Bean Mappings mit MapStruct 1.2

Einfach, schnell und sicher!

Thomas Much
@thmuch

Über…

Thomas Much

Freelancer, Hamburg

@thmuch
#JAX2018

Agile Developer Coach & Software Developer (Java et al.)

Mag coole kleine Tools & Libs
(und erzählt gerne darüber 😉)

Java und Beans, eine lange Geschichte…

1995

1997

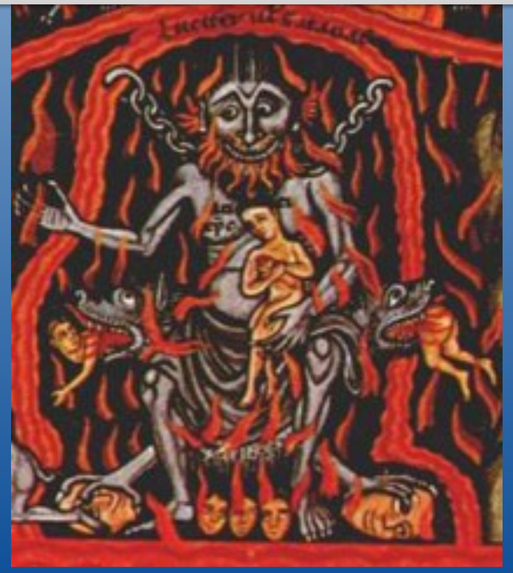
JavaBeans

1998

Form Beans

1999

DTOs



2000

POJOs

2005

JAXB Entities

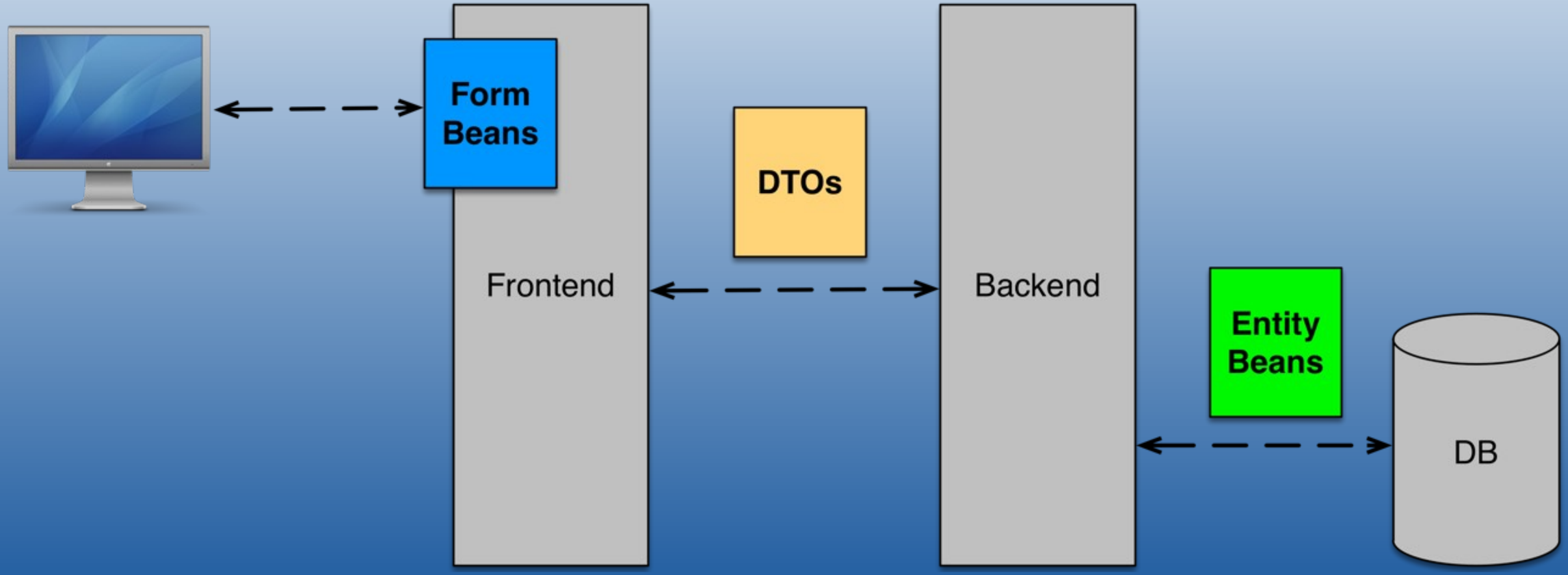
2006

JPA Entities

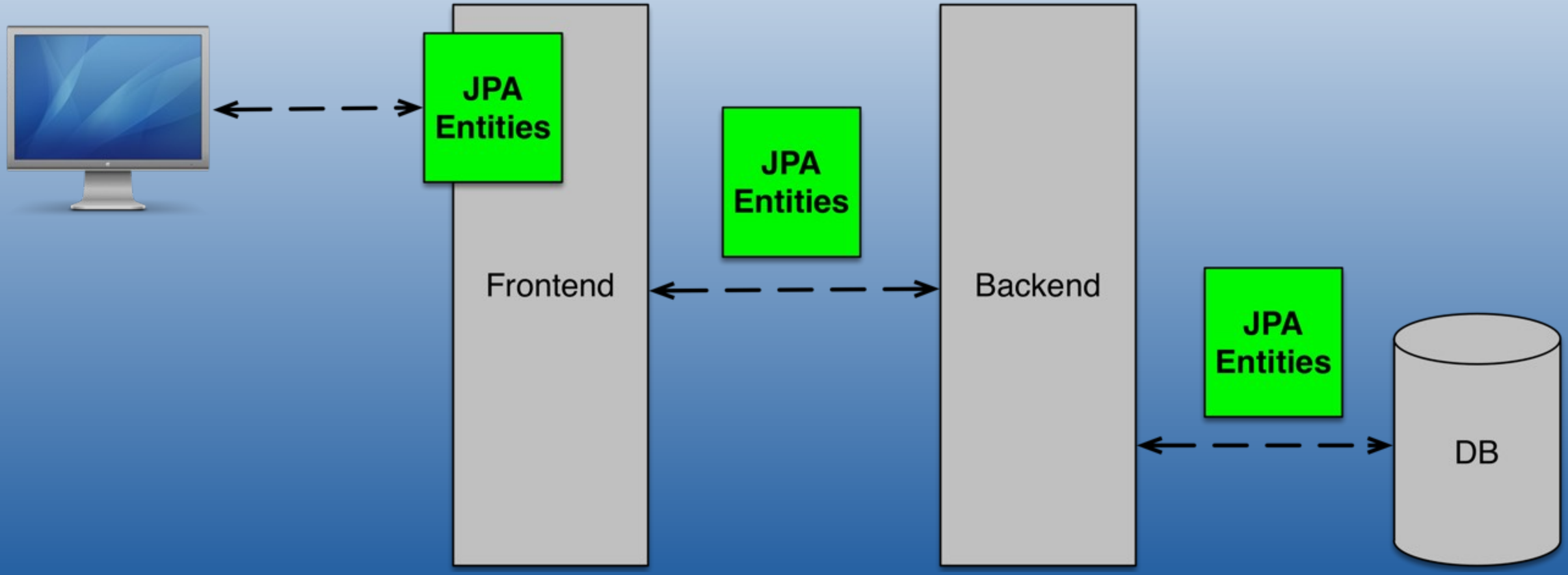
2015

Typsicheres, schnelles Bean-Mapping!

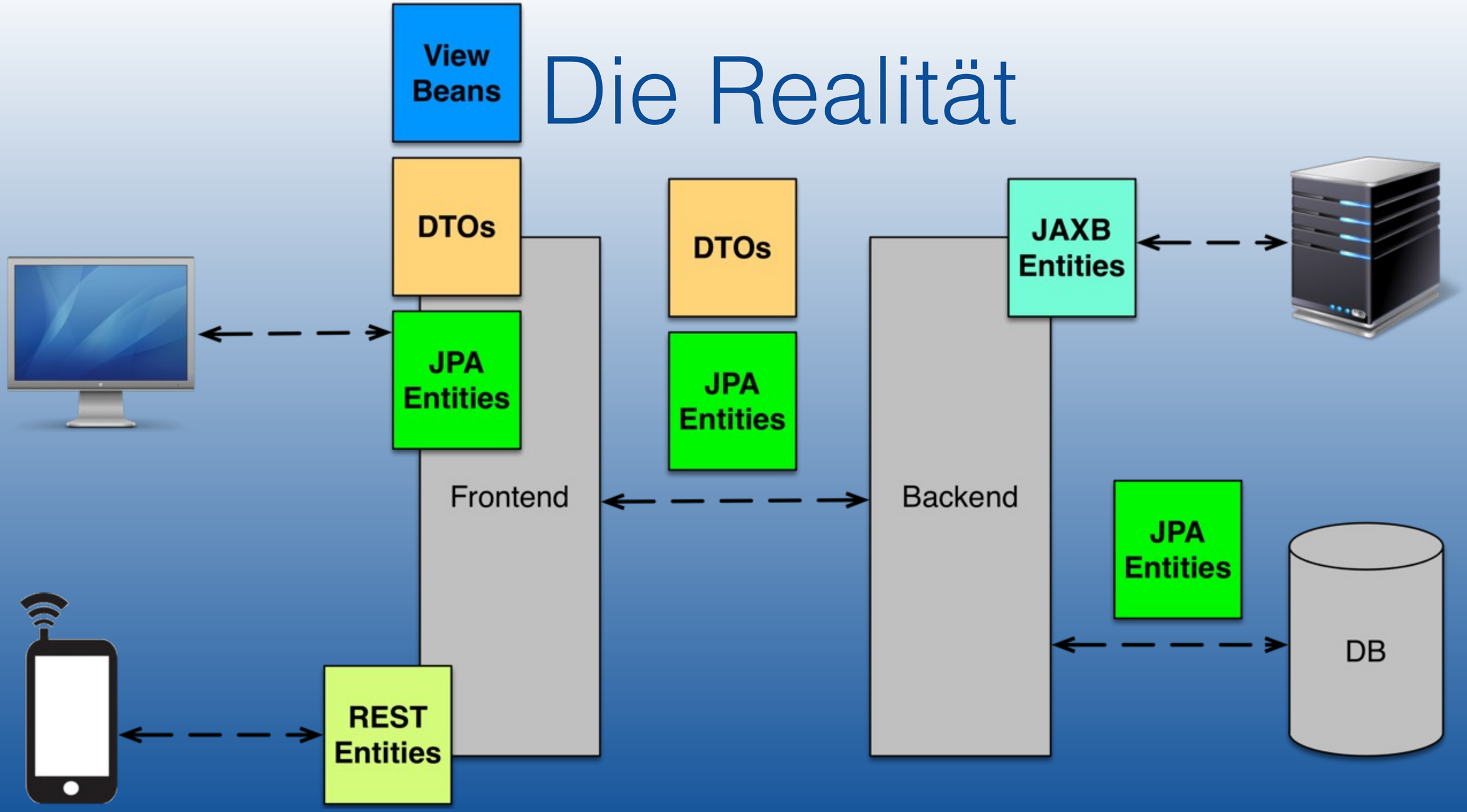
Klassische Architekturen (J2EE)



