frühjahrs campus 2005

Kaffee mit Apfelgeschmack

Optimierung von Java-Anwendungen für Mac OS X

Thomas Much

thomas@muchsoft.com

www.muchsoft.com

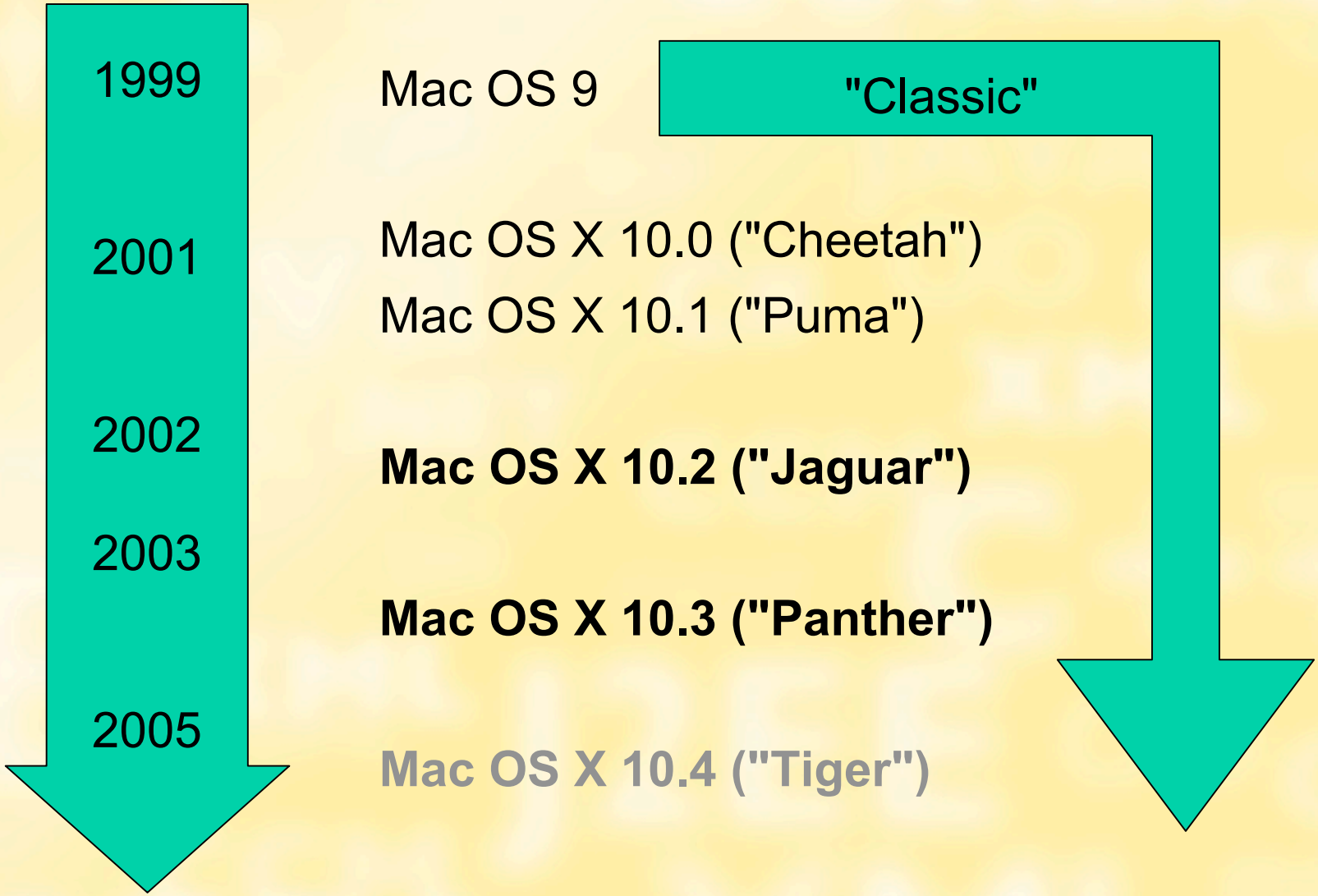
MATHEMA

Themen

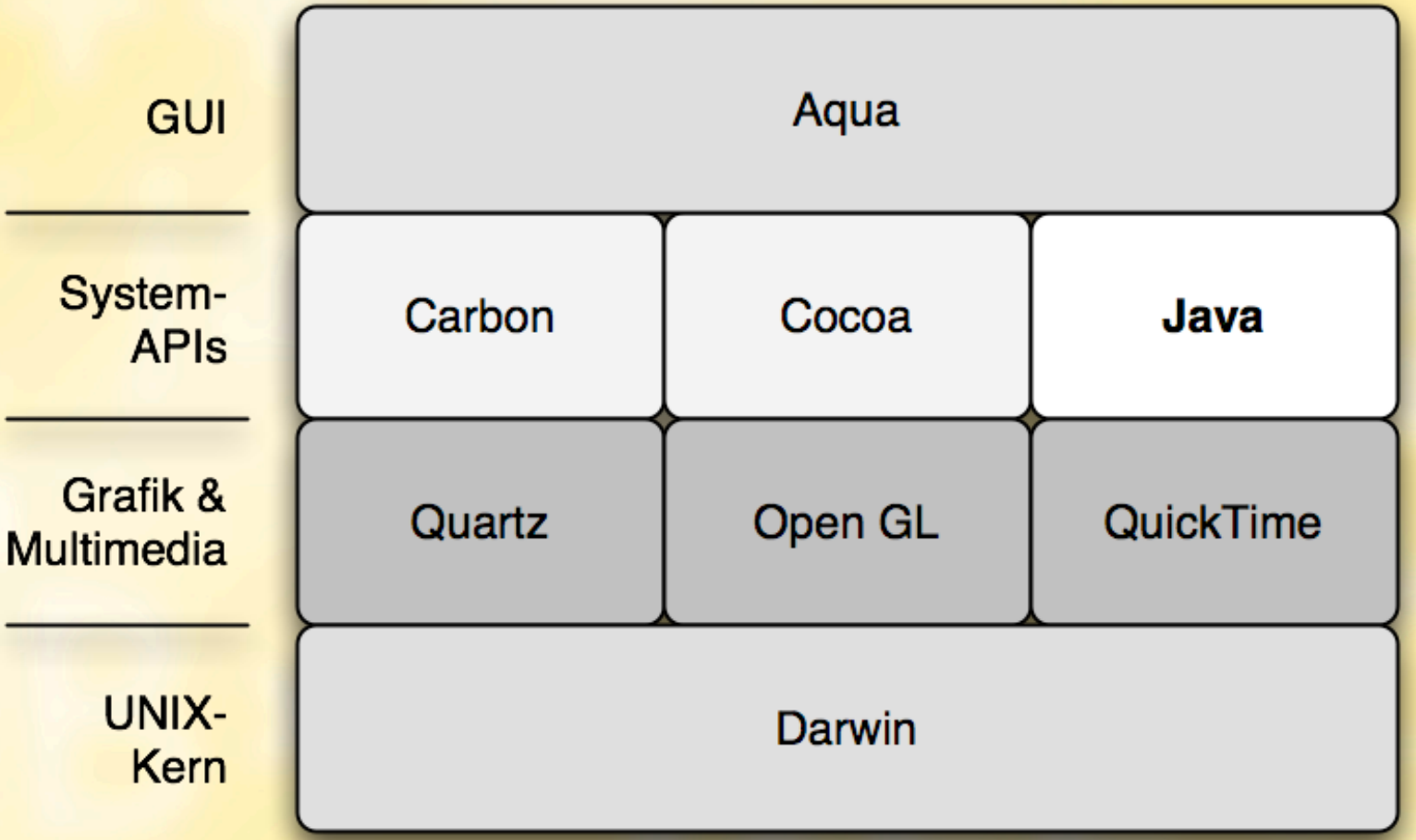
- Überblick: Mac OS X und Java
- Ausführbare Java-Anwendungen
  - JAR-Archive, Java Web Start, Programmpakete
  - Programmpakete mit Xcode bzw. mit Ant und Eclipse
- Benutzungsoberflächen (GUI)
  - bessere System-Integration mit Apple-spezifischen Klassen
  - portable System-Integration
