Bean-Mapping mit MapStruct

Thomas Much
@thmuch

www.muchsoft.com

Java und Beans, eine lange Geschichte…

1995

1997

JavaBeans

1998

Form Beans

1999

DTOs

2000

POJOs

2005

JAXB-Entities

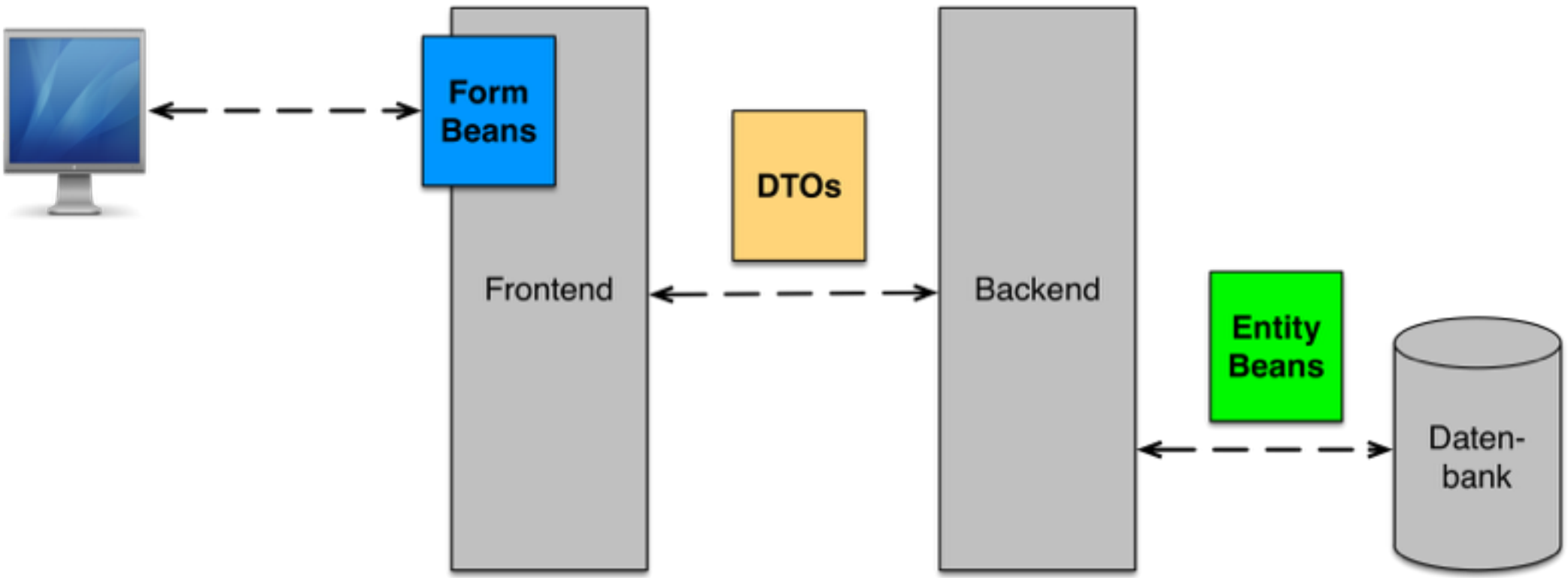
2006

JPA-Entities

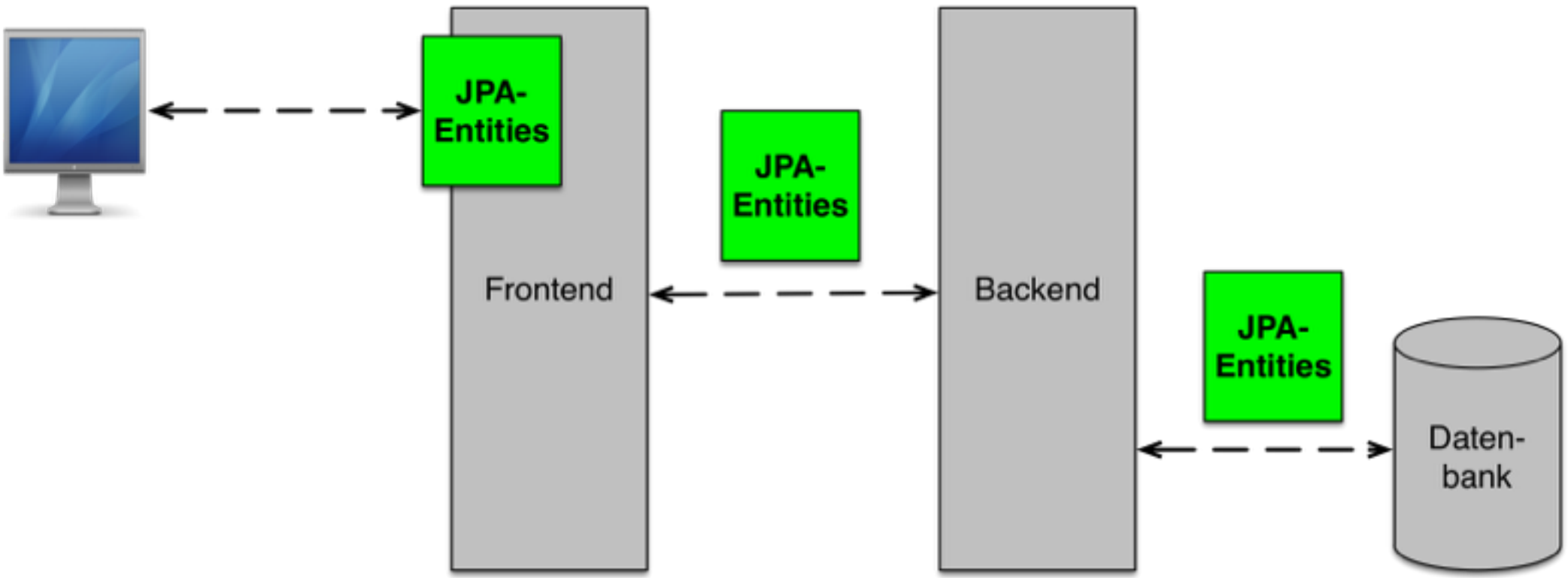
2015

Typsicheres, schnelles Bean-Mapping!

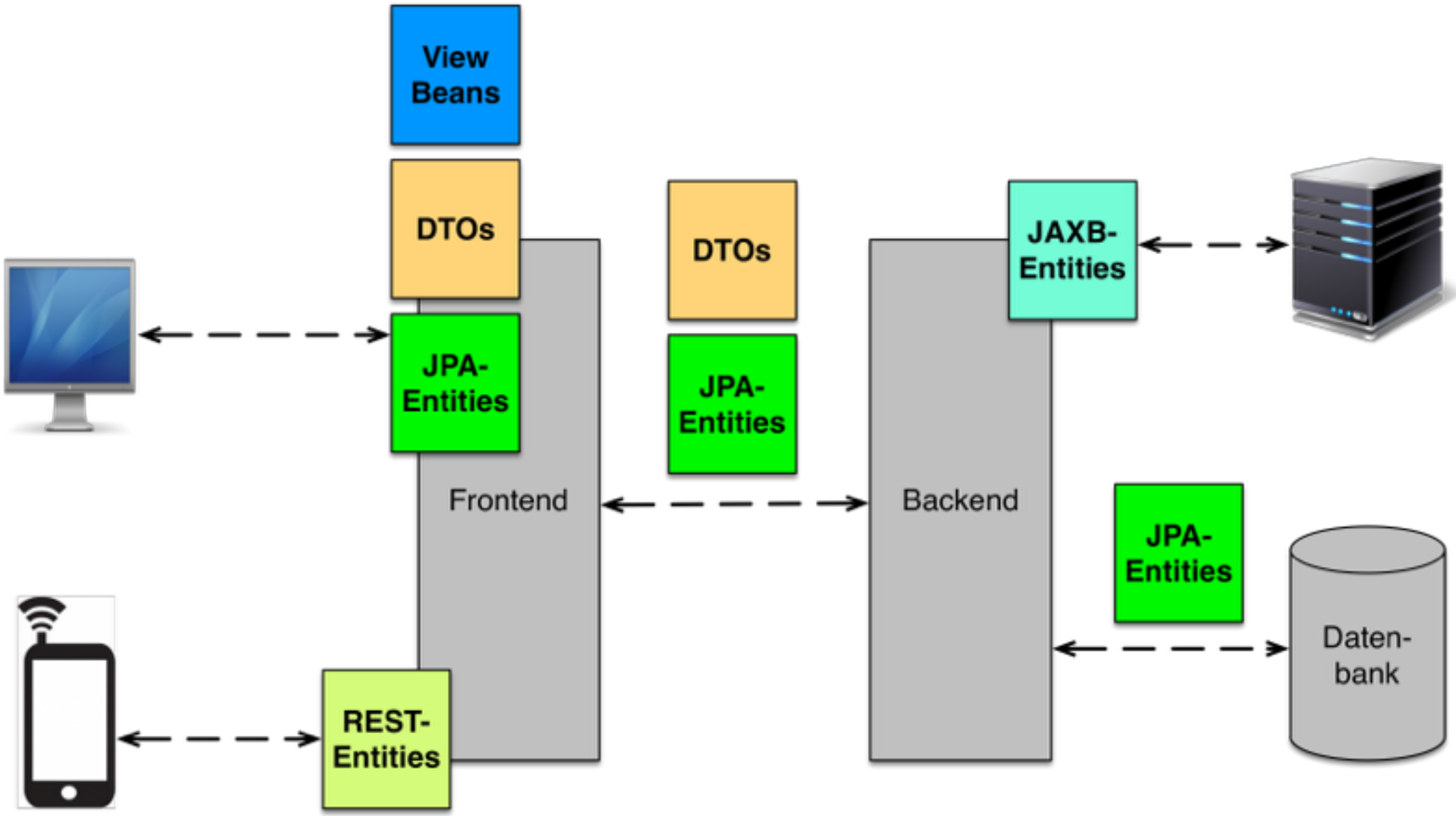
Alte Architekturen (J2EE)



