142
Label 129
LAF 142, 591
Landeskennung 157
Länge von Dateinamen 294
Laufwerk 29
 virtuelles 220
Laufzeitfehler 88
Laufzeitparameter 259
Laufzeitumgebung 22, 321, 513
Laufzeitverhalten 408
LaunchServices 202
Layout-Manager 120, 253, 274
 Positionsangabe 126
ld 318
learnWord() 347
Legacy-Target 90
LF 294
LGPL 590

- lib 67, 311
- LIBRARY_STYLE 319
- line.separator 294, 548
- LineBreak 177
- Linefeed 294, 548
- Link 31, 591
- LINKED_CLASS_ARCHIVES 581
- Linker 318, 325
- Linkliste 14
- Linux 23, 24, 55, 58, 302
- Listener 121, 124
- LiteSwitch 28
- LiveConnect 274
 - Verfügbarkeit 281
- live-resize 553
- ll 41
- LOAD 137
- loadLibrary() 310
- LOCAL_ADMIN_APPS_DIR 583
- LOCAL_APPS_DIR 584
- LOCAL_DEVELOPER_DIR 584
- LOCAL_LIBRARY_DIR 584
- Locale 157
- localhost 239, 449
- log4j 62, 465
- loggc 575
- lokale Klasse 527
- lokale Variable 515
- Lokalisierung 149, 157, 192, 210, 218, 591
- Look & Feel 56, 144
 - im Programmcode setzen 145
- lookup() 472
- LSBackgroundOnly 171
- LShasLocalizedDisplayName 219
- lsMac 199

M

- Mac 591
- Mac OS
 - Extended Format 69
 - Java 59
 - Updates 538
 - Versionen 8 und 9 23, 49, 531
- Mac OS X 23
 - Application Bundle 187
 - Applikation 187
 - Emulation 13
 - Java-Struktur 60
 - Systemarchitektur 55
 - Version 10.1 35
 - Versionen 586
- Mac OS X Server 24, 464
 - /etc/hostconfig und MYSQL 435
 - Dokumentation 479
 - MySQL 432
 - MySQL-Startuptem 434
- MAC-Adresse 310, 591
- Mach-O 318
- MACINTOSH 580
- Macintosh Look & Feel 142
- Macintosh Runtime for Java 23
- MacLookAndFeel 144
- MacRoman 296, 549
- MacSOUP 46
- MagicDraw 417
- Mailingliste, java-dev 46
- main() 514
 - Argument-Array 132
- main.m 331, 337
- Main-Class 184, 238
- MainClass 212, 216, 550
- MainMenu.nib 331
- make 310, 385, 412
- Makefile 310
- makeKeyAndOrderFront() 337
- man pages 42
- Manifest 183, 238, 391, 492
- MANIFEST.MF 183
- markMisspelledWords() 346
- Maske 188
- Math 516
- Maustasten 30
- Maven 396
- MAVEN_HOME 397
- MaxFDLimit 569
- MaxHeapFreeRatio 574
- MaxNewSize 574
- MaxPermSize 575
- MAYSCRIPT 279
- MBean 465
- MDI 29, 143, 591
- ME4SE 496
- Media Access Control 310
- Mehrfachvererbung 521
- Mehrprozessorsystem 575