  - Look & Feel und Apples Human Interface Guidelines

Überblick:
Mac OS X und Java

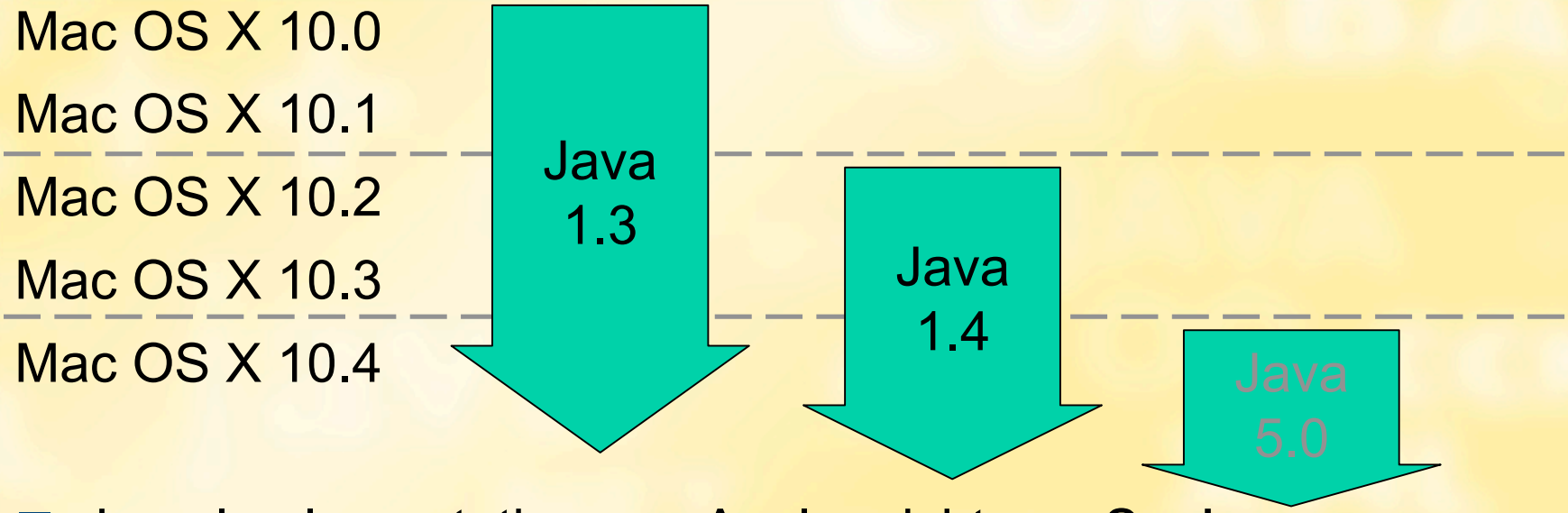
Mac OS (X) Geschichte



Mac OS X Systemaufbau



Java auf Mac OS X



- Java-Implementation von Apple, nicht von Sun!
  - basiert auf Sun-Sourcen, Source-Austausch in beide Richtungen
- nur jeweils eine Hauptversion möglich
  - Java-Updates aktualisieren i.d.R. alle Hauptversionen
  - (offiziell) kein Downgrade, mehrere Testsysteme nötig
  - nicht alle Sun-Bugfix-Versionen verfügbar/nötig