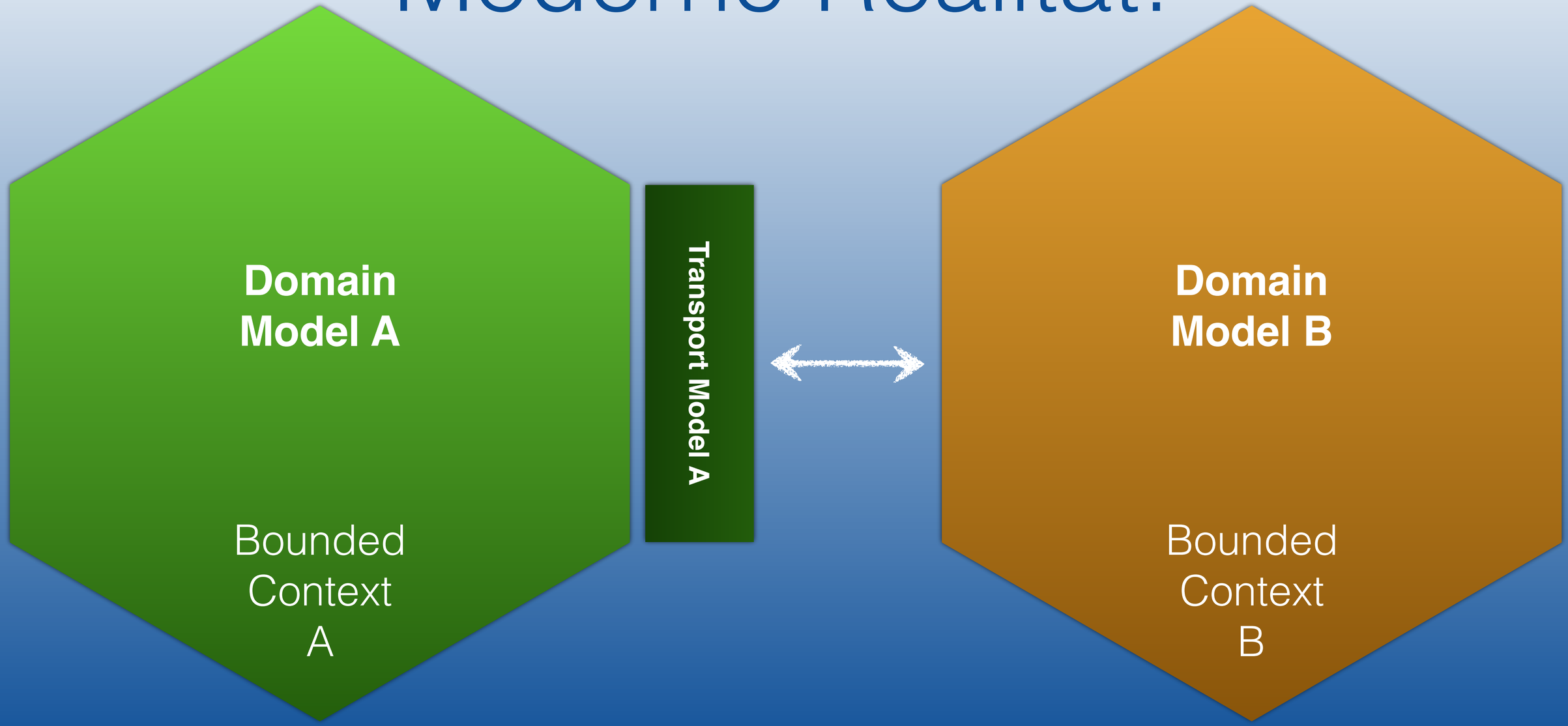
Was Java EE & Co. versprechen



Die Realität



Moderne Realität!



Doch wieder DTO-Hölle?

DTOs, Entities, ViewObjects… lassen sich generieren.

IDE → Generate, Lombok @Data

Schreiben der Beans ist kein Problem (mehr).