- Menu 127
- MenuBar 127
- MenuItem 127
- Menüpunkt mit Häkchen 153
- MenuShortcut 127
- Menüzeile 27, 148, 181, 535, 551, 553
 - AWT 127, 134
 - JDialog 163
 - ohne offene Fenster 148
 - Swing 143, 162
- Merge 92
- Meta Package 227
- META_MASK 150
- Metadaten 327, 477
- META-INF 183, 475
- metainf 475
- Metal 142, 146
- MetalLookAndFeel 144
- Meta-Programmierung 290
- METHOD 456
- Method 289
- Methode 513
 - veraltet 328
- Metrowerks CodeWarrior 110
- MicroEmulator 496
- Microsoft 267
- Microsoft Office 11
- MIDI 375
- MIDlets 485
 - beenden 489
 - Formular anzeigen 489
 - implementieren 488
 - JAR-Archiv 491
 - Zustand 489
- MIDlet-Suite 492
- MIDP 484, 591
 - API 486
 - Applikation 484
 - Bluetooth 500
 - Emulator 486, 494, 507
 - Ereignisbehandlung 507
 - Farbe 506
 - Game Action 507
 - Grafik 506
 - Referenz-Implementation 485, 496
 - RMS 493
 - Unicode 504
- MIME-Typ 204, 217, 243, 447, 502
- MinHeapFreeRatio 574
- Minimal.app 596
- MisFox 200
- MisspelledWord 347
- mixed 576
- mn 573
- Mnemonics 165
- modaler Dialog 120
- Model, View, Controller 121
- module 473
- more 184
- MouseAdapter 135
- MouseEvent 135
- MouseListener 136
- mousePressed() 135
- mouseReleased() 135
- Movie 362, 366, 371
- MovieController 372
 - play() 373
- Mozilla 46, 259, 268, 281
- MP3 374
- MPEG 362
- mpkg 227
- MRJ 23, 49, 59, 91, 121, 328, 531, 586, 591
 - FAQ 536
 - SDK 539
- MRJ Software Development Kit 324
- mrj.version 49, 548, 586
- MRJAboutHandler 122, 130, 552
- MRJAdapter 131, 147
 - openURL() 300
- MRJApp.properties 192, 215, 550
- MRJAppBuilder 193
- MRJApplicationUtils 123
- MRJClasses 59, 91
- MRJFileUtils 300, 328, 330
- MRJOpenApplicationHandler 122, 131
- MRJOpenDocumentHandler 123, 131, 205
- MRJOSType 328
- MRJPluginCarbon 281
- MRJPrefsHandler 123, 131
- MRJPrintDocumentHandler 123
- MRJQuitHandler 123, 132
- MRJToolkit 23, 536
- MRJToolkit.jar 328
- MRJToolkitStubs 139, 329

ms 573
Multimedia 365
Multiple Document Interface 143
Multitasking 537
Musik 366, 374
mv 179
MVC 121, 333, 591
mx 573
MySQL 432, 459
 beenden 436
 grafische Oberflächen 437
 JDBC-Treiber 437
 starten 435
mysql 436
MySQL Manager 432
mysqladmin 435
MYSQLCOM 434
mysqld_safe 435
mysqldump 437
mysqlimport 437

N

NAME 279
Namensdienst 463
nanoTime() 302
NanoTimer 302
narrow() 472
native Routinen 309, 325
Navigator 267, 274
nearestneighbor 557
net.java.games.jogl 359
NetBeans 103, 206
 Launcher 104
NetInfo 433, 591
NetInfo Manager 432, 438
Netscape 267, 274, 281
netscape.jar 277
netscape.javascript 275
Netzwerkkarte 310
Netzwerkkommunikation 295
Netzwerkkonfiguration 302
Netzwerkverbindung 240
never 166
newInstance() 141, 290, 291
Newline 548
NewObjectArray() 323
NewRatio 574

newrt.jar 543
Newsgroup 530
NewSize 574
NewSizeThreadIncrease 576
Newsreader 46
NeXT 57, 269
next() 428
NeXTSTEP 331, 592
NeXTSTEP-Konfigurationsdatei 450
NFS 297
nib4j 340
NIB-Datei 331, 337, 338, 592
niutil 433
nm 315
noclassgc 575
NORMAL 154
Normalisierung 426
NORTH 125
NPAPI 325
NSAppleEventDescriptor 341
NSAppleScript 340
NSApplicationMain() 337
NSJavaNeeded 339
NSJavaPath 339
NSJavaRoot 339
NSMenu 555
NSMutableDictionary 341
NSObject 333
NSSpeechRecognizer 344
NSSpeechSynthesizer 344
NSSpellChecker 345
NSWorkspace 300
NumberFormat 514

O

Oberflächendesign 142
Oberklasse 520
Obfuscator 404, 508
OBJECT 255
Objective-C 57, 330, 333, 589
 id 334
Objekt 518
Objektdatei 318
objektorientierte Datenbank 423
Objektvariable 519
OBJROOT 579
ODBC 441, 592