Struktur der Apple-Implementation

- Anwender

  `/Programme/Dienstprogramme/Java/`

  `/Library/Internet Plug-Ins/`

- Entwickler

  `/Developer/Applications/Java Tools/`

  plus Beispiele+Dokumentation in `/Developer`

- System

  `/System/Library/Frameworks/JavaVM.framework/Versions/`

  ```
  1.3.1/
  1.4.2/
      Home/
          bin/
          lib/
  ```

Symlinks in `/usr/bin/` auf

```
<Version>/
    Commands/
        java
        javac
```

JAVA_HOME

- auf `/Library/Java/Home` setzen
- Alias für
  - `/System/Library/Frameworks/JavaVM.framework/Home/`
  - bzw.
    `/System/Library/Frameworks/JavaVM.framework/Versions/1.4.2/Home/`
    (jeweils aktuellste Version)

- in einer Shell (tcsh, bash u.a.):

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

⇨ `/Programme/Dienstprogramme/Terminal.app`

- für den GUI-Parent-Prozess mit RCEnvironment:

⇨ http://www.rubicode.com/Software/RCEnvironment/

Erweiterungsverzeichnisse

- `~/Library/Java/Extensions/`
  `/Library/Java/Extensions/`
  `/System/Library/Java/Extensions/`
- bequem, aber gefährlich
  - stehen in `java.ext.dirs` und werden somit vor dem Klassenpfad ausgewertet
- `/Library/Java/Home/lib/ext/` gibt es auch
  - aber: gehört dem System, wird beim Update überschrieben
  - daher: nicht verwenden

Mac OS X erkennen

- System-Property abfragen:

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

- Bibliothek "Sys" (Open Source) verwenden:

```
import com.muchsoft.util.Sys;
// ...
if ( Sys.isMacOSX() ) {
}
else if ( Sys.isLinux() ) {
}
```

⇨ http://www.muchsoft.com/java/

Konfigurationsdateien speichern

- Benutzerdaten
  - `~/Library/Preferences`
  - konkret z.B. `/Users/much/Library/Preferences/`
  - ⇨ `Sys.getPrefsDirectory()`
- Systemdaten
  - `/Library/Preferences/`
  - ⇨ `Sys.getLocalPrefsDirectory()`
- alternativ mit Java Preferences API (ab Java 1.4)
  - dann aber systemabhängige Dateien

Ausführbare Java-Anwendungen

JAR-Archive

- Problem:
  Anwender möchte Applikation per Doppelklick starten
- Standard für lokale Applikationen:
  - JAR-Archive mit passendem Manifest
  - Windows 2000, XP, Linux KDE ... und Mac OS X
- wie gewohnt:

```
jar cvfm  classes.jar  manifest.txt  *.class
```

manifest.txt:

```
Main-Class: de.paket.Hauptklasse
```

- ausgeführt von

```
/System/Library/CoreServices/Jar Launcher
```

Java Web Start (JAWS)

- Standard für entfernt geladene Applikationen
  - ein Klick startet Applikation:
    `<a href="meineapp.jnlp"> ... </a>`
- seit Mac OS X 10.1 (Java 1.3.1) fester Bestandteil
  - auf anderen Systemen erst ab Java 1.4.1 fest dabei
- Programm-Manager:
  `/Programme/Dienstprogramme/Java/Java Web Start.app`
- lokal testen: Apache Web-Server vorinstalliert
  - "Personal Web Sharing", Home: `~/Web-Sites/`
- kein automatischer Download von JREs
- keine zu alten / zu konkreten Java-Versionen voraussetzen!
  - `<j2se version="1.2+">` geht aber problemlos

Programmpakete (Bundles)

- natives Programmformat von Mac OS X
- Verzeichnisstruktur:

```
MeineApplikation.app/
    Contents/
        Info.plist
        PkgInfo
        MacOS/
            JavaApplicationStub
        Resources/
            MeineApplikation.icns
            Java/
                MeineApplikation.jar
                libMeineApplikation.jnilib
```

Type+Creator: APPL????

Kopie von
/System/Library/Frameworks/
JavaVM.framework/
Resources/MacOS/
JavaApplicationStub

Info.plist (1)

```
<?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>CFBundleExecutable</key>
   <string>JavaApplicationStub</string>
   <key>CFBundleGetInfoString</key>
   <string>1.0 (01.01.2005) Copyright ... </string>
   <key>CFBundleIconFile</key>
   <string>MeineApplikation.icns</string>
   <key>CFBundleName</key>
   <string>MeineApplikation</string>
```

Info.plist (2)

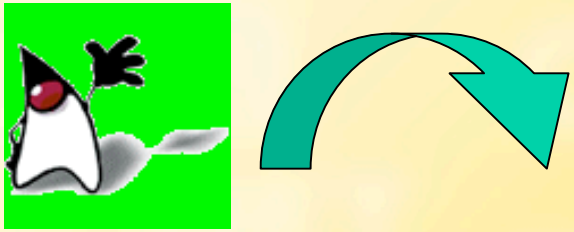
```
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleSignature</key>
    <string>????</string>
    <key>Java</key>
    <dict>
        <key>ClassPath</key>
        <string>$JAVAROOT/MeineApplikation.jar</string>
        <key>MainClass</key>
        <string>de.paket.Hauptklasse</string>
        ...
    </dict>
</dict>
</plist>
```