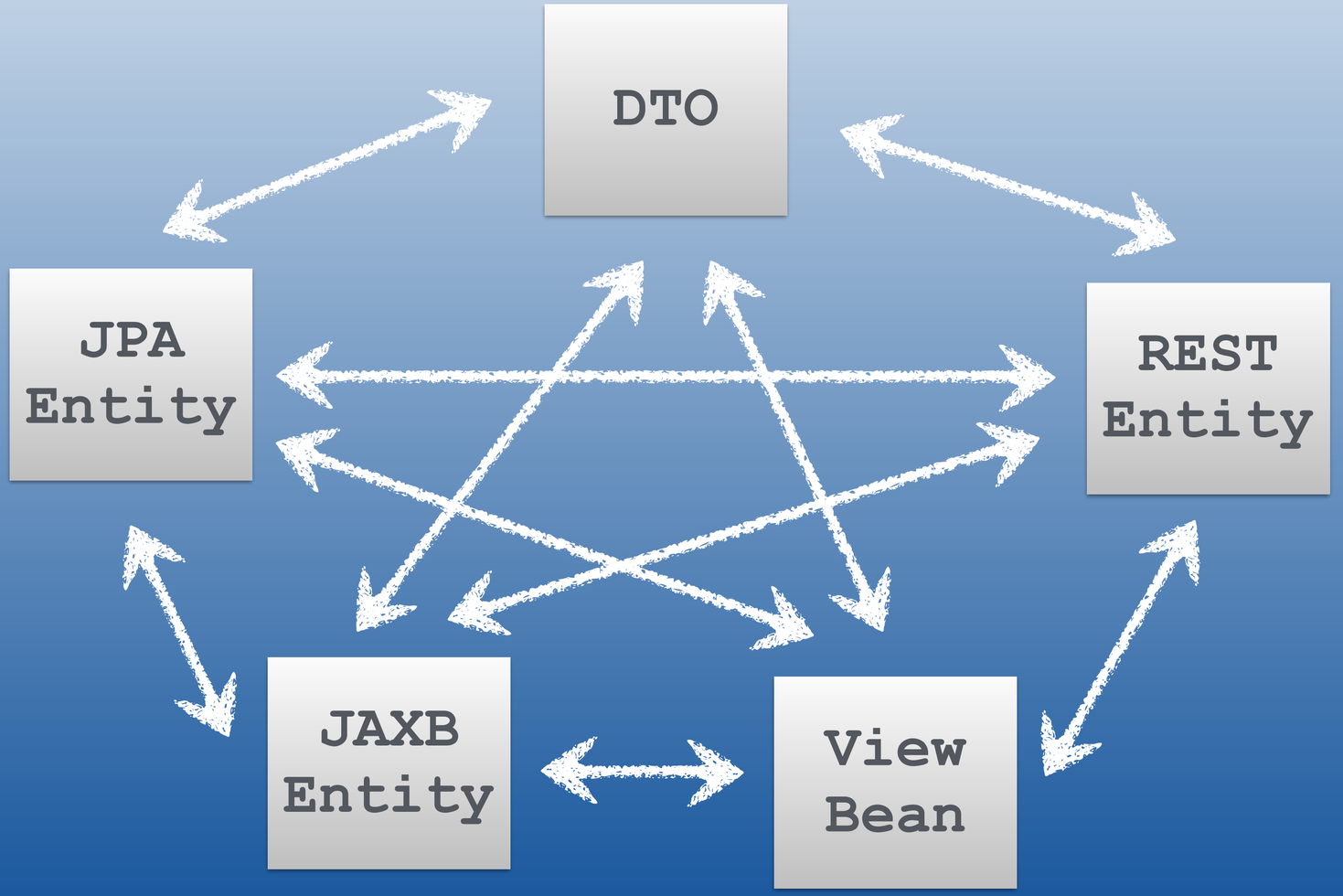
Eher das Schreiben des Mapping-Codes…

Das erforderliche Mapping…

```
@Entity
@Table(name = "CUSTOMERS")
public class Customer {

  @Id
  Long id;

  long customerId;

  String name;

  @Enumerated
  Title title;

  … getters and setters! …
}
```

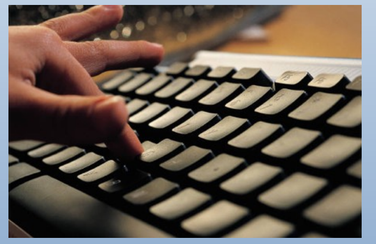
```
public class CustomerDTO {


  long id;

  String customerId;

  String name;


  String title;

  … getters and setters! …
}
```

… kann aufwendig werden

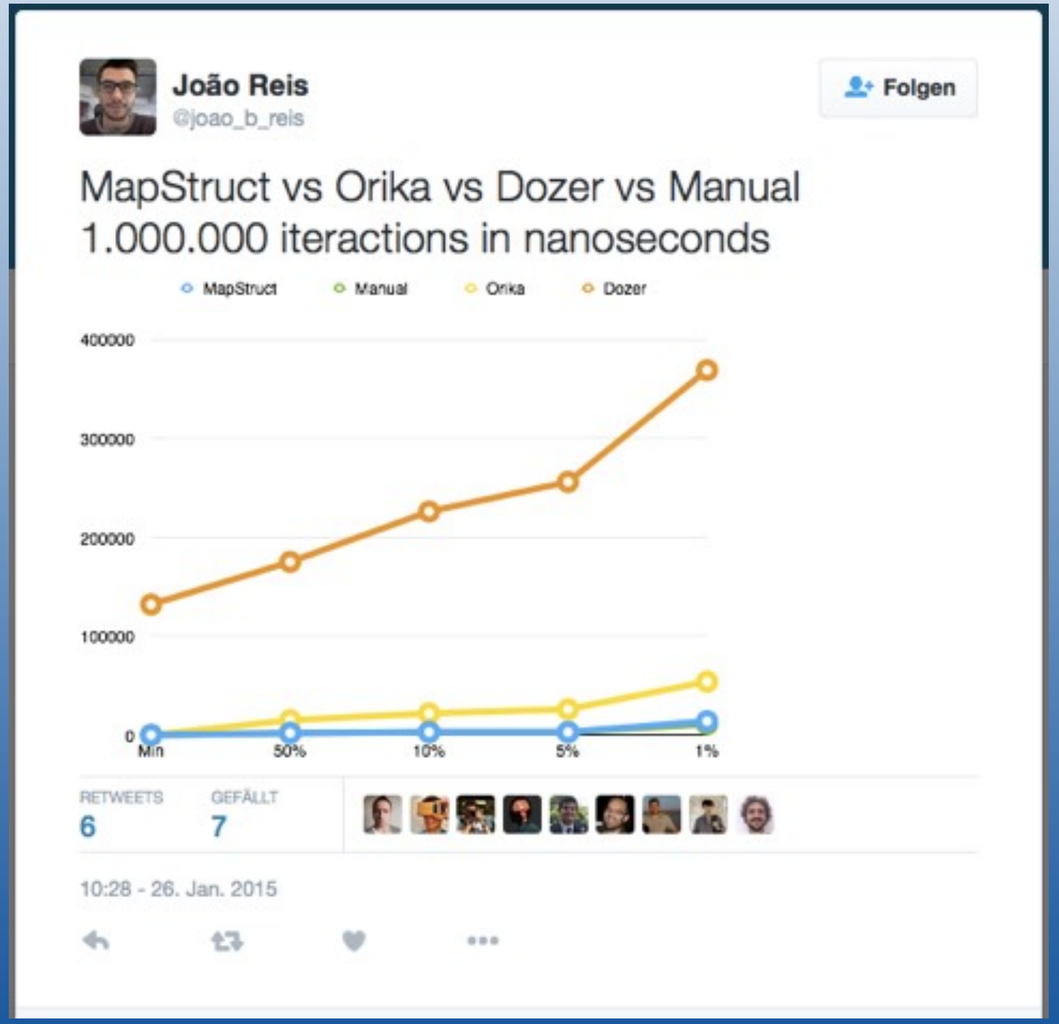


Mapping-Implementation – wie?

- Selber programmieren
  - Alle Getter/Setter von Hand aufrufen
  - Eigene Reflection-Routinen
- Reflection-basierte Bibliotheken:
  - Apache BeanUtils, Dozer, …
- Probleme: Aufwand, fehlende Typsicherheit, Performance …



Performance… Performance!



https://twitter.com/joao_b_reis/status/559780053979250688

- Annotationsprozessor, der Mapping-Code (Quelltext!) generiert.
- Keine Reflection!
- Typsicher und *schnell*.

