MATHEMA campus 2015

Machst Du noch Reflection oder annotierst Du schon?

Bean-Mapping mit MapStruct

Thomas Much

thomas@muchsoft.com
www.muchsoft.com

20 Jahre Java, 18 Jahre Beans

1995 Java

1997 JavaBeans

1998 Form Beans

1999 DTOs

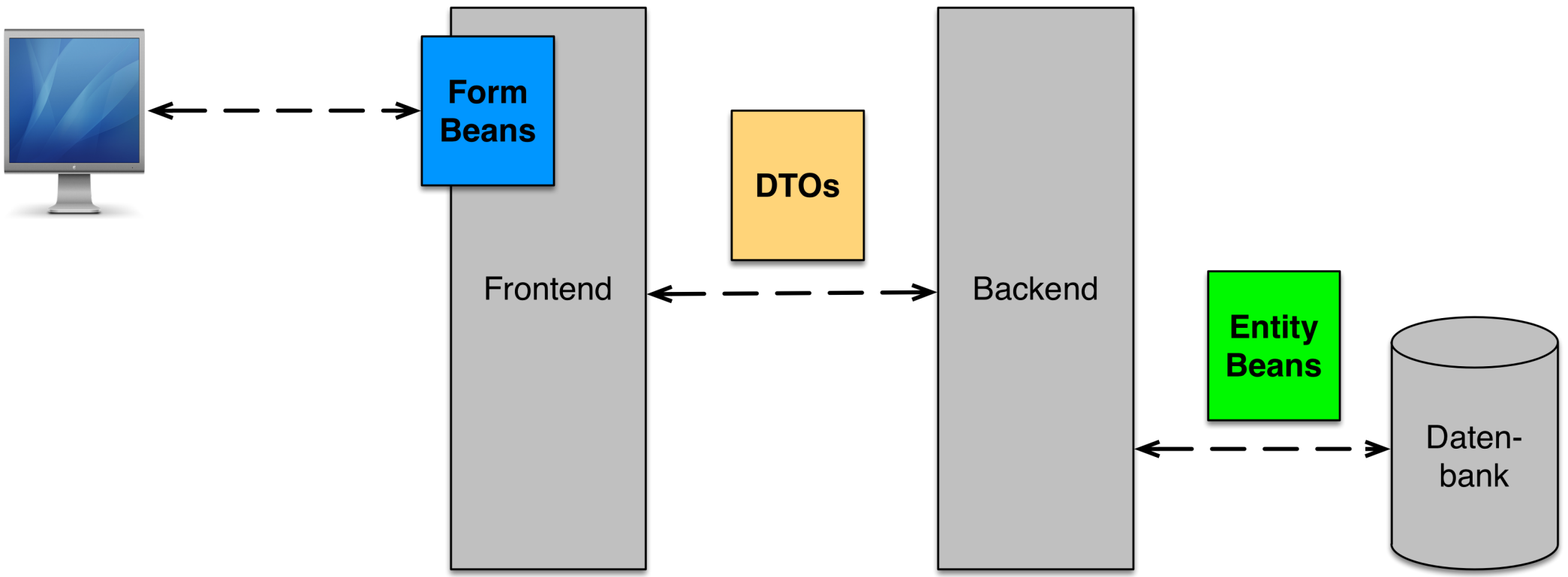
2000 POJOs

2005 JAXB-Entities

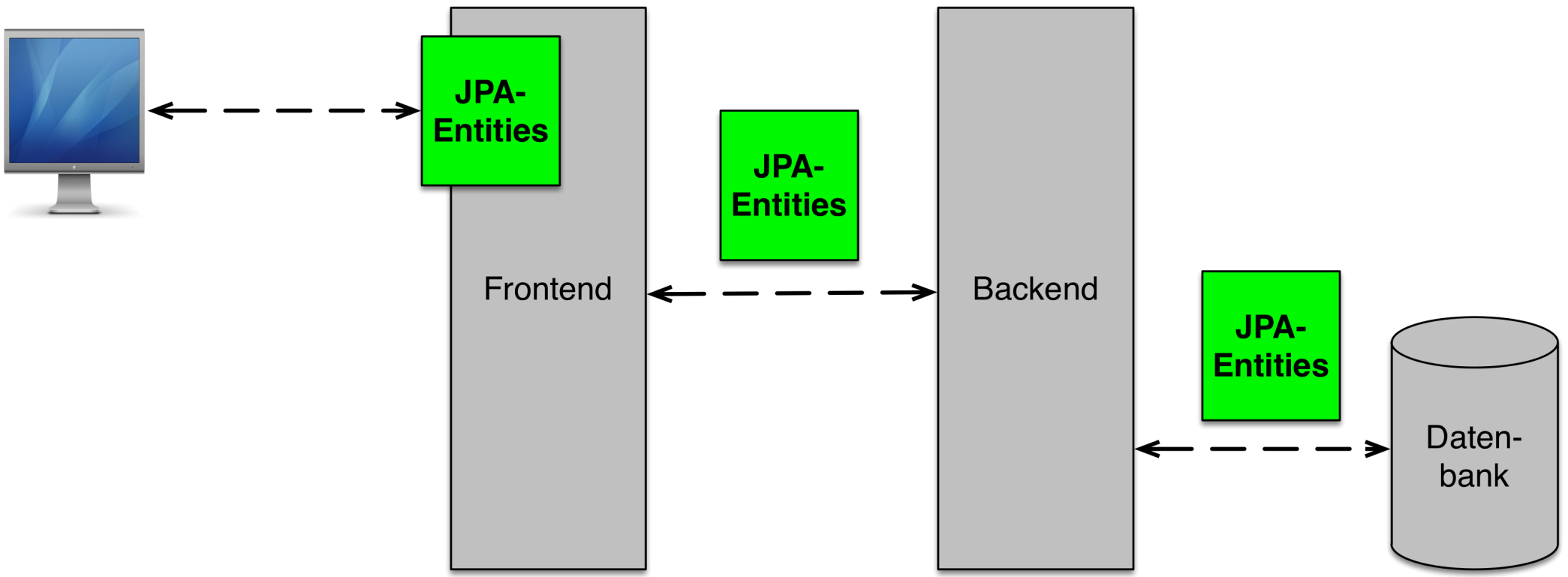