Was Java EE 5+ und JPA versprechen



Die moderne Realität

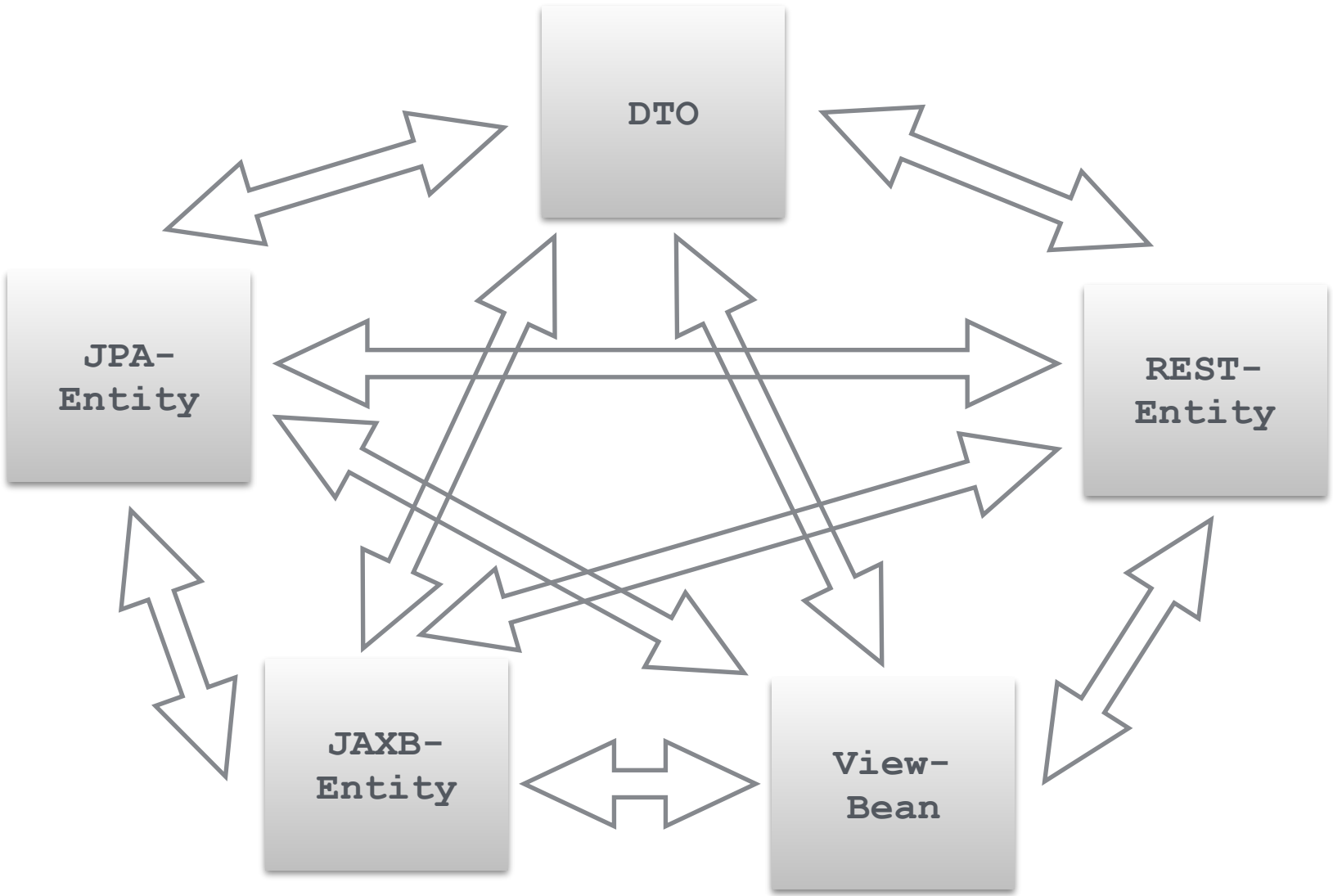


Das erforderliche Mapping …

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

  @Id
  Long id;

  long kundennummer;

  String name;

  @Enumerated
  KundenArt kundenart;

  … Getter und Setter! …
}
```

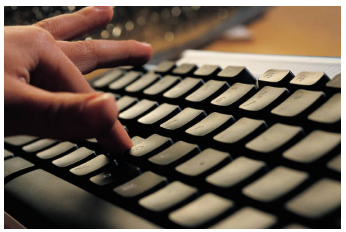
```
public class KundeDTO {


  long id;

  String kundennummer;

  String name;


  String kundenart;