- ODBC Administrator 441
- ODBMS 592
- Office 11
- offline-allowed 240
- Öffnen mit 179
- OmniCore CodeGuide 106
- OmniWeb 269
- OO 592
- OOA 592
- OOD 592
- OODB 423
- OOP 592
- open 297, 298, 300
- Open Database Connectivity 441
- Open Source 56, 592
- OpenGL 56, 358, 381, 558
- OpenQTJ 373
- OpenTalk 302
- openURL() 300
- Opera 270
- operationFailed() 304
- Optimierung 405
- OptimizeShapes 558
- Option-Taste 16
- Oracle 440, 442, 539
 - JDeveloper 108
- Oracle 10g 440
- Ordner 30, 592
- Organizer 483, 500
- os.arch 548
- os.name 48, 548, 585
- os.version 548
- OSA 342
- osalang 342
- osascript
 - AppleScript 342
 - Sprachausgabe 344
- OSXAdapter 164
- osxutils 199
- OTHER_JAVA_CLASS_PATH 581
- otool 321
- Outlet 333, 335, 338
 - Java 337
- OutOfMemoryError 538
- Output Files 93
- OutputStreamWriter 296
- <oXygen> 417

P

- pack() 123
- Package 67, 254, 510, 592
- PACKAGE_TYPE 582
- packageIsTraversable 166
- PackageMaker 192, 223
- PageAttributes 375
- PageFormat 379, 380
- paint 253
- Paket 67, 510, 592
 - anonymes 511
 - Installation 223
 - Java 187, 254
 - Programm 187, 190
- Paket → s. Package
- Paket-Bit 187
- Paketinhalt zeigen 190
- Paketname 176
- Palm Desktop 500
- PalmOS 500
- Panel 120, 125
- Panther 24, 592
- Papierformat 377
- parallele Schnittstelle 301
- PARAM 254, 261
- Parameter 550
 - formale 513
- Partition 31, 592
- passwd 433
- PATH 69, 386, 414
- path.separator 294, 548
- PB 76, 592
- pbdevelopment.plist 582
- pbhelpindexer 85
- pbxbuild 87, 577
- PDA 483, 500
- PDF 56, 190, 377
 - anzeigen 298
 - erzeugen 299
 - Plugin 299
 - sichern als 299
- PearPC 13
- Perf 301
- Perforce 94
- Performance Pack 362
- Performanz 408
- Permanente Generation 575
- Permission 263

Persistenz 463, 467, 592
 Personal Web Sharing 242, 257, 304
 Petstore 62
 Pfadangaben 35, 51
 im Buch 596
 Pfadtrennzeichen 294
 pg_ctl 438
 Photoshop 188
 PHP 445
 pico 37
 PICT 354, 373, 374
 Pixie 189
 pkg 224
 PkgInfo 191, 207, 211, 582
 PKGINFO_FILE 582
 PLAF 142, 592
 PLAIN_DIALOG 134
 Plattformunabhängigkeit 287, 309
 Player 363
 plist 54, 208, 214, 450
 Pluggable Look & Feel 142
 Plugin 59, 60, 65, 255, 258, 277, 325, 536
 deaktivieren 266
 Einstellungen 259
 Java Embedding 259
 PNG 189, 354, 492
 Policy 263
 policytool 264
 Polymorphismus 523
 POP3 295
 Pop-up-Menü 135
 Popup-Trigger 136
 Port, 8080 449
 Portabilität 287, 309
 PortableRemoteObject 472
 Poseidon 417
 POSIX 347, 592
 PosixSuites 347
 POST 456
 POSTGRES 439
 PostgreSQL 438
 beenden 439
 JDBC-Treiber 439
 starten 438
 StartupItem 439
 Post-Inkrement 517
 PowerBook 11
 PowerPC 548
 PPC 23, 531, 548
 Pramati Server 464
 PRC Converter 500
 PRC-Datei 500
 Prebinding 319
 Preferences 52
 Prefix.h 331
 preverify 490, 491, 494
 Primärschlüssel 425
 Primary Key 425
 primitiver Datentyp 515
 print() 295, 379, 380, 514
 Printable 379
 PrintCompilation 576
 printenv 69
 PrinterJob 379
 printf 546
 PrintJavaStackAtFatalState 569
 PrintJob 376
 end() 376
 println() 295, 514
 PrintSharedSpaces 566, 576
 printStackTrace() 472
 PrintStream 295
 PrintTenuringDistribution 575
 PrintWriter 295, 446
 private 514
 Process 343
 Processing 348
 processMouseEvent() 136
 PRODUCT_NAME 389, 392, 579
 prof 569
 Profil 484
 Profiler 408
 Programm 509
 deinstallieren 187
 löschen 187
 Programmabbruch 529
 Programmcode 191
 Programme 31
 Ordner 220, 225
 Programmiereditor 111
 Programmierrichtlinien 510, 530
 Programm-Manager 235
 Programm-Menü 27, 130, 180, 211