- Mind. Java 6, spezielle Unterstützung für Java 8+
- Minimale Laufzeitabhängigkeit (< 20 K)
  bzw. gar keine (je nach Komponentenmodell)

Versionen

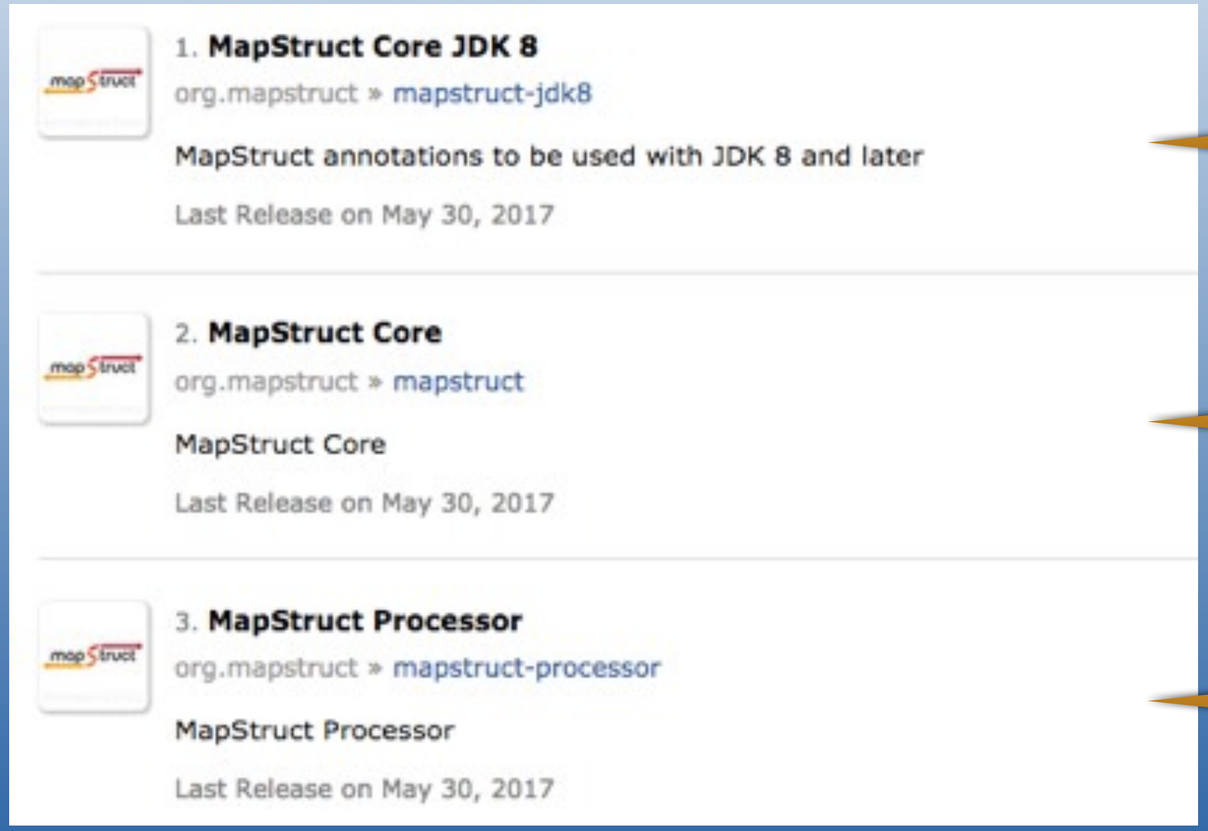
MapStruct 1.0 (2015)

MapStruct 1.1 (2016)

MapStruct 1.2 (2017)

MapStruct 1.3 (Beta)

JARs (Dependencies)



Java 8+

Java 6 / 7

Java 6+, unnötig zur Laufzeit

http://mvnrepository.com/artifact/org.mapstruct

Beispiel – JPA-Entity nach DTO

```
@Entity
@Table(name = "CUSTOMERS")
public class Customer {

  @Id
  Long id;

  long customerId;

  String name;

  @Enumerated
  Title title;

  … getters and setters! …
}
```

```
public class CustomerDTO {


  long id;

  String customerId;

  String name;


  g title;

  ers and setters! …
}
```

Alle public-Properties
sollen gemappt werden
(auch aus Oberklassen)

Wünsch dir was!

```
public interface CustomerMapper {

   CustomerDTO customer2DTO(Customer customer);

}
```

Wünsch dir was!

```
@Mapper(componentModel = "cdi")
public interface CustomerMapper {

  CustomerDTO customer2DTO(Customer customer);

}
```

```
@Inject
private CustomerMapper mapper;
…

CustomerDTO dto =
  mapper.customer2DTO( customer );
```

Die Implementation generieren

- Wie erzeugen wir aus dem Interface die Mapping-Implementation?
- Speichern  z.B. in Eclipse.
- Genauer: **Compiler aufrufen** (z.B. via Maven/Gradle)
- **Annotationsprozessor erledigt den Rest!**

Generierter Quelltext

```
@Generated(
        value = "org.mapstruct.ap.MappingProcessor",
        date = "2017-06-15T13:11:17+0200",
        comments = "version: 1.2.0.Beta2, compiler: Eclipse JDT (IDE) ..."
)
@ApplicationScoped
public class CustomerMapperImpl implements CustomerMapper {

    @Override
    public CustomerDTO customer2DTO(Customer customer) {

        if (customer == null) {
            return null;
        }

        CustomerDTO customerDTO = new CustomerDTO();

        customerDTO.setCustomerId(String.valueOf(customer.getCustomerId()));

        if (customer.getId() != null) {
            customerDTO.setId(String.valueOf(customer.getId()));
        }

        customerDTO.setName(customer.getName());

        if (customer.getTitle() != null) {
            customerDTO.setTitle(customer.getTitle().name());
        }

        return customerDTO;
    }
}
```

Thread-sicher.