  … Getter und Setter! …
}
```

… kann aufwendig werden



Mapping-Implementation – wie?

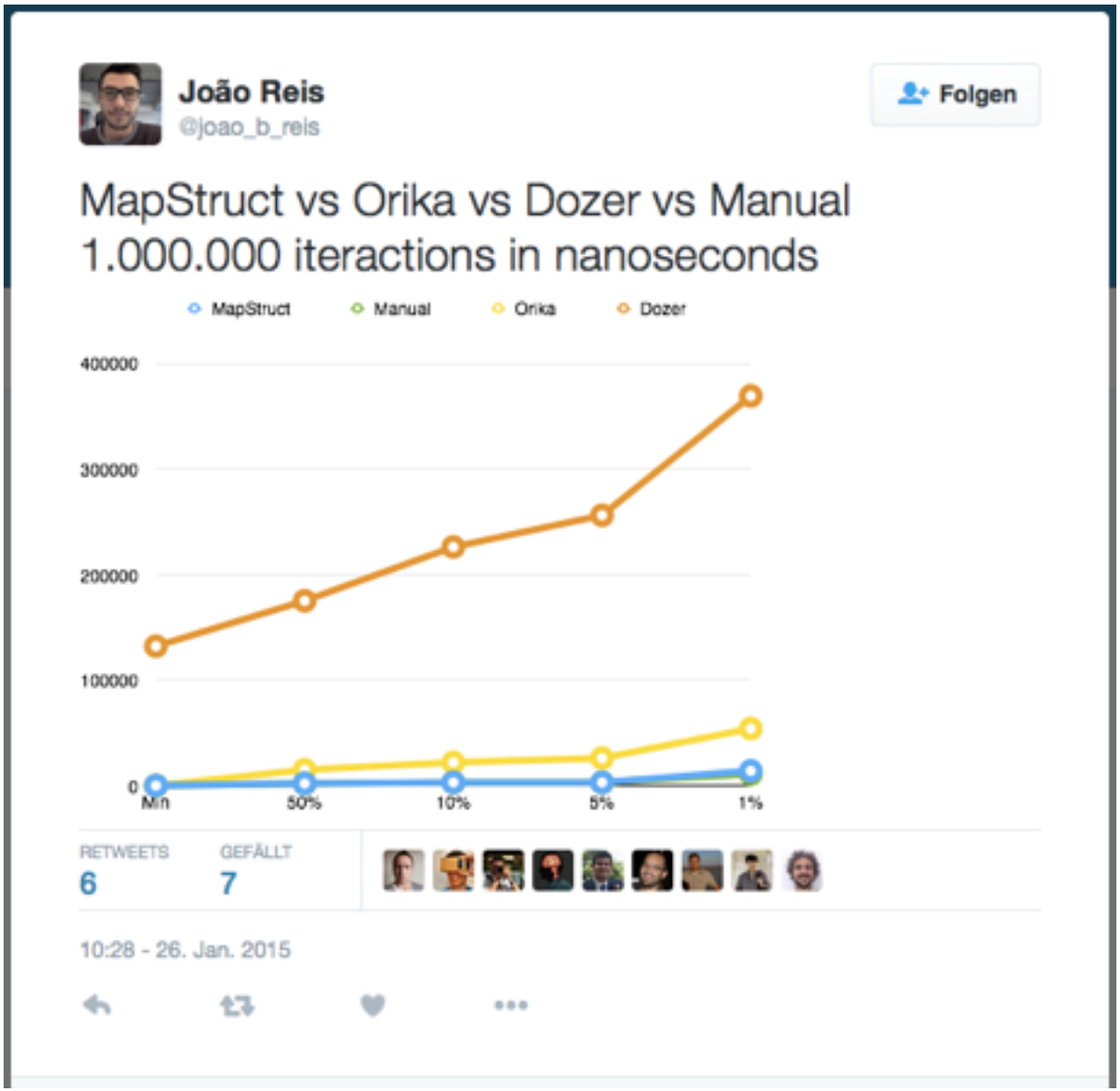
- Selbst programmiert
  - alle Getter/Setter von Hand aufrufen
  - eigene Reflection-Routinen



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

- Probleme: Aufwand, fehlende Typsicherheit, Performance …

Performance… Performance!



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