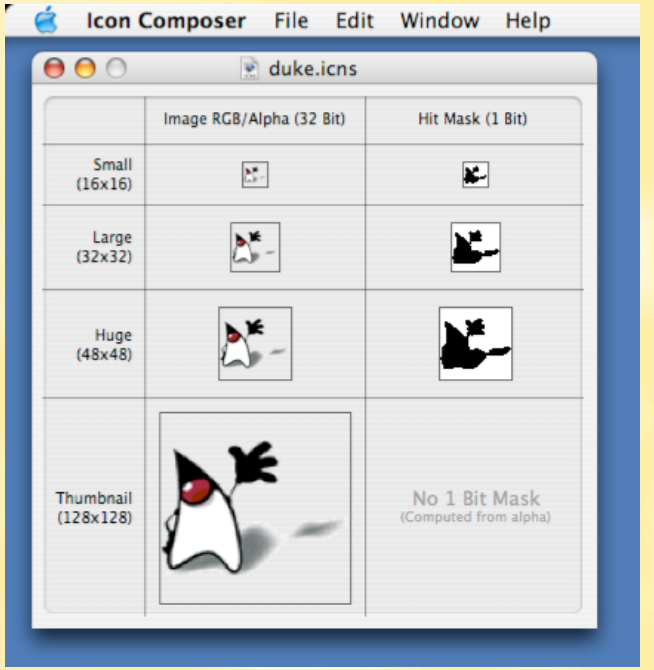
Info.plist (3)

```
<key>Java</key>
<dict>
    ...
    <key>JVMVersion</key>
    <string>1.4+</string>
    <key>Properties</key>
    <dict>
        <key>apple.laf.useScreenMenuBar</key>
        <string>true</string>
        <key>apple.awt.showGrowBox</key>
        <string>true</string>
    </dict>
    <key>VMOptions</key>
    <string>-Xfuture</string>
```

Programmsymbol (Icon)

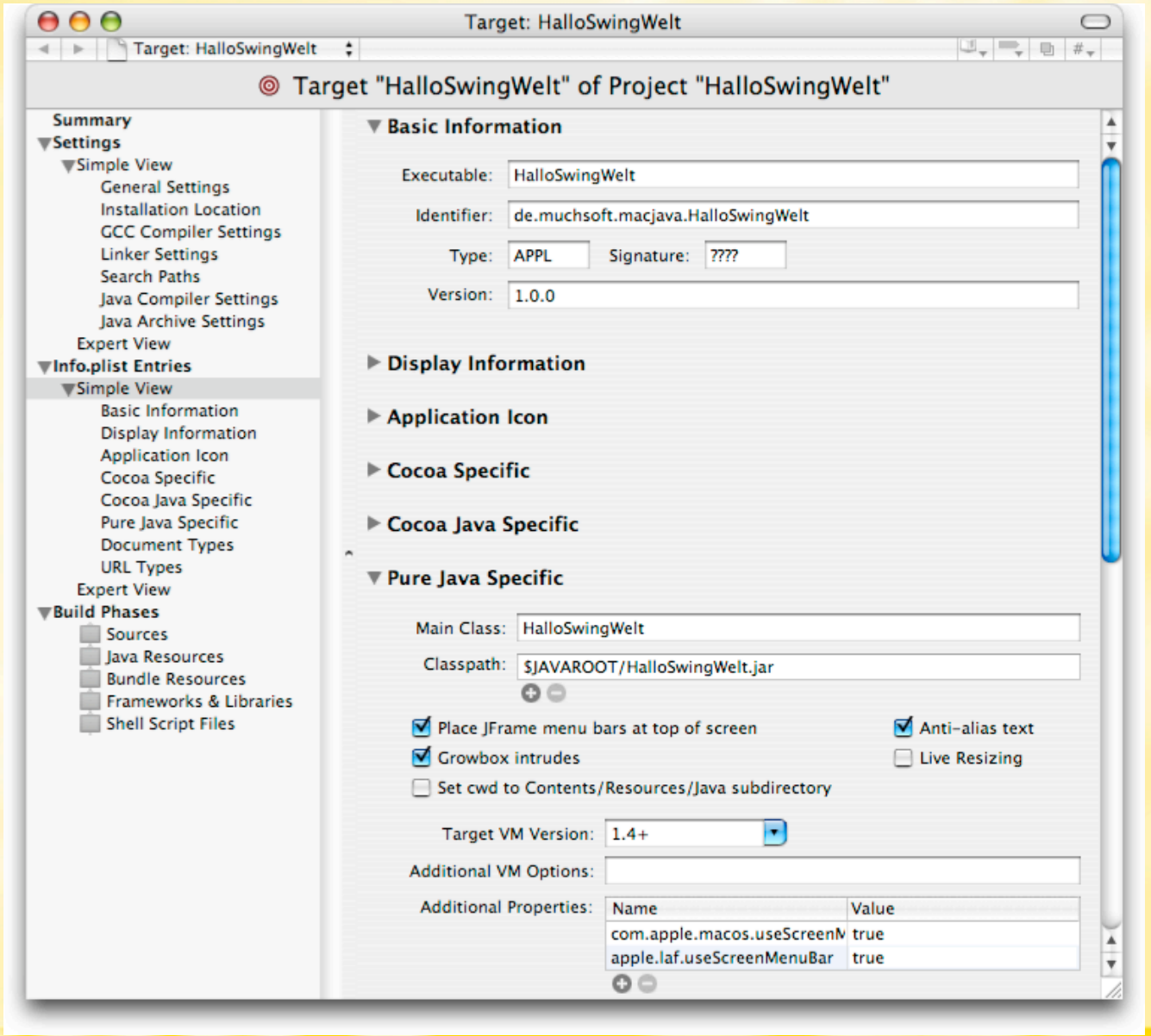
- von `/Developer/Applications/Utilities/Icon Composer.app` aus Bilddatei mit Transparenz/Alphakanal icns-Datei erzeugen lassen:





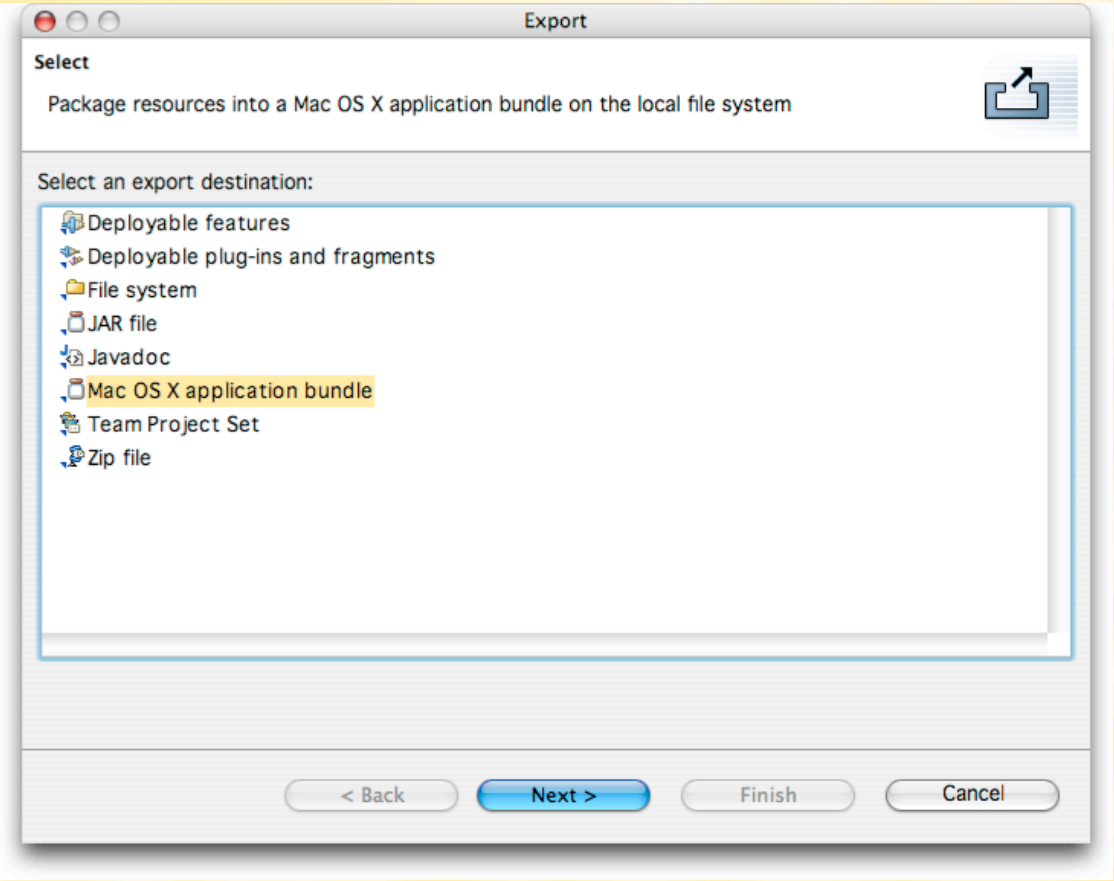
- auch normale Bilddatei möglich
  - einfach JPEG-, TIFF-, PNG-Datei in Info.plist eintragen
  - minimale Maske wird dann aber nicht automatisch erzeugt
- bei Shell-Skripten über `java`-Optionen konfigurierbar

Xcode



MATHEMA

Export als Bundle mit Eclipse 3.0



- nur in der Mac-Version
- scheint nur mit SWT problemlos zu funktionieren

Ant (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<project default="build" basedir=".">
  <property environment="env"/>
  <property name="product"  value="${env.PRODUCT_NAME}"/>
  <property name="build"    location="build"/>
  <property name="jar"      location="${build}/${product}.jar"/>

  <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"/>

  <target name="build" depends="compile">
    ...
  </target>
```

Ant (2)