- Programmname 180, 239, 552
- Programmpaket 166, 187, 190, 236, 317, 338, 394, 582, 592, 596
 - im Dateiauswahl-Dialog 138
 - von Hand erzeugen 206
- Programmsymbol 180, 188, 211, 236
- ProGuard 405
- Project Builder 24, 42, 62, 76, 90, 203, 577
- Project Rave 104
- project.xml 397
- PROJECT_NAME 578
- Projekt 90, 387
 - Gruppe 80
 - Index 83
 - Struktur 79
 - Template 77
 - Typ 77
- Projektplanung 416
- Projektverwaltung 75
- Prolog 209
- Propeller-Taste 16
- Properties 214, 293
- properties, Umlaute und Sonderzeichen 158
- properties-Datei 158
- Property 547
- Property List 208, 219, 592
- Property List Editor 214
- protected 513
- Protokoll 525
- Prototyp 275
- Proxy
 - HTTP 259
 - HTTPS 259
- Proxy-Einstellung 250
- Prozedur 513
- prozedurale Programmierung 513
- Prozess 33, 533, 537
- Prozessverwaltung 347, 535
- ps 33, 533
- psmp-Datei 225
- psql 438
- pthread 324
- public 513
- put() 166
- putClientProperty() 160, 166
- PWD 579

- pwd 53
- Python 112

Q

- QDRect 374
- QT 365
- QTCanvas 367
- QTException 368
- QTFactory 368, 372
- QTFile 374
- QTImageProducer 374
- QTJ 366
- QTJava 366, 374, 556
 - Versionen 366
 - Xcode 369
- QTJava.zip 369
- QTSession 367, 369, 371, 373
- QTUtils 374
- quality 557
- Quaqua-Look&Feel 172
- Quartz 56
- Quartz Extreme 196, 559
- Quelltext 411
- Quelltext-Datei 510
- Quelltextkodierung 83
- Quelltextverwaltung 94
- Query 424
- QuickTime 56, 362, 365, 381
 - Im- und Export 373
 - Versionen 366
- QuickTime for Java 366
- QuickTime Player 366
- quicktime.app.display 366, 367
- quicktime.app.display.FullScreen-Window 556
- quicktime.app.view 366, 371
- quicktime.std 371

R

- Radio-Button 125
- Rational Rose 417
- RCEEnvironment 94
- RDBMS 592
- Receipts 227
- Rechnerstart 31, 531, → s. Booten
- Rechtschreibprüfung 36, 345
- Rechtsklick 80, 135
- ReduceSignalUsage 570

Referenzdatentyp 516
Reflection 139, 163, 289, 290
registerAboutHandler() 123, 130, 552
registerJava13Handler() 140
registerJava14Handler() 163
registerOpenApplicationHandle 123
registerOpenDocumentHandle 123
registerPrefsHandle 123
registerPrintDocumentHandle 123
registerQuitHandle 123
reguläre Ausdrücke 22
Relation 425
relationale Datenbank 423
remote 471
Remote Method Invocation 469
RemoteException 469
Remote-Interface 469
RenderDrawArc 562
RenderDrawOval 562
RenderDrawPolygon 562
RenderDrawRect 562
RenderDrawRoundRect 562
RenderDrawShape 562
RenderFillArc 562
RenderFillOval 562
RenderFillPolygon 562
RenderFillRect 562
RenderFillRoundRect 562
RenderFillShape 562
RenderGlyphs 563
RenderImage 562
Rendering Hint 556
RenderLine 562
RenderString 563
RenderUnicode 563
Rendezvous 302, 592
Repository 94
ReservedCodeCacheSize 570
reservierte Schlüsselwörter 509
Resource Fork 188
ResourceBundle 148, 157, 192
Ressourcen, Zugriff 244
RestartService() 450
ResultSet 428, 429
return 514
Reverse Engineering 108
Richtlinientool 264
RIGHT 125
RMI 469
Role 204, 217
rollback() 430
ROOT 452, 455, 457
root 30, 579, 592
Root Pane 134, 161
ROOT-Kontext 452, 457
Round Trip Engineering 108, 417
round() 516
rs 570
RS232-Schnittstelle 301
rt.jar 288, 405, 543
Rückgabety 513
Ruhezustand 533
Run 87
run() 130, 527
Rundung 516
runhprof 569
Runnable 133, 527
Runtime.exec() 297, 298, 299, 342
RXTX 301