2006 JPA-Entities

2015 Typsicheres, schnelles Bean-Mapping!

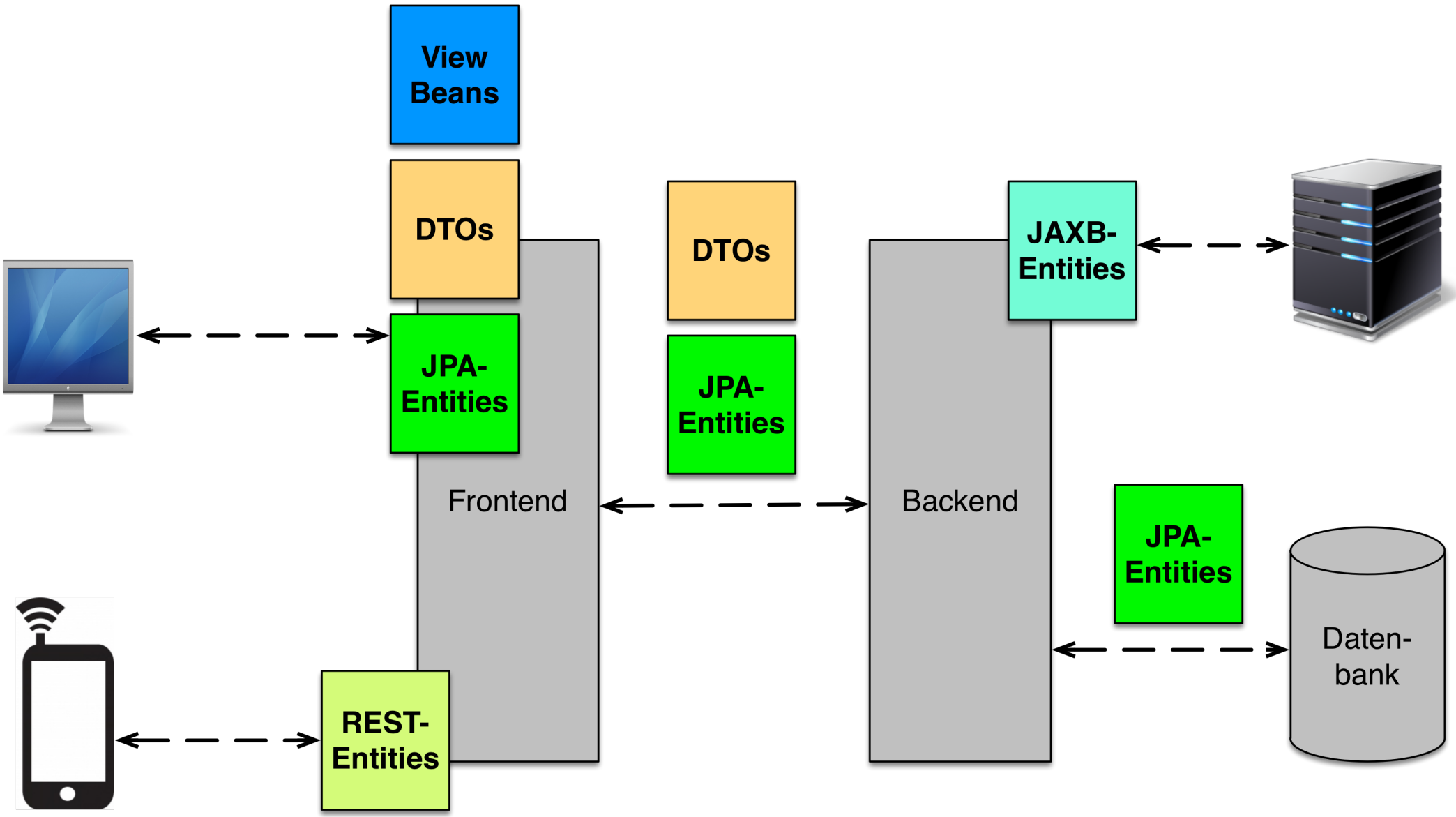
Alte Architekturen



Was Java EE 5+ und JPA versprechen



Die moderne Realität

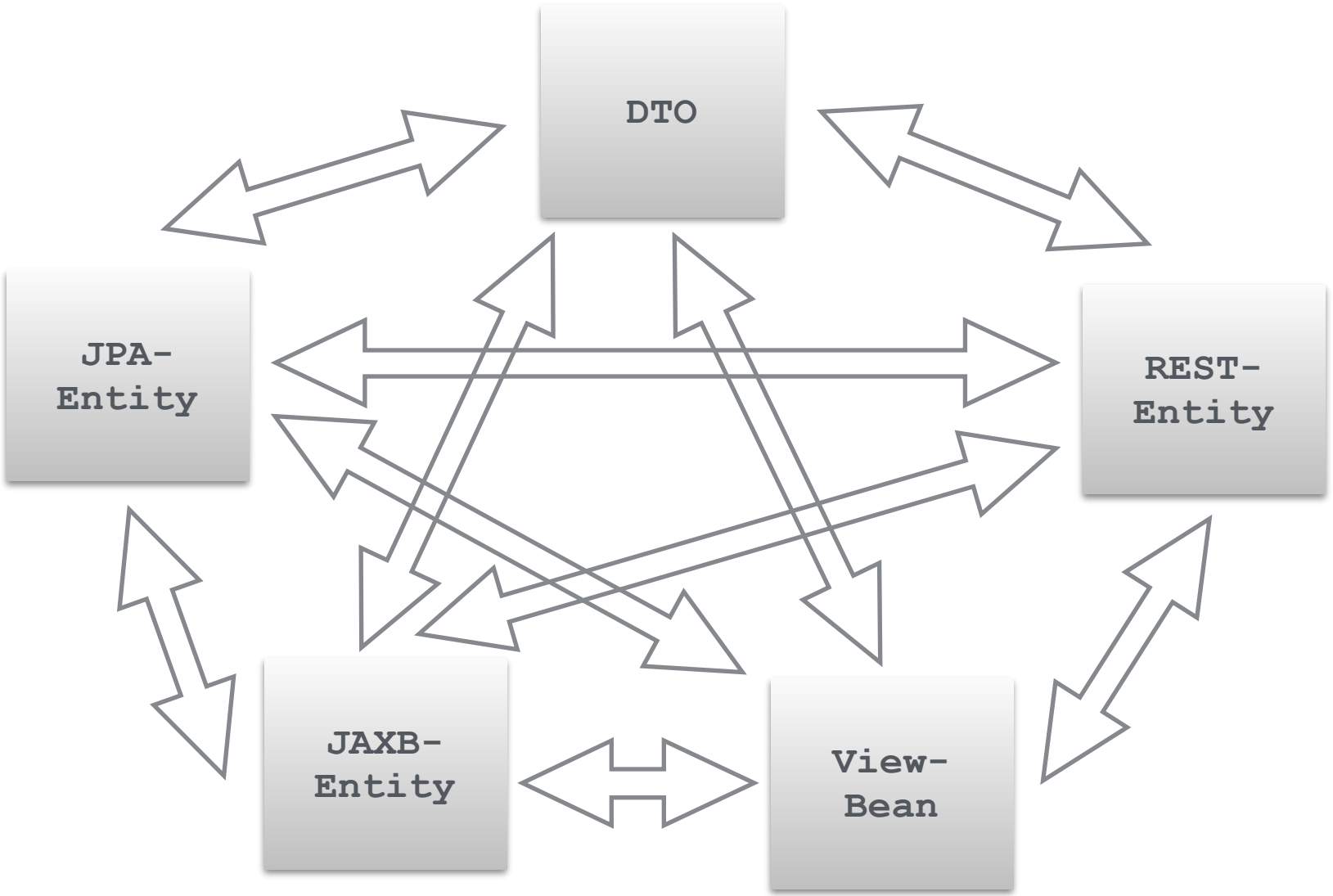


Das erforderliche Mapping …

```
@Entity
@Table(name = "KUNDEN")
public class Kunde {

  @Id
  Long id;

  long kundennummer;

  String name;

  @Enumerated
  KundenArt kundenart;

  … Getter und Setter! …
}
```

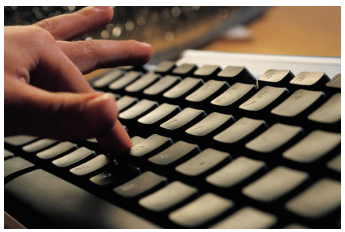
```
public class KundeDTO {

  long id;

  String kundennummer;

  String name;


  String kundenart;

  … Getter und Setter! …
}
```

… kann aufwendig werden



Mapping-Implementation – wie?