```
<property name="resources" location="${contents}/Resources"/>
<property name="java"      location="${resources}/Java"/>
<mkdir dir="${macos}"/>
<mkdir dir="${java}"/>
<copy todir="${macos}" file="/System/Library/Fram
                             /JavaVM.framework/Versions/Current
                             /Resources/MacOS/JavaApplicationStub"/>
<copy todir="${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}"/>
<chmod file="${stub}" perm="755"/>
```

oder aus bestehendem Bundle kopieren

Lokalisierung

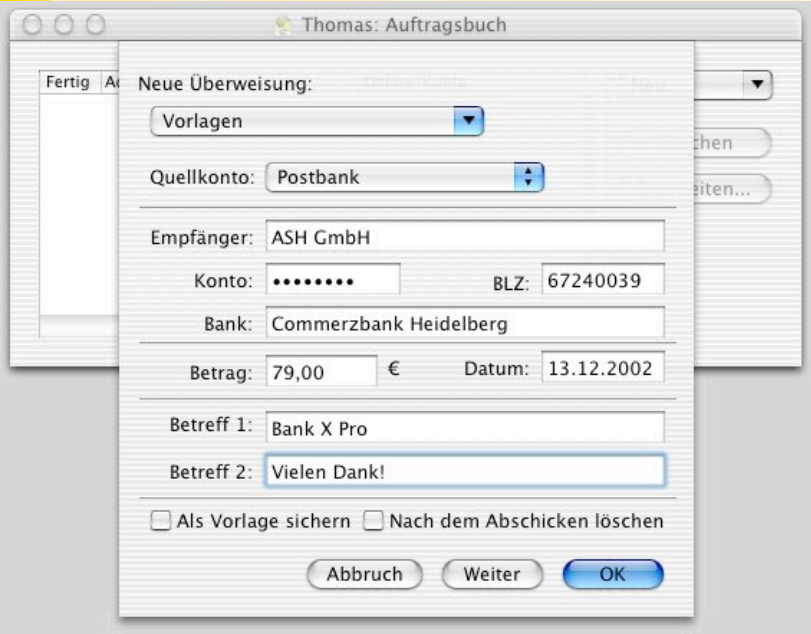
```
MeineApplikation.app/
    Contents/
        Info.plist
        PkgInfo
        MacOS/
            JavaApplicationStub
        Resources/
            MeineApplikation.icns
            English.lproj/
            French.lproj/
            German.lproj/
            Java/
                MeineApplikation.jar
                MeineApplikation.properties
                MeineApplikation_de.properties
```

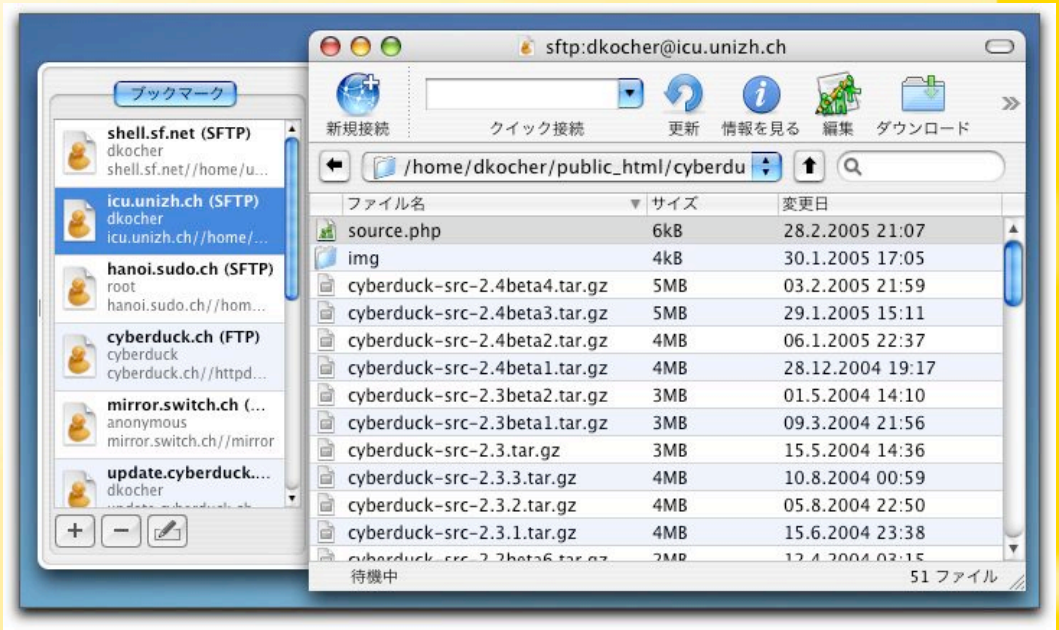
leer, aber für lokalisiertes Apfel- und Programm-Menü nötig

Benutzungsoberflächen (GUI)

Cocoa & Aqua

- das Cocoa-Framework wird normalerweise mit Objective-C programmiert





- auch mit Java ansprechbar: Cocoa-Java-Bridge
  - hier aber nicht Thema, da nicht portabel
- AWT & Swing: von Apple an Aqua-Oberfläche angepasst

GUI-Einbindung

- AWT: wie üblich System-Aussehen (beim Mac also Aqua)
- Swing: System-L&F ist Aqua
  - bei Entwicklung auf anderem System besser Metal-L&F verwenden, da nur dieses Aussehen systemübergreifend gleich ist
  - dann wirken Java-Anwendungen aber wie Fremdkörper, Benutzer-Akzeptanz schwierig (gilt nicht nur für Mac OS X)
- technisches Problem: Interaktion mit Systemoberfläche (z.B. Apfel- und Programm-Menü)
  - Lösung: spezielle Apple-Klassen
  - leider unterschiedlich für Java 1.3 und 1.4 ...

Apple-Klassen für Java 1.3

```
import com.apple.mrj.*;

public class AWTTest extends Frame
                     implements MRJAboutHandler {
  public AWTTest() {
    ...
    MRJApplicationUtils.registerAboutHandler( this );
  }
  public void handleAbout() { ... }
}
```

- weitere Handler für Quit, Prefs, OpenDocument ...

Apple-Klassen für Java 1.4