MapStruct

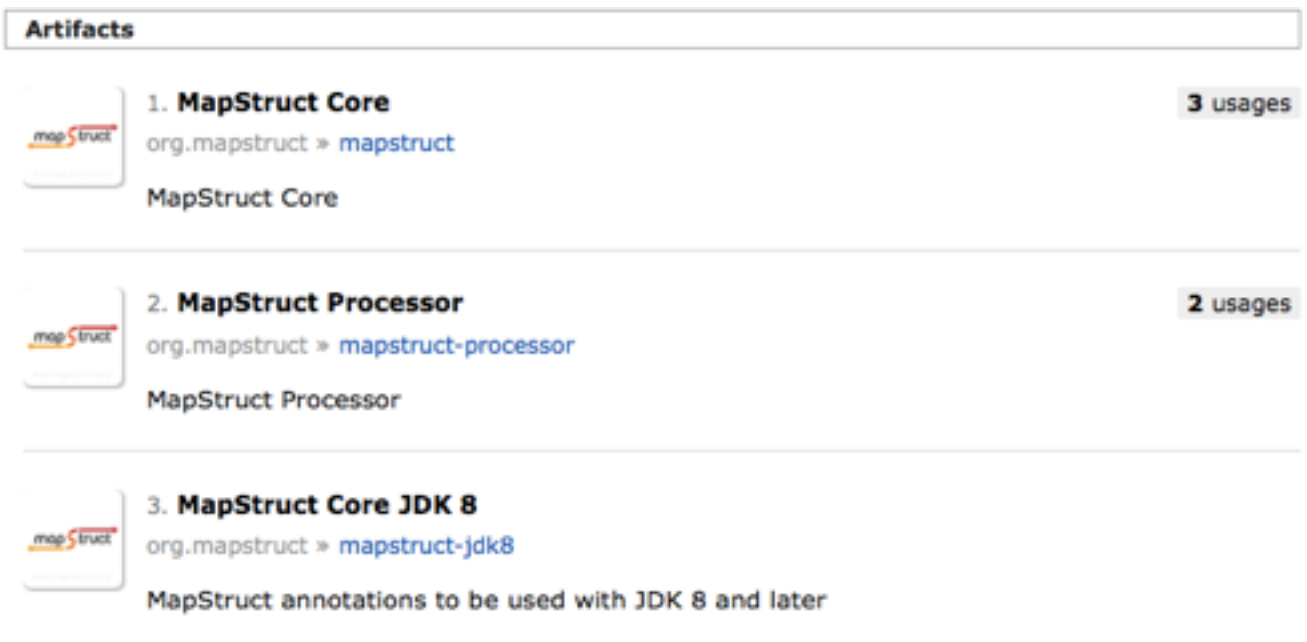
- **MapStruct: *http://mapstruct.org/***

- 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

Artefakte (Dependencies)

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

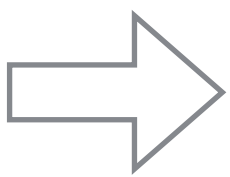


- *http://sourceforge.net/projects/mapstruct/files/*

Beispiel – JPA-Entity nach DTO

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