S
Safari 266, 281
safe_mysql 435
Sandbox 235, 245, 262
SAVE 137
Schaltfläche 120
 Reihenfolge beim Mac 126
Schleife 517
Schlüssel 425
Schlüsselwort 509
Schnittstelle 123, 525
Schrägstrich 69
Schreibtisch 590
SCM 94, 592
Scriptlet 454
Scrollbar 159
SDK 463, 593
Search Paths 91
Security Policy 263
Security-Manager 246
Segmente 42
Selbst-Referenz 520
SELECT FROM 425, 428
Separator 293
Serializable 169
serielle Schnittstelle 301

Server 24, 234, 242
 Server → s. Web-Server
 Server-Administration 464
 Server-Anwendungen 170
 Server-VM 566
 Service 450
 serviceFound() 303
 serviceLost() 304
 servlet.jar 475
 Servlet-Container 445, 448, 465
 ServletException 446
 Servlet-Kontext 457
 Servlet-Mapping 458
 Servlets 445, 446, 467, 472
 Xcode 447
 Session 454, 467
 SessionBean 78, 467, 470
 SessionContext 470
 session-type 471
 setAccelerator() 149, 165
 setAutoCommit() 430
 setBackground() 134, 155
 setBounds() 155
 setContentType() 446
 setDirectory() 137
 setEnabled() 151
 setEnabledPreferencesMenu() 155
 setEnabledPrefs() 163
 setenv 65
 setExtendedState() 128, 154
 SetFile 199, 207, 223, 328
 setFile() 138
 setFilenameFilter() 138
 setFullScreenWindow() 554
 setHandled() 156, 164
 setHelpMenu() 127
 setJMenuBar() 149
 setLayout() 125
 setLocation() 123, 129
 setLookAndFeel() 145
 setMenuBar() 127
 setMenuBarVisible 555
 setMnemonic() 165
 setMode() 138
 setProperty() 549
 setReadOnly() 428, 430
 setRenderingHint() 556
 setResizable() 123
 setSessionContext() 470
 setState() 128, 153
 Setter 519
 setUndecorated() 134, 155, 554
 setUp() 400
 setVisible() 124, 133
 setWindowDecorationStyle() 134
 sh 177
 Sharing 242
 SharingMenu 243
 Shark 409
 Sheets 38
 Shell 38, 40, 51, 64, 93, 176, 536, 593
 Shell Script Files 93
 Shell Script Target 90
 SHIFT_MASK 149
 Shift-Taste 16
 Shortcut 16, 28
 show() 134, 135
 showConfirmDialog() 133
 showDocument() 248, 272, 281
 showGrowBox 551
 showStatus() 271
 Sicherheit 463
 Sicherheitslücke 11
 Sicherheitsrichtlinie 263
 SIGHUP 570
 SIGINT 570
 Signale 570
 Signatur 211, 323, 525
 Signature 199
 Signieren 246, 262
 SIGQUIT 570
 SIGTERM 570
 SIGUSR1 570
 SIGUSR2 570
 SimpleDateFormat 514
 SimpleUniverse 356
 Sites 242
 Sitzung 467
 SKIP_INSTALL 578
 Skript 176, 593
 Skripteditor 340
 Skriptsprache 340, 413
 smallTabs 553
 Smart Group 80
 anlegen 82
 SMB 297