- Selbst programmiert
  - alle Getter/Setter von Hand aufrufen
  - eigene Reflection-Routinen



- Reflection-Bibliotheken:
  - Apache BeanUtils
    http://commons.apache.org/proper/commons-beanutils/
  - Dozer
    http://dozer.sourceforge.net/

- Probleme: Aufwand, Typunsicherheit, Performance …

MapStruct

- **MapStruct: *http://mapstruct.org/***

- Annotations-Prozessor, der Mapping-Code generiert
- Keine Reflection!
- Typsicher und schnell

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

Achtung, Beta!

- Seit 2013 in Entwicklung
- Aktuelle Version 1.0.0.Beta4
- Finale Version bald erwartet

http://de.wikipedia.org/wiki/Entwicklungsstadium_(Software)#Beta-Version

Artefakte (Dependencies)

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

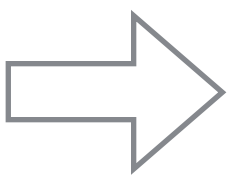
Artifact	Usages	Last Version	Description
mapstruct **(6)** MapStruct Core	1	1.0.0.Beta4	MapStruct Core
mapstruct-jdk8 **(3)** MapStruct Core JDK 8	0	1.0.0.Beta4	MapStruct annotations to be used with JDK 8 and later
mapstruct-processor **(6)** MapStruct Processor	0	1.0.0.Beta4	MapStruct Processor

- *http://sourceforge.net/projects/mapstruct/files/*

Beispiel – JPA-Entity nach DTO

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

```java
@Entity
@Table(name = "KUNDEN")
public class Kunde {

  @Id
  Long id;

  long kundennummer;

  String name;

  @Enumerated
  KundenArt kundenart;

  … Getter und Setter! …
}
```



```java
public class KundeDTO {

  long id;

  String kundennummer;

  String name;

  String kundenart;

  … Getter und Setter! …
}
```

Beispiel – MapStruct minimal

```
import org.mapstruct.Mapper;
import org.mapstruct.factory.Mappers;


@Mapper
public interface KundeMapper {

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


    KundeDTO kunde2KundeDto(Kunde kunde);

}
```

```
KundeDTO dto =
  KundeMapper.INSTANCE.kunde2KundeDto( kunde );
```