Clean Code.
Leicht zu lesen.

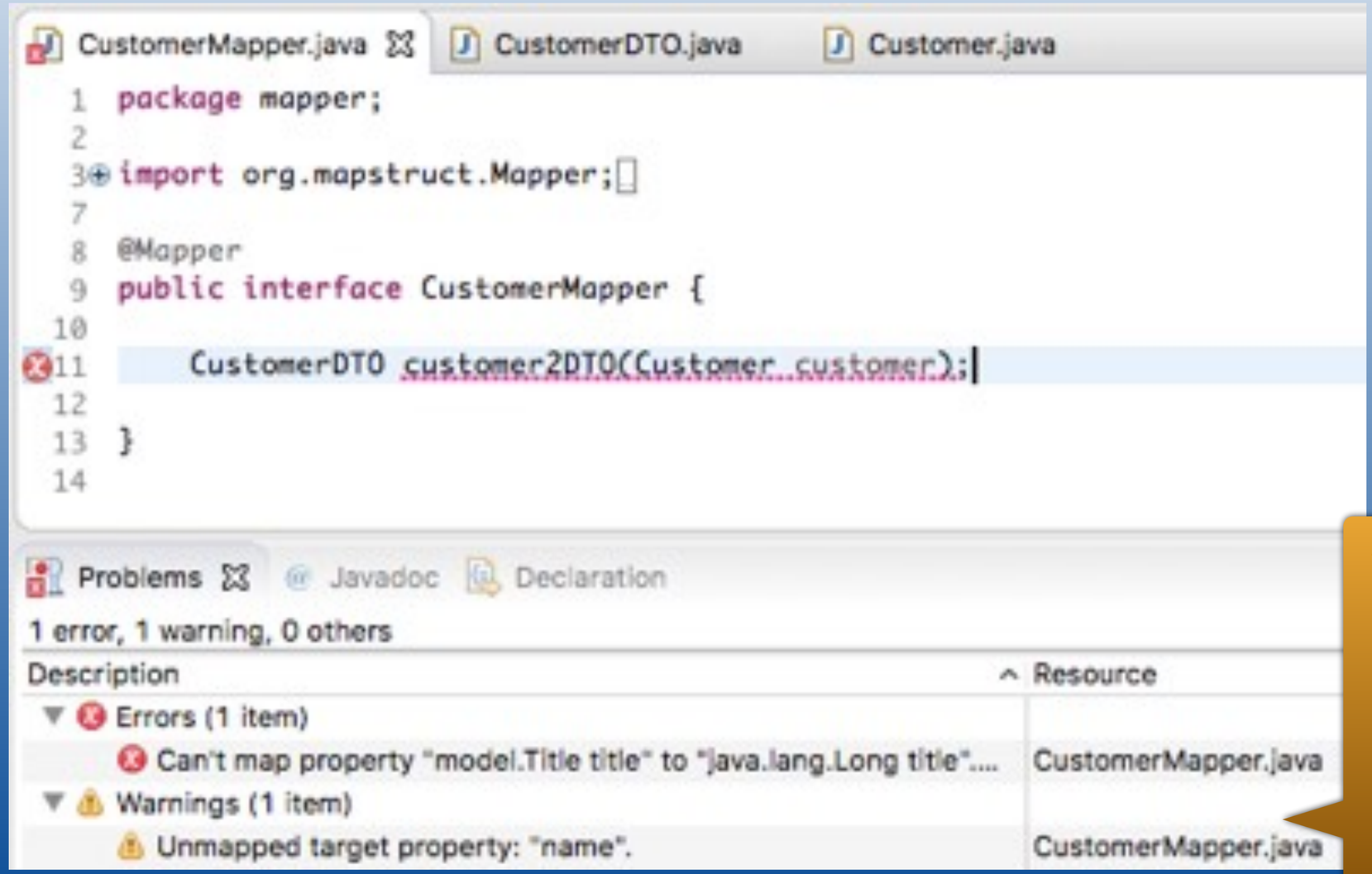
@Generated-Attribute
sind optional

Kein Injection-Container? Kein Problem!

```
@Mapper
public interface CustomerMapper {

   CustomerMapper INSTANCE =
      Mappers.getMapper( CustomerMapper.class );

   CustomerDTO customer2DTO(Customer customer);

}
```

```
CustomerDTO dto =
   CustomerMapper.INSTANCE.customer2DTO( customer );
```

Typ- (& Wert-) Sicherheit



WARN / ERROR / IGNORE

Policy global setzen
oder pro Mapper
(oder Default verwenden)

Grundlegende Mappings

```
@Mapper
public interface CustomerMapper {

   @Mappings({
      @Mapping(target="custNo", source="customerId"),
      @Mapping(target="password", ignore=true),
      @Mapping(target="lastLogin", dateFormat="dd.MM.yyyy")
   })
   CustomerDTO customer2DTO(Customer customer);

}
```

Enum-Werte abbilden

```
@Mapper
public interface CustomerMapper {

    @ValueMapping(target="HR", source="MR")
    @ValueMapping(target="FR", source="MRS")
    @ValueMapping(target="FR", source="MS")
    @ValueMapping(target="<NULL>", source="<ANY_UNMAPPED>")
    DtoTitle title2DtoTitle(Title title);

    CustomerDTO customer2DTO(Customer customer);

}
```

Siehe `MappingConstants`

Viele implizite Umwandlungen

Primitive Datentypen ↔ Wrapper (inkl. null-Checks)

Number-Typen (incl. Big…) ↔ andere Number-Typen & Präzisionen

String ↔ so ziemlich alles (inkl. Enums & Date/Number-Formate)

Date/Calendar ↔ Joda ↔ Java 8 Date/Time

JAXB ↔ Elemente, Collections

Objekt-Referenzen

```
@Mapper
public interface CustomerMapper {

   CustomerDTO customer2DTO(Customer customer);
}
```


```
@Entity
public class Customer {

   Address address;

   … other properties …
   … getters and setters! …
}
```