```
import com.apple.eawt.*;

public class SwingTest extends JFrame
                       implements ApplicationListener {
  public SwingTest() {
    ...
    Application app = new Application();
    app.addApplicationListener( this );
  }
  public void handleAbout( ApplicationEvent e ) { ... }
  public void handleQuit( ApplicationEvent e )  { ... }
  ...
}
```

Portables Einbinden der Apple-Klassen

- Problem: ClassLoader referenziert Klassen, selbst wenn sie nur bedingt instanziiert werden
  - `if (Sys.isMacOSX()) { registerQuitHandler()... }` funktioniert also *nicht!*
- Lösung: Reflection
- "Sys"-Bibliothek bietet fertige Lösung:
  - `Java1?Handler` deklarieren portable Interfaces
  - `Java1?Integration` sprechen Apple-Klassen direkt an
  - `Java1?Adapter` binden (nicht öffentliche) Integration-Klassen auf Mac OS X per Reflection ein

Portables Einbinden für Java 1.3

```
import com.muchsoft.util.mac.*;

public class AWTPortabel extends Frame
                         implements Java13Handler {
  public AWTPortabel() {
    ...
    Java13Adapter.registerJava13Handler( this );
  }
  public void handleAbout() { ... }
  public void handleQuit()  { ... }
  ...
}
```

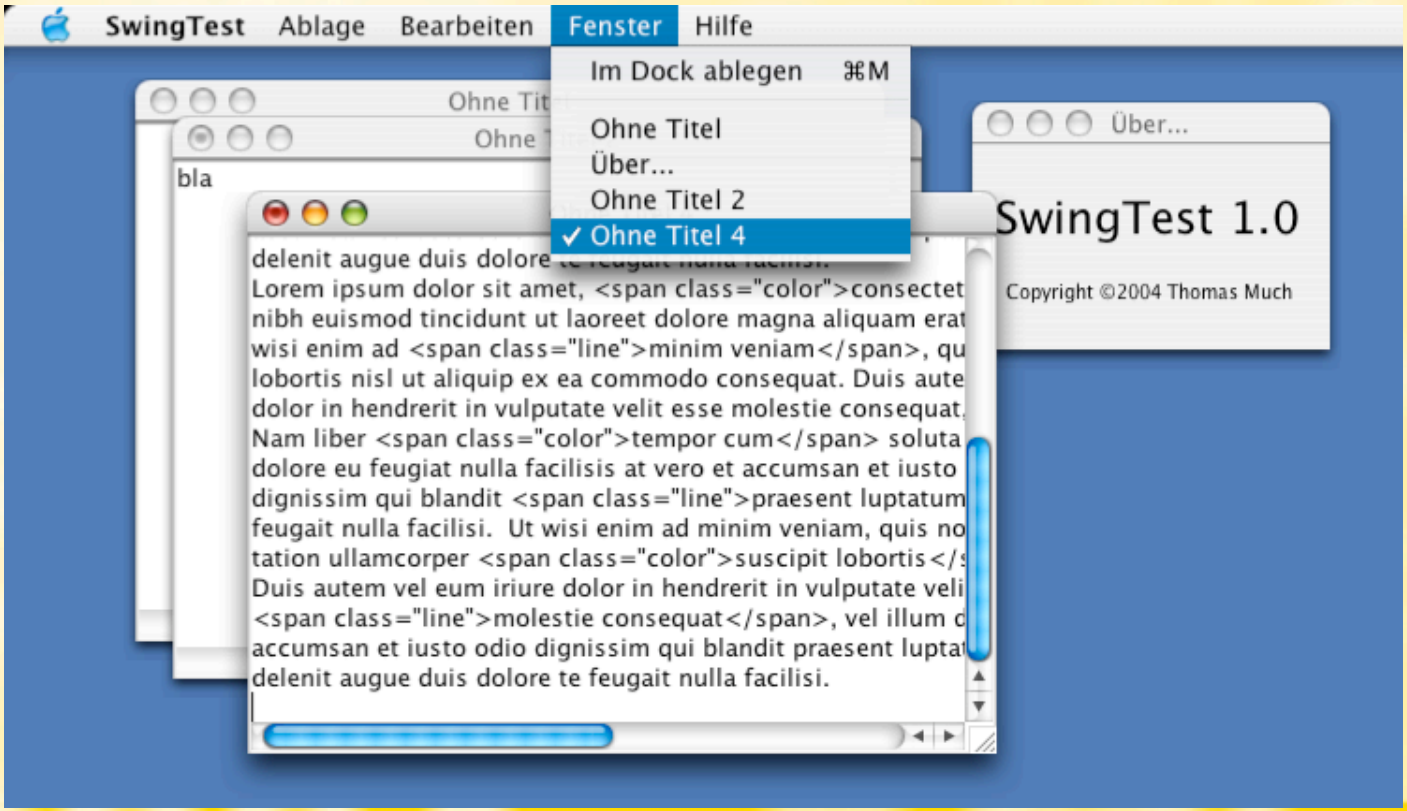
Portables Einbinden für Java 1.4