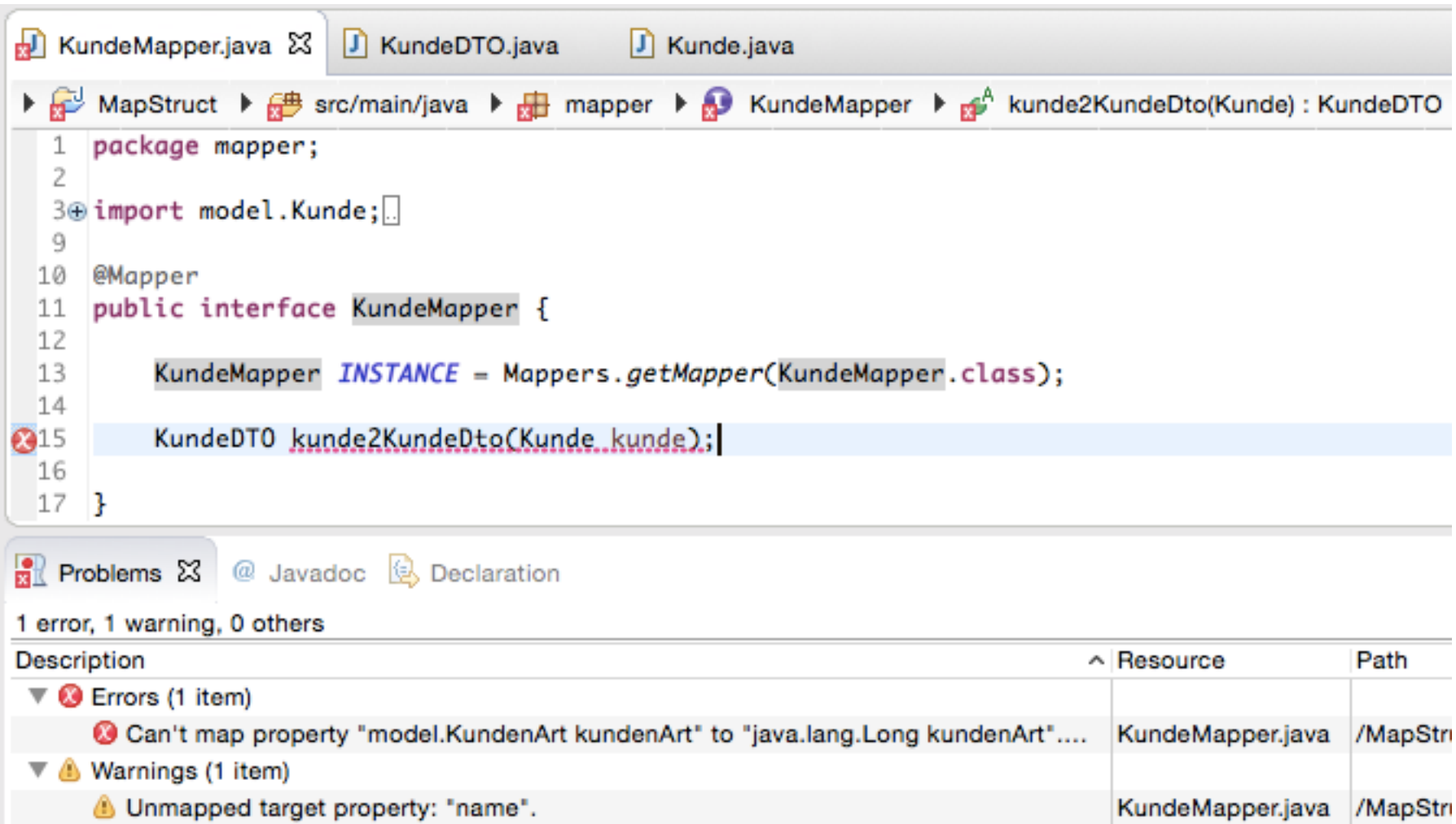
Beispiel – generierter Code

```
@Generated(
    value = "org.mapstruct.ap.MappingProcessor",
    date = "2015-04-21T22:01:21+0200",
    comments = "version: 1.0.0.Beta4, compiler: Eclipse JDT (IDE) …"
)
public class KundeMapperImpl implements KundeMapper {

    @Override
    public KundeDTO kunde2KundeDto(Kunde kunde) {
        if ( kunde == null ) {
            return null;
        }

        KundeDTO kundeDTO = new KundeDTO();

        kundeDTO.setName( kunde.getName() );
        kundeDTO.setKundennummer( String.valueOf( kunde.getKundennummer() ) );
        if ( kunde.getKundenArt() != null ) {
            kundeDTO.setKundenArt( kunde.getKundenArt().toString() );
        }
        if ( kunde.getId() != null ) {
            kundeDTO.setId( kunde.getId() );
        }

        return kundeDTO;
    }
}
```

date- und comments-Attribute können unterdrückt werden

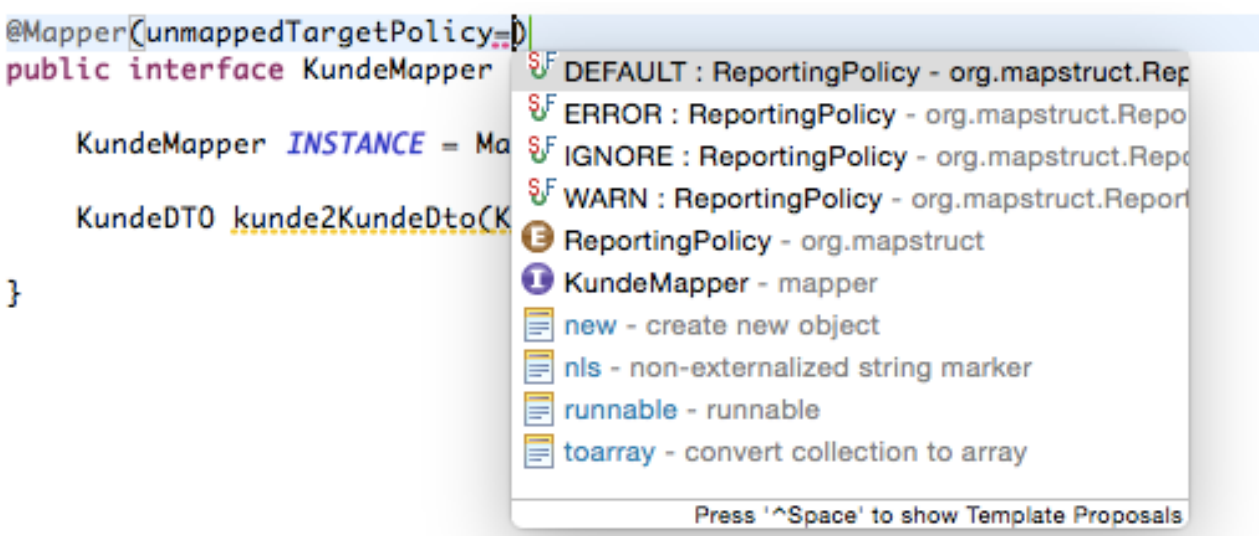
Mapper sind Thread-sicher!