- Soft Button 503
- Software-Aktualisierung 44, 249
- Softwareprozess 416
- Solaris 23, 24, 55, 302, 311
- Sonderzeichen 38
- Sound 374
- Source Control Management 94
- sourcepath 582
- Sources 92
- SOUTH 125
- Speech Framework 343
- speed 557
- Speicherauslastung 33
- Speicherschutz 535
- Speicherverbrauch 408
- Speicherverwaltung 519
- Spelling Framework 345
- SpellingChecker 347
- Spiele 359
- Splash-Screen 240
- Sprachausgabe 343
- SQL 424, 593
 - Abfrage 425
- SQL/J 423, 593
- SQLException 429
- src.jar 411
- SRCROOT 579
- ss 573
- Stack 573
- Stack Trace 41, 88, 569
- staff 579
- Standard-Klassenpfad 68
- Standardverzeichnisse 583
- Stapelverarbeitungs-Datei 177
- start() 130, 253
- Startpunkt 514
- startRealtimeChecking() 346
- StartService() 449
- Start-Tag 209, 254
- StartupItem 434, 439, 449
 - MySQL 434
 - PostgreSQL 439
- StartupParameters.plist 450
- Startvolume 31, 532, 593
- Stateful Session-Bean 468
- Stateless Session-Bean 468
- Statement 424, 428, 429
- static 513
- statische Imports 545
- statischer Initialisierungsblock 311
- Steuerungsobjekt 121, 467
- Steuerung-Taste 16
- stop() 253
- StopService() 450
- String, getBytes() 296
- stringFlavor 169
- StringTokenizer 288
- Struts 459
- Stub 191
- Stub-Klasse 139, 163, 329
- su 438
- Subklasse 520
- Subroutine 513
- Subversion 94, 593
- Suchreihenfolge von Erweiterungs-
Bibliotheken 68
- sudo 592
- Suffix 593
- suggestGuessesForWord() 347
- Suite 347, 400
- Suite/P Toolkit 347
- Sun 22, 70
 - Bug Parade 292
- Sun Java Studio Creator 104
- sun.boot.class.path 288
- sun.cpu.endian 548
- sun.jdbc.odbc.JdbcOdbcDriver 441
- sun.misc.Perf 301
- super 521
- super() 521
- Superklasse 520
- Superuser 592, 593
- Superuser-Rechte 348
- SurvivorRatio 574
- Swing 22, 56, 119, 142, 251, 272, 291, 537, 593
 - Anpassungen für andere Systeme 150
 - Brushed Metal 168
 - Cocoa 339
 - Performanz 291
 - Threads 274, 291
 - Xcode 146
- swing.defaultlaf 144
- swing.properties 144
- SwingSet2 142

- SwingUtilities 133, 145, 274
 - invokeAndWait() 291
 - invokeLater() 291
- switch 517, 545
- Switcher 12, 371
- Sybase ASE 441
- symbolischer Link 589
- SYMROOT 578
- Synthesizer 344
- Sys 50, 140, 163, 585, 596
 - getHomeDirectory 53
 - getJavaHome 66
 - getLocalPrefsDirectory 53
 - getPrefsDirectory 53
 - getPrefsDirectory() 293
 - getWorkingDirectory 53
 - isAMac 50
 - isLinux 50
 - isMacOS 50
 - isMacOSX 50
 - isOS2 50
 - isWindows 50
- System
 - currentTimeMillis() 301
 - Kodierung 296
 - nanoTime() 302
 - Programmierumgebungen 57
 - Property 48, 52, 214, 241, 246, 488, 547, 567
- System.exit 246, 262
- System.exit() 132
- System.gc() 575
- System.getProperty() 293
- System.loadLibrary() 311
- SYSTEM_ADMIN_APPS_DIR 583
- SYSTEM_APPS_DIR 583
- SYSTEM_CORE_SERVICES_DIR 583
- SYSTEM_DEMOS_DIR 583
- SYSTEM_DEVELOPER_APPS_DIR 583
- SYSTEM_DEVELOPER_DEMOS_DIR 583
- SYSTEM_DEVELOPER_DIR 583
- SYSTEM_DEVELOPER_DOC_DIR 583
- SYSTEM_DEVELOPER_GRAPHICS_TOOLS_DIR 583
- SYSTEM_DEVELOPER_JAVA_TOOLS_DIR 583

- SYSTEM_DEVELOPER_PERFORMANCE_TOOLS_DIR 583
- SYSTEM_DEVELOPER_UTILITIES_DIR 583
- SYSTEM_DOCUMENTATION_DIR 583
- SYSTEM_LIBRARY_DIR 583
- Systemabhängigkeit 309
- Systemanpassungen 47
- Systemarchitektur 55
- Systemeinstellungen 26, 532
 - Sprache 344
- Systemerweiterungen 534
- Systeminstallation 32, 42
- Systemordner 533
- SystemStarter 451
- Szene-Graph 357