```
@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! …
}
```

Beispiel – Wünsch Dir was!

```
import org.mapstruct.Mapper;


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

    KundeDTO kunde2KundeDto(Kunde kunde);

}
```

Kann global per Property festgelegt werden. Auch z.B. für Spring.

```
@Inject
private KundeMapper mapper;

…

KundeDTO dto = mapper.kunde2KundeDto( kunde );
```

Beispiel – Implementierung?

- Wie erzeugen wir aus dem Interface die Mapping-Implementierung?

- *Speichern* z.B. in Eclipse.
- Genauer: Compiler-Aufruf (geht also auch bei javac)

- Annotations-Prozessor erledigt den Rest!

Beispiel – Live-Demo!

- Wie erzeugen wir aus dem Interface die Mapping-Implementierung?

- *Speichern* z.B. in Eclipse.
- Genauer: Compiler-Aufruf (geht also auch bei jav

- Annotations-Prozessor erledigt den Rest!

Beispiel – generierter Code

```
@Generated(
    value = "org.mapstruct.ap.MappingProcessor",
    date = "2016-03-04T21:56:55+0100",
    comments = "version: 1.0.0.Final, compiler: Eclipse JDT (IDE) …"
)
@ApplicationScoped
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!

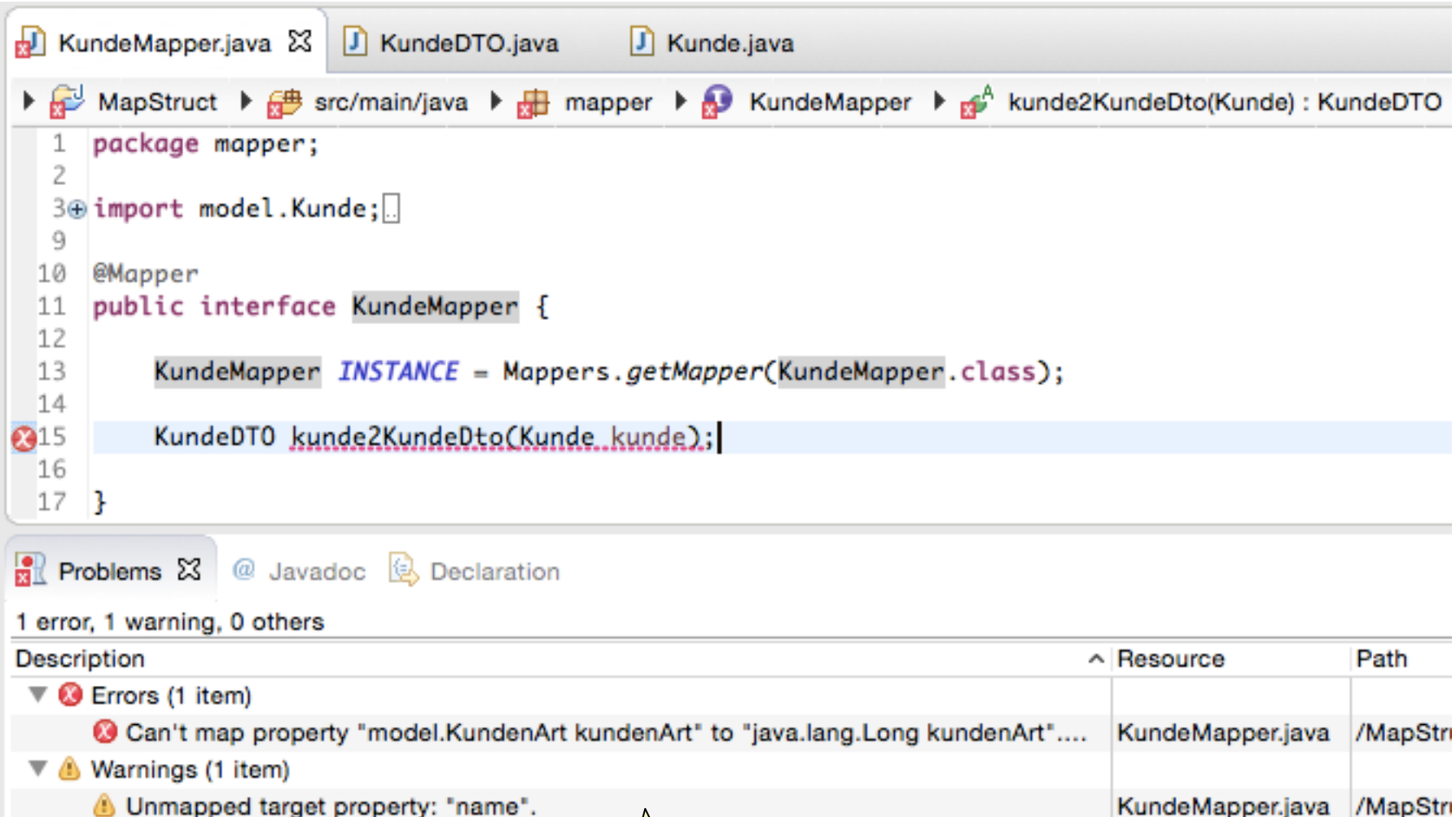
Kein Injection-Container? Kein Problem!

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


@Mapper(componentModel = "default")
public interface KundeMapper {

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

    KundeDTO kunde2KundeDto(Kunde kunde);
}
```

Dann wird ein kleines Runtime-JAR (ca. 16 K) benötigt.

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

Typsicherheit



WARN/ERROR/IGNORE konfigurierbar (pro Mapper bzw. global)