Typsicherheit

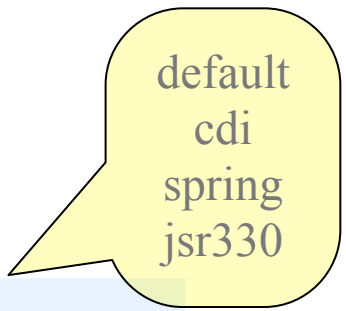


Meldungen für Target-Properties

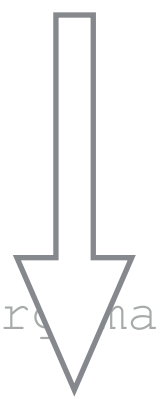
Alternativ global
per System-Property
konfigurieren



Mapper in Komponentenmodellen




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

    KundeDTO kunde2KundeDto(Kunde kunde);

}
```



```
@Generated(
    value = "org.mapstruct.ap.MappingProcessor",
)
@ApplicationScoped
public class KundeMapperImpl implements KundeMapper {
    …
}
```

Updates – Beans ergänzen

```
@Mapper
public interface KundeMapper {

    void updateKundeFromDto(
                                KundeDTO kundeDto,
                                @MappingTarget Kunde kunde);

}
```

Alternativ Target-Typ als Rückgabe (für fluent-Aufrufe)

Max. ein Mapping-Target

Objektgraphen mappen

```
public class Kunde {
   …

   Adresse adresse;

   … Getter und Setter! …
}
```

```
public class KundeDTO {
   …

   AdressDTO adresse;

   … Getter und Setter! …
}
```

```
@Mapper
public interface KundeMapper {

    KundeDTO kunde2KundeDto(Kunde kunde);

    AdresseDTO adresse2AdresseDto(Adresse adresse);

}
```

Unterschiedliche Property-Namen

```
@Mapper
public interface KundeMapper {


    @Mappings({
        @Mapping(source="id", target="entityId"),
        @Mapping(source="name", target="kundenname")
    })
    KundeDTO kunde2KundeDto(Kunde kunde);


    @Mapping(source="plz", target="postleitzahl")
    AdresseDTO adresse2AdresseDto(Adresse adresse);

}
```

Setter-Reihenfolge

```
@Mapper
public interface KundeMapper {

    @Mapping(
        target="kundennummer",
        dependsOn="kundenArt")
    @Mapping(
        target="name",
        dependsOn={"vorname","nachname"})
    KundeDTO kunde2KundeDto(Kunde kunde);

}
```

setKundennummer() wird nach setKundenArt() aufgerufen

Keine zugesicherte Reihenfolge innerhalb des Arrays

Autom. (implizite) Typumwandlung

Primitive Typen ↔ Wrapper (inkl. null-Checks)

int ↔ Integer etc.

int/long/byte ↔ Integer etc.

Primitive Typen & Wrapper ↔ String

int/Integer/Boolean ↔ String etc.

enum ↔ String

BigInteger/BigDecimal ↔ Primitive/Wrapper/String

Autom. Typumwandlung (JAXB)

JAXBElement<T> ↔ T

List<JAXBElement<T>> ↔ List<T>

XMLGregorianCalendar ↔ Date/Calendar

Autom. Typumwandlung (Datum+Zeit)

Date/Calendar/XMLGregorianCalendar ↔ String / dateFormat

Joda ↔ String / dateFormat

Joda ↔ Date, Calendar

java.time ↔ String / dateFormat

java.time ↔ Date, Calendar

Autom. Typumwandlung (Datum+Zeit)

```
@Mapper
public interface KundeMapper {

    @Mapping(source="geburtsdatum", dateFormat="dd.MM.yyyy")
    KundeDTO kunde2KundeDto(Kunde kunde);

    @IterableMapping(dateFormat="dd.MM.yyyy")
    List<String> dateList2StringList(List<Date> list);

}
```

Mehrere Quell-Objekte – Beans mixen

```
@Mapper
public interface KundeMapper {

    @Mappings({
        @Mapping(target="empfaengerName", source="kunde.name"),
        @Mapping(target="postleitzahl", source="adresse.plz"),
        @Mapping(target="gewichtInKg", source="gewicht")
    })
    Lieferanschrift buildLieferanschrift(
                                        Kunde kunde,
                                        Adresse adresse,
                                        Integer gewicht);

}
```

Bel. Verschachtelungstiefe
(null-Checks bei jedem Sprung)

Bel. viele Quell-Parameter

Arrays und Collections

CollectionMappingStrategy (Setter vs. Adder) konfigurierbar

```
@Mapper
public interface KundeMapper {

    KundeDTO kunde2KundeDto(Kunde kunde);

    List<KundeDTO> kunden2Dtos(List<Kunde> kunden);

    KundeDTO[] kunden2Dtos(Kunde[] kunden);


    KundeDTO[] kundenList2DtoArray(List<Kunde> kunden);

    List<KundeDTO> kundenArray2DtoList(Kunde[] kunden);