T

- Tabelle 423, 424
- Tabs 166, 553
- Tabulator 590
- Tag 209, 254
- Tag Library 455
- Taglib 455
- Target 80, 90, 387
 - Abhängigkeiten 90, 314
 - aktives 80
 - Cocoa Java Specific 339
 - Cocoa Specific 339
 - Legacy 90
 - Pure Java Specific 338
 - Typ 90
- TARGET_BUILD_DIR 578
- TARGET_NAME 579
- TargetSurvivorRatio 574
- Task 387
- Tastatureingaben 27
- Tastaturkürzel 16, 28
 - AWT-Menüs 128
 - Swing-Menüs 149
- Tastaturkürzel → s. Shortcut
- TCP 303
- tcsh 39, 65, 177, 414
- tearDown 400
- TEMP_DIR 491, 579
- TEMP_FILES_DIR 579
- Template 24, 77

- temporärer Speicher 537
- temporäres Verzeichnis 330
- Tenured-Generation 574
- Terminal 38, 51, 176, 593
- Test First-Ansatz 398
- TestCase 398
- Testen 398
- Testfall 401
- TestRunner 400, 402
- TestSuite 398
- TEXT 328
- text/html 204
- text/plain 204
- textantialiasing 557
- Textausgabe 536
- TextEdit 35, 296, 298
- Textkodierung, Xcode 158
- Text-to-speech 343
- this 520
- Thread 130, 527, 528, 537, 576
- ThreadStackSize 576
- Thread-Synchronisation 569
- throw 529
- Ticker 505
- Ticks 326
- TIFF 354
- Tiger 24, 541, 593
- tiger 543
- tigerc 543
- Tilde 38
- title 239
- TKPLAFUtility 144
- TLAB 576
- toFront() 154
- Together Control Center 108
- Together Solo 108
- Tomcat 445, 448, 465
- Tomcat Monitor 452
- Tool 203
- Toolkit 149, 374, 376
- tools.jar 288
- top 33
- toString() 520, 521
- tr 177
- transaction-type 471
- Transaktion 431, 591
- Transaktionsverwaltung 463
- Transferable 169
- Transform3D 357
- TransformGroup 357
- Transmit 244
- Transparenz 189
- Treiber, JDBC 427
- Trennzeichen
 - Dateien 293
 - Pfadangaben 294
- Trojaner 11
- TruBluEnvironment 533
- TRUE 161
- try 528
- TTS 343
- Typ einer Klasse 289
- TYPE_INT_ARGB 563
- Typecast 516, 589
- Type-Code 191, 199, 217, 296, 327
- Typumwandlung 516, 589

U

- Über...-Menüpunkt 130
- Überladen 519
- Überschreiben 521
- Übersetzer 512, 589
- Übersetzung 149, 157, 192, 591,
 - s. Compiler
- UDP 303
- UFS 297
- UI 119, 593
- ui.jar 288, 328, 329
- UIManager 145
 - put() 166
- Umgebungsvariable 65, 69, 94, 176
- UML 108, 113, 416, 593
- Umlaute 83
- Umschalttasten 28
- Unicode 192, 209, 219, 295
 - Escape-Sequenz 129, 158
- UninstalledProducts 578
- Unit-Test 398
- UNIX 11, 293, 295, 593
- UnsatisfiedLinkError 314
- unsichtbares Fenster 155
- UnsupportedLookAndFeelException
 - 145
- Unterbrechungspunkt 88, 102
- Unterklasse 123, 520
- Unterpaket 512

- Unterprogramm 513
- Update 44
- updateComponentTreeUI() 145
- URI 456, 593
- URL 299, 593, 596
- URL-Clip 596
- USB 301
- USB-Docking-Station 500
- USB-Maus 80
- UseAltSigs 570
- usecolorsync 563
- UseConcMarkSweepGC 575
- usedebugkeys 561
- use-file-dialog-packages 553
- UseFileLocking 573
- useMDEF 553
- Usenet 46
- UseParallelGC 575
- User Interface 119
- user.dir 53
- user.home 52, 53
- USER_APPS_DIR 584
- USER_LIBRARY_DIR 584
- Users 31
- useScreenMenuBar 551, 553
- UseSharedGen 573
- UseSharedSpaces 576
- UseTLAB 576
- UseTwoImageLazyPixelConversion 563
- usr 30
- UTF-16 192, 219, 296
- UTF-8 209, 239, 296

V

- Variable 515
- variable Argumentlisten 545
- Vector 147
- veraltete Methode 328
- verbose 567
- Vererbung 520, 526
 - Klassenbasiert 275
 - Prototyp-basiert 275
- Verknüpfung 31
- version.plist 192
- Versionierung 94
 - Applet 261
 - Installationspaket 227