```
import com.muchsoft.util.mac.*;

public class SwingPortabel extends JFrame
                           implements Java14Handler {
  public SwingPortabel() {
    ...
    Java14Adapter.registerJava14Handler( this );
  }
  public void handleAbout( EventObject e ) { ... }
  public void handleQuit( EventObject e )  { ... }
  ...
}
```

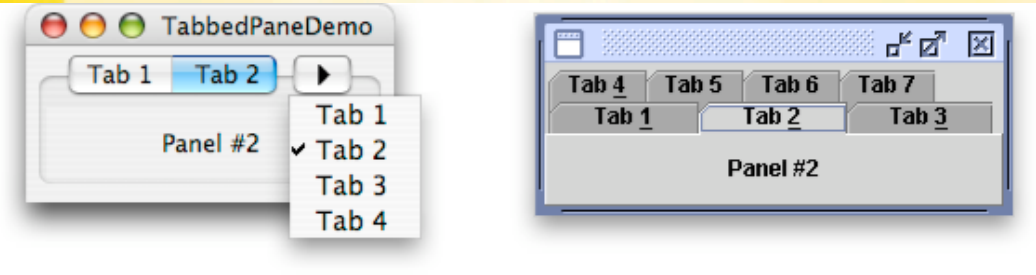
Human Interface Guidelines (1)

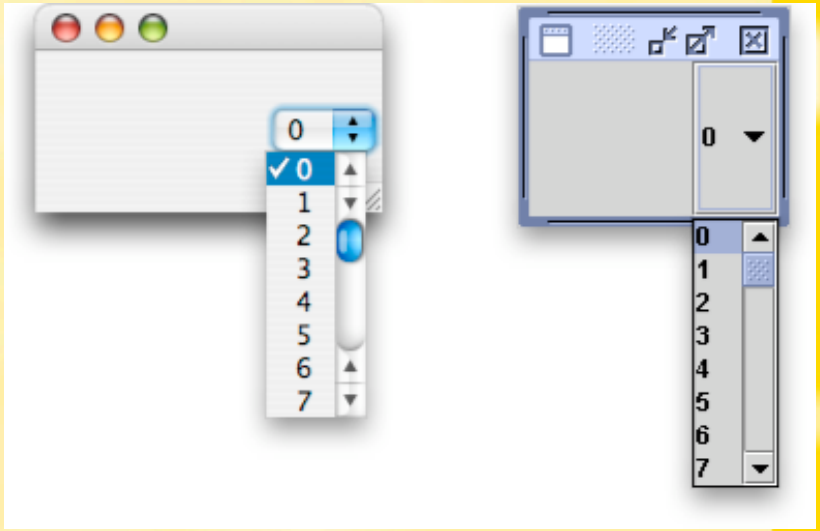
- von Anfang an für Mac OS festgelegt, für Aqua aktualisiert
- u.a. Menüs & Menüpunkte
- z.B. Fenster-Menü (der Mac kennt kein MDI!):



Human Interface Guidelines (2)

- Problem: Aqua-L&F beachtet im Zweifel HIG und nicht Swing-Richtlinien (zumindest meistens)
  - nur bedingt portabel ⇨ unbedingt testen!

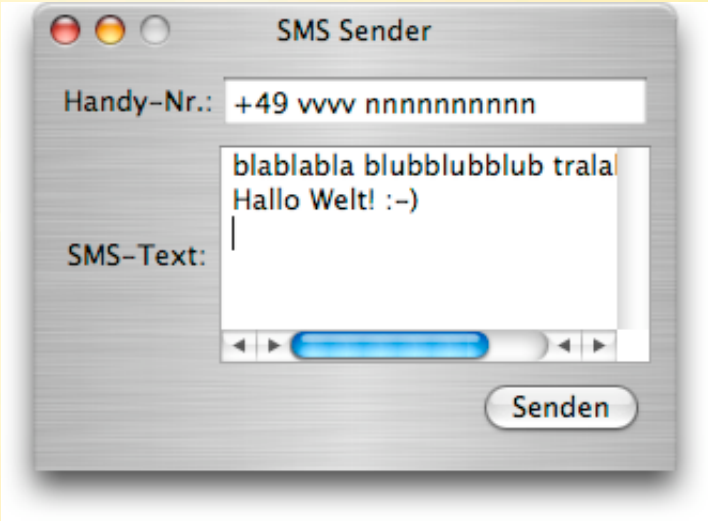




- bessere Beachtung der HIG mit Aqua-Ergänzung "Quaqua"
  - ⇨ http://www.randelshofer.ch/quaqua/

Brushed Metal

- neueres Aussehen, zusätzlich zu Aqua
- Property `apple.awt.brushMetalLook` auf `true` setzen



- nur beim AWT (Frame) verfügbar, kein komplettes L&F!
- ab Java 1.4, funktioniert erst ab Mac OS X 10.3 richtig
- sollte nur sparsam (beim Hauptdialog) eingesetzt werden

Ausblick

- Application-Framework
  - nur für Menüleisten und Multi-Dokument-Verwaltung
  - für AWT und Swing (und SWT?)
  - optimiert für Windows, Linux, Mac OS X
  - einheitliche Schnittstelle für alle Java-Versionen (ab 1.3)
  - eigenes java.net-Projekt? oder an JDIC anbinden?
- andere Lösungen
  - z.B. MRJAdapter (aber nur für Mac-Optimierung)
  - ⇨ http://www.roydesign.net/mrjadapter/

Zusammenfassung

Optimierung von Java-Apps für Mac OS X?

- von Mac-Nutzern erwartet, da höhere Ansprüche an Bedienbarkeit von Software
  - davon können auch Windows-Nutzer profitieren!
- leider Unterschiede zwischen Java 1.3 und 1.4
- mit Mac OS X-Testrechner kein Problem
- auch ohne Mac-Testrechner möglich:
  - JAR-Archive, Programmpakete, Basis-Systemintegration
  - aber: GUI muss getestet werden!
    - Aqua-L&F beachtet Apple-Richtlinien
    - sonst am besten Metal-L&F einsetzen, aber ... s.o.

Vielen Dank!

thomas@muchsoft.com

www.muchsoft.com

"Java für Mac OS X"

Galileo Computing 2005

630 Seiten, mit CD-ROM

ISBN 3-89842-447-2