Referenz-Mapping anpassen

```
@Mapper(disableSubMappingMethodsGeneration=true)
public interface CustomerMapper {

  CustomerDTO customer2DTO(Customer customer);

  AddressDTO address2DTO(Address address);

}
```

Andere Mapper einbinden

```
@Mapper(uses=AddressMapper.class)
public interface CustomerMapper {

  CustomerDTO customer2DTO(Customer customer);
}

@Mapper
public interface AddressMapper {

  AddressDTO address2DTO(Address address);
}
```

Eigener Mapping-Code

```
@Mapper
public interface CustomerMapper {

   CustomerDTO customer2DTO(Customer customer);

   default AddressDTO address2DTO(Address address) {
      AdressDTO dto = new AddressDTO();
      ... custom mapping code ...
      return dto;
   }
}
```

Alternativ mit einer abstrakten Klasse

Alten Mapping-Code einbinden

```
@Mapper(uses=LegacyAddressMapper.class)
public interface CustomerMapper {
  CustomerDTO customer2DTO(Customer customer);
}

public class LegacyAddressMapper {
  public AddressDTO address2DTO(Address address) {
    ... custom mapping code ...
    return dto;
  }
}
```

Default Constructor oder *statische* Mapping-Methode nötig.

Mehrere Quell-Parameter

```
@Mapper
public interface PrintLabelMapper {

   @Mapping(target="name", source="customer.name")
   @Mapping(target="street", source="address.street")
   @Mapping(target="city",   source="address.city")
   PrintLabelDTO customerAddress2DTO(
                       Customer customer, Address address);
}
```

Pfad-Ausdrücke können generell in "source" und "target" verwendet werden.

Updates & Reverse-Mappings

```
@Mapper
public interface CustomerMapper {

    CustomerDTO customer2DTO(Customer customer);

    void updateCustomerDTO(Customer customer,
            @MappingTarget CustomerDTO dto);

    Customer dto2Customer(CustomerDTO dto);
}
```

Alternativ Target-Typ als Rückgabe

Max. 1 @MappingTarget!

Mappings wiederverwenden

```
@Mapper
public interface CustomerMapper {

   @Mapping( … )
   CustomerDTO customer2DTO(Customer customer);

   @InheritConfiguration(name="customer2DTO")
   void updateCustomerDTO(Customer customer,
            @MappingTarget CustomerDTO dto);

   @InheritInverseConfiguration(name="customer2DTO")
   Customer dto2Customer(CustomerDTO dto);
}
```

Context-Parameter

```
@Mapper
public interface CustomerMapper {

    CustomerDTO customer2DTO(Customer customer,
                             @Context Locale locale);

    AddressDTO address2DTO(Address address,
                           @Context Locale locale);
}
```

Aufruf aus eigenem Code

Wird in der generierten Implementation durchgereicht!

Eigene Mapper mit TargetType

```
@ApplicationScoped
public class JpaEntityMapper {

   @PersistenceContext
   private EntityManager manager;

   public <T extends BaseEntity> T resolve(Long id,
                                 @TargetType Class<T> entityClass) {
      return (id != null) ?
               manager.find(entityClass, id) : null;
   }
}
```

Bean-Fabriken

```
public class DtoFactory {
   public CustomerDTO createCustomerDTO() {
      return new CustomerDTO();
   }
}

public class EntityFactory {
   public <T extends BaseEntity>
      T createEntity(@TargetType Class<T> entityClass) {
         return entityClass.newInstance();
   }
}
```

Streams & Collections von Beans

```
@Mapper
public interface CustomerMapper {

    CustomerDTO customer2DTO(Customer customer);

    List<CustomerDTO> list2List(List<Customer> customer);
    List<CustomerDTO> array2List(Customer[] customer);
    CustomerDTO[] set2Array(Set<Customer> customer);

    List<CustomerDTO> stream2List(Stream<Customer> cust);
    Stream<CustomerDTO> set2Stream(Set<Customer> cust);
}
```

Streams & Collections von Objekten

```
@Mapper
public interface StreamCollectionMapper {

    Set<Long> int2Long(Stream<Integer> stream);
    Stream<Long> int2Long(List<Integer> list);

    @IterableMapping(dateFormat="dd.MM.yyyy")
    List<String> date2String(List<Date> list);

    @MapMapping(valueDateFormat="dd.MM.yyyy")
    Map<String,String> map2map(Map<Long,Date> map);
}
```

Exceptions

Jede *Checked Exception*,

die nicht an der Mapping-Methode deklariert ist,

wird als RuntimeException weitergeworfen.

In Mapping-Aufrufe einklinken

- **Callbacks**
  - @BeforeMapping, @AfterMapping
- **Decorators**
  - @DecoratedWith wird mitgeliefert
    – aber besser CDI-Decorators verwenden.

MapStruct SPI

- **Accessor Naming Strategy** (Zugriffsmethoden) anpassen
  - z.B. Fluent API statt Getter/Setter
- Mapping-**Ausnahmen**
  - Bestimmte Typen vom automatischen Mapping ausschließen

Viele weitere Möglichkeiten

- Konstante Werte & Java-Ausdrücke als "source".
- Default-Werte, wenn die Quelle null ist.
- Reihenfolge von Setter-Aufrufen ("dependsOn").
- Angepasste null-Prüfungen & Default-null-Werte.
- Zentrale Mapping-Konfigurationen ("config", @MapperConfig).
- Auswahl mehrdeutiger Mapping-Methoden per Qualifier oder Name