- Java Web Start 235
- Versions (Verzeichnis) 63, 198, 211
- Versionsangabe 194, 240
- Versionsverwaltung 94
- VERTICAL_SCROLLBAR_ALWAYS 159
- vi 37, 40
- Videodaten 362, 366
- Viewer 204, 217
- Virtual Machine 22
- Virtual Reality 366
- Virus 11
- VISE 230
- VisualParadigm 417
- VK_XXX 127, 149
- VM 22
- VMOptions 213, 216, 551, 565
- void 513
- Vollbildmodus 554
- vollqualifizierter Klassenname 511
- Volume 29, 220, 533, 593,
 - s. Laufwerk
 - deaktivieren 221
- Vorabversionen 46
- Vorschau 190
- VR 366

W

- W3C 255
- Wahl-Taste 16
- WAP 502
- WAR 385, 475
- war 459, 473, 475, 581
- web 473
- Web Sharing 304
- web.xml 452, 457, 458, 473, 475
- Webapplikation 78, 455, 457, 474
- Webapplikationsarchiv 459, 473
- webapps 452, 455, 457
- Web-Browser 60, 64, 234, 265, 299, 325
- WebCore 266
- WEB-INF 452, 457, 475
- WebKit 266, 269, 339
- Weblog 459
- WebLogic Server 464
- WebObjects 464, 479
- Web-Server 234, 302, 445
- Web-Sites 239, 242

- web-uri 473
- Weichzeichnen 556, 589
- WEST 125
- while 517
- WHITE 134
- Window 124
- window 276
- WindowAdapter 124
- windowClosing() 124, 365
- WindowEvent 124
- WindowListener 124, 363
- windowModified 160
- Windows 23, 24, 55, 57, 186, 267, 293, 295, 296, 302, 304, 311
- Wireless Toolkit 485, 494
- Word-Dokumente 35
- WorkingDirectory 213, 216, 550
- Wrapper 311
- Wurm 11
- Wurzel-Ebene 161
- Wurzelelement 209
- Wurzelverzeichnis 30
- Wurzelverzeichnis → s. root

X

- X Window System 486
- X₁₁ 486, 495
 - FAQ 508
- Xcode 24, 42, 62, 76, 186, 203, 245, 577, 593
 - Ant 77, 388
 - Ant-Projekttypen 393
 - Applet 257
 - AWT 122
 - Build 86
 - Cocoa Java 331
 - Custom Build Command 389
 - Debugger 88
 - Dokumentation 84
 - Editor teilen 82
 - Eingabevervollständigung 83
 - Executable 392, 401
 - External Target 389
 - Fehler und Warnungen 86
 - Index neu erstellen 83
 - Index Templates 84
 - Java-API-Dokumentation 85
 - Jikes 413
 - JNI 310
 - JUnit 401
 - Klassenhierarchie statt JAR-Archiv 186
 - Konsole 87
 - Lesezeichen 82
 - LiveConnect 277
 - Maven 397
 - MIDlets 487
 - QTJava 369
 - Servlets 447
 - Shark 409
 - Suchen 83
 - Swing 146
 - Target löschen 389
 - Tastenfunktionen 82
 - Textkodierung 158
 - Umlaute im Quelltext 83
 - Unterbrechungspunkt 88
 - Version 2.0 76
 - Versionen 1.2 und 1.5 76
 - XDoclet 477
 - Xcode Tools 42, 386
 - xcodebuild 87, 577
 - xcodeindex 84
 - Xdock 180
 - XDoclet 62, 78, 465, 477, 478
 - Xcode 477
 - XML 22, 54, 208, 238, 279, 593
 - Dokument 214
 - Editor 417
 - Prolog 209
 - Transformation 417
 - Xnoclassgc 409
 - XP 398
 - Xrun 569
 - XrunShark 409
 - XSL-FO 417
 - XSLT 417

Y

 - YourSQL 437

Z

 - Zeichen, Kodierung 295
 - Zeichen, zusammengesetzte 297

Zeichenkette 514
Zeilenende 184, 294
 umwandeln 177
Zeilenumbruch 177, 294, 548
Zeitmessung 301
zentrieren auf dem Bildschirm 129
Zeroconf 302

Zertifikat 246
 selbst-signiert 247
ZIP-Archiv 183
Zugriffsrechte 32, 53, 178
Zusicherung 570
Zwischenablage 248