    Set<String> ints2Strings(Set<Integer> ints);
}
```

Maps mappen

```
@Mapper
public interface KundeMapper {

    KundeDTO kunde2KundeDto(Kunde kunde);

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

Ebenso keyDateFormat

enum-Werte abbilden

```
@Mapper
public interface KundeMapper {

    KundeDTO kunde2KundeDto(Kunde kunde);

    @Mapping(target="EXPRESS", source="PREMIUM")
    @Mapping(target="EXPRESS", source="AKTION")
    @Mapping(target="STANDARD", source="NORMAL")
    DtoKundenArt kundenArt2DtoKundenArt(KundenArt art);

}
```

Eigener Mapping-Code (1)

```
@Mapper
public abstract class KundeMapper {


    @Mappings({
        …
    })
    public abstract KundeDTO kunde2KundeDto(Kunde k);


    public AdresseDTO adresse2AdresseDto(Adresse adr) {
        … eigener Mapping-Code …
    }

}
```

Eigener Mapping-Code (2)

Eigene Mapping-Klasse
muss dasselbe Komponentenmodell wie
der MapStruct-Mapper verwenden!

```
@Mapper(uses=AdresseMapper.class)
public interface KundeMapper {


    @Mappings({
        …
    })
    KundeDTO kunde2KundeDto(Kunde kunde);

}
```

Bean-Fabriken

```
public class DtoFactory {
    public KundeDTO createKundeDTO() {
        return new KundeDTO();
    }
}
```

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

```
@Mapper(uses={DtoFactory.class, EntityFactory.class})
public interface KundeMapper {

    KundeDTO kunde2KundeDto(Kunde kunde);
    Kunde kundeDto2Kunde(KundeDTO dto);
}
```

Exceptions beim Mapping

```
public class SelbstgeschriebenerMapper {
    public KundenArt long2KundenArt(Long zahl)
        throws KundenArtException, SchwereException {
        …
    }
}
```

Nicht deklarierte Checked-Exceptions werden als RuntimeExceptions weiter geworfen

```
@Mapper(uses=SelbstgeschriebenerMapper.class)
public interface KundeMapper {

    Kunde dto2Kunde(KundeDTO dto) throws KundenArtException;

}
```

Konstante Werte im Ziel setzen

```
@Mapper
public interface KundeMapper {

    @Mapping(
      target="stringAttribut", constant="mein String")
    @Mapping(
      target="intAttribut", constant="23")
    @Mapping(
      target="longWrapperAttribut", constant="12345678")
    @Mapping(
      target = "dateAttribut",
      dateFormat = "dd.MM.yyyy",
      constant = "24.04.2015"
    )
    KundeDTO kunde2KundeDto(Kunde kunde);

}
```

Berechnete Werte im Ziel setzen

```
@Mapper(imports=Date.class)
public interface KundeMapper {

    @Mapping(
      target="geburtsdatum",
      expression=
        "java( new Date(kunde.getGeburtsdatumMillis()) )"
    )
    KundeDTO kunde2KundeDto(Kunde kunde);

}
```

Methoden per Qualifier auswählen (1)

```
public class AnredeMapper {

    public String deutsch2Englisch(String anrede) { … }
    public String englisch2deutsch(String anrede) { … }
}
```

```
@Mapper(uses=AnredeMapper.class)
public interface KundeMapper {

    KundeDTO kunde2KundeDto(Kunde kunde);
}
```

Methoden per Qualifier auswählen (2)

```
@AnredeUebersetzer
public class AnredeMapper {

    @DeutschNachEnglisch
    public String deutsch2Englisch(String anrede) { … }

    @EnglischNachDeutsch
    public String englisch2deutsch(String anrede) { … }
}
```

Selbstgeschriebene @Qualifier-Annotationen

```
@Mapper
public interface KundeMapper {
    @Mapping(target="anrede",
        qualifiedBy={
          AnredeUebersetzer.class,EnglischNachDeutsch.class
        })
    KundeDTO kunde2KundeDto(Kunde kunde);
}
```

null-Mapping anpassen

- Bei null-Wert wird standardmäßig null gesetzt.

- Mit

```
nullValueMappingStrategy =
    NullValueMappingStrategy.RETURN_DEFAULT
```

  werden stattdessen Default-Werte gesetzt
  (für primitive Type, Listen etc.)

- Verfügbar bei @Mapper etc.