Integration in Build-Tools

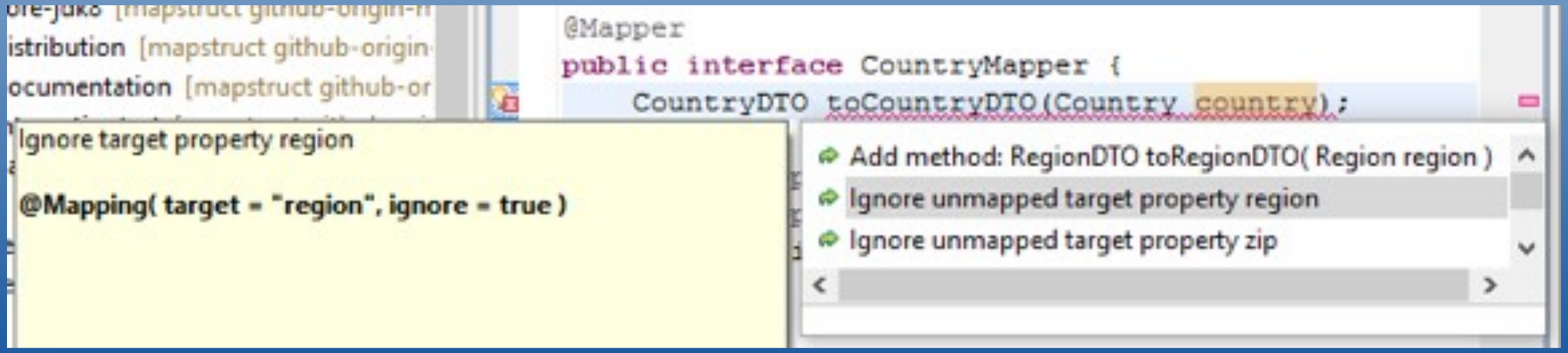
- javac
  - SPI – MapStruct einfach auf den Classpath packen!

Besser: Processorpath

- Maven, Ant, Gradle
  - MapStruct-Dokumentation enthält (einfache) Setup-Anleitung:
    - MapStruct-Dependency hinzufügen, Annotationsprozessor aktivieren.

Eclipse Plug-in

- Code Completion
- Quick Fixes
- *https://github.com/mapstruct/mapstruct-eclipse*



IntelliJ Plug-in

- Code Completion
- Goto Declaration
- Find Usages

```
blic interface CarMapper {

    CarMapper INSTANCE = Mappers.getMapper( CarMapper.class );

    @Mappings({
        @Mapping(source = "numberOfSeats", target = "seatCount"),
        @Mapping(source = "manufacturingDate", target = "manufacturingYear")
    })
    CarDto carToCarDto(Car car);

    @InheritInverseConfiguration
    Car carDtoToCar(CarDto carDto);
```

- *https://plugins.jetbrains.com/plugin/10036-mapstruct-support*

Neu in MapStruct 1.2

- Unterstützung für Java 8 Streams.
- Mappings für public-Felder.
- Automatische Generierung von Sub-Mapping-Methoden.
- Integration mit Project Lombok.
- Unterstützung für Java 9 (experimentell).

Neu in MapStruct 1.3 (Beta)

Unterstützung für Immutables via Builder.

Implizites Mapping String ↔ java.util.Currency

…?

Lombok vs. MapStruct

```
@Mapper
public interface CustomerMapper {
   CustomerDTO customer2DTO(Customer customer);
}
```

```
@Data
public class CustomerDTO {
   private long id;
   private String customerId;
   private String name;
   private String title;
}
```

```
@Entity
@Table(name = "CUSTOMERS")
public class Customer {
   @Id
   Long id;

   ... Getter und Setter! ...
}
```

Lombok vs. MapStruct

Manipuliert AST-Bytecode

```
@Mapper
public interface CustomerMapper {
   CustomerDTO customer2DTO(Customer customer);
}
```

```
@Data
public class CustomerDTO {
  private long id;
  private String customerId;
  private String name;
  private String title;
}
```

```
@Entity
@Table(name = "CUSTOMERS")
public class Customer {
   @Id
   Long id;

   … Getter und Setter! …
}
```

Lombok vs. MapStruct

Manipuliert AST-Bytecode

```
@Mapper
public interface CustomerMapper {
   CustomerDTO customer2DTO(Customer customer);
}
```

Generiert

CustomerMapperImpl.java

Sourcecode!

```
@Data
public class CustomerDTO {
   private long id;
   private String customerId;
   private String name;
   private String title;
}
```

```
@Entity
@Table(name = "CUSTOMERS")
public class Customer {
   @Id
   Long id;

   ... Getter und Setter! ...
}
```

MapStruct – Fazit

- Hervorragend produktiv einsetzbar!
- Generierter Code ist schnell, einfach und *lesbar*.
- Flexible (Custom-)Mappings.
- Einfache Einbindung in div. Komponentenmodelle.
- Ausführliche und gut geschriebene Dokumentation.

Neugierig geworden?

http://mapstruct.org/

https://github.com/mapstruct/mapstruct-examples

😀

Die Entwickler:

Danke!

Gunnar Morling is the founder of the MapStruct project.

He is a long-time Java developer and open-source committer. He's leading the Debezium project at Red Hat, part of the Hibernate team and the spec lead for Bean Validation 2.0 (JSR 380). You can follow him on Google+ and Twitter or check out his Blog.

Andreas Gudian was the first committer to join Gunnar in his efforts.

He is an experienced developer and architect for enterprise Java applications, where MapStruct can make a real difference. Andreas contributes to various open source projects and is also committer at the Apache Maven project.

Sjaak Derksen, enthusiastic first-hour user of MapStruct

He has well over 15 years of experience in Java / JEE development as architect, technical lead, developer and tester in the domain of Telecommunications. Sjaak started working more recently on spatial subsurface data interchange (e.g. Inspire, GML) where he believes MapStruct can be a true asset, reducing the amount of repetitive, error-prone work.

Filip Hrisafov, lead of the MapStruct team

He is a young Java developer and consultant, who uses MapStruct to help him in his daily work on Java enterprise applications. He is passionate about Open Source and contributes to various open source projects.

Danke!

thomas@muchsoft.com
www.javabarista.de
@thmuch