Mapping-Konfiguration "vererben"

```
@Mapper
public interface KundeMapper {

    @Mappings(
        …
    )
    KundeDTO kunde2KundeDto(Kunde kunde);


    @InheritConfiguration(name="kunde2KundeDto")
    @Mappings({ … })
    void updateDtoFromKunde(
              Kunde kunde,
              @MappingTarget KundeDTO dto);

}
```

Umgekehrte Mappings

```
@Mapper
public interface KundeMapper {

    @Mappings(
        …
    )
    KundeDTO kunde2KundeDto(Kunde kunde);


    @InheritInverseConfiguration
    Kunde kundeDto2Kunde(KundeDTO dto);

}
```

Gemeinsam genutzte Konfigurationen

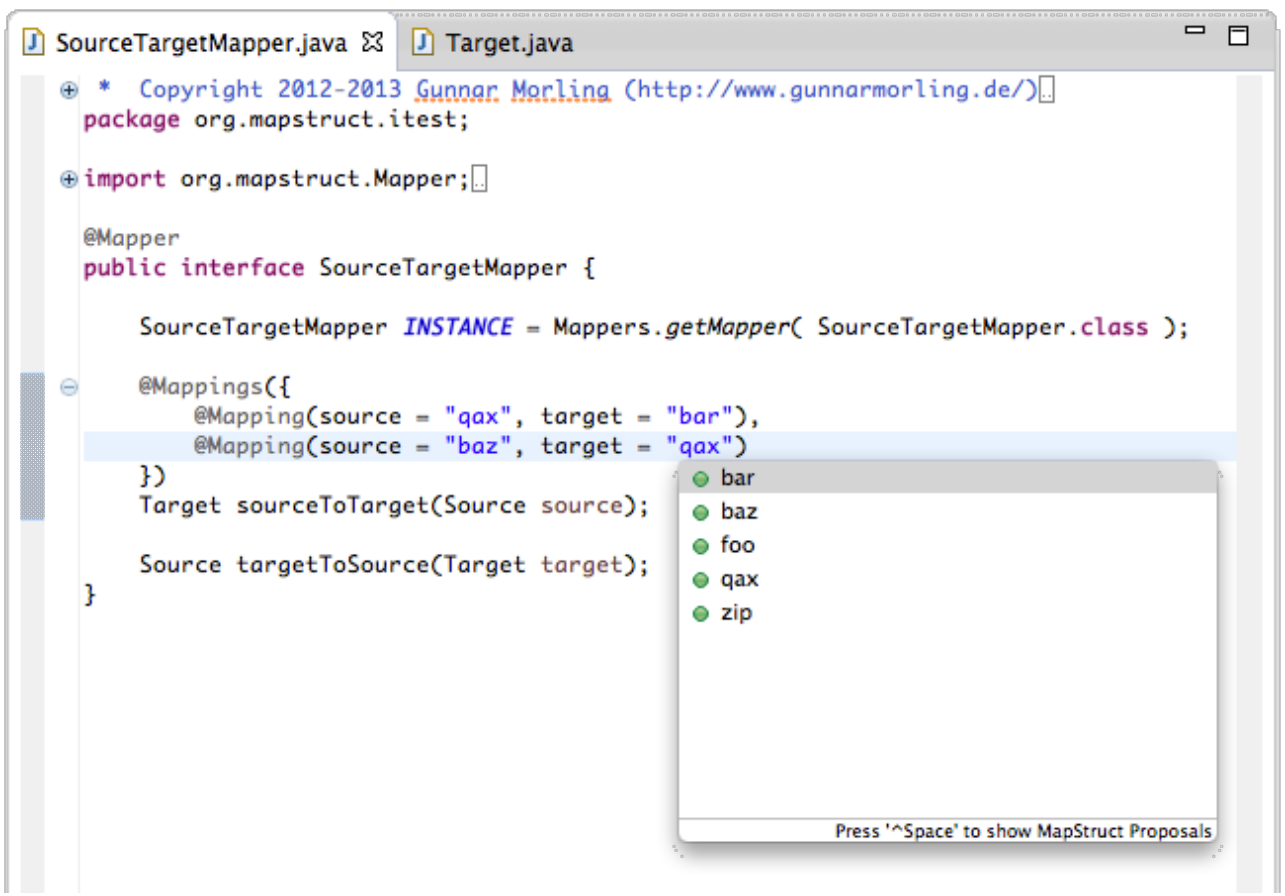
```
@MapperConfig(
    mappingInheritanceStrategy=
      MappingInheritanceStrategy.AUTO_INHERIT_FROM_CONFIG
)
public class MeineKonfiguration {

   @Mappings({ … })
   BasisDTO entity2Dto(BasisEntity entity);

}
```

Hier wird noch kein Mapping-Code generiert!

```
@Mapper(config=MeineKonfiguration.class)
public interface KundeMapper {
   @Mappings({ … })
   KundeDTO kunde2KundeDto(Kunde kunde);
}
```

Ausblick – Eclipse-Plugin

- *https://github.com/mapstruct/mapstruct-eclipse/*
- Frühes Entwicklungsstadium



MapStruct – Fazit

- Trotz Beta schon produktiv einsetzbar!
  - Zur Not wird der generierte Code eingecheckt und später von Hand weiter gepflegt.
- Generierter Code ist einfach und *lesbar.*
- Flexible (Custom-)Mappings
- Einfache Einbindung in div. Komponentenmodelle

Fragen?

Vielen Dank! 😀

thomas@muchsoft.com

www.muchsoft.com

www.javabarista.de