Properties umbenennen, ignorieren, formatieren

```
@Mapper
public interface KundeMapper {

    @Mappings({
        @Mapping(target = "kundennummer", source = "knr"),
        @Mapping(target = "name", ignore = true),
        @Mapping(target = "datum", dateFormat = "dd.MM.yyyy")
    })
    KundeDTO kunde2KundeDto(Kunde kunde);

}
```

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

Updates und Reverse Mappings

```
@Mapper
public interface KundeMapper {

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

    void updateKundeDto(
                        Kunde kunde,
                        @MappingTarget KundeDTO dto);

    Kunde kundeDto2Kunde(KundeDTO dto);

}
```

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

Max. ein Mapping-Target

Mappings wiederverwenden

```
@Mapper
public interface KundeMapper {

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

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


    @InheritInverseConfiguration(name = "kunde2KundeDto")
    @Mappings({ … })
    Kunde kundeDto2Kunde(KundeDTO dto);

}
```

Mappings (Konfigurationen) zentral definieren

```
@MapperConfig(
    mappingInheritanceStrategy=
      MappingInheritanceStrategy.AUTO_INHERIT_FROM_CONFIG
)
public interface 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);
}
```

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);

}
```

Eigener Mapping-Code (1)

```
@Mapper
public abstract class KundeMapper {

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

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

}
```

Alternativ mit Interface
und Java-8-Default-Methoden.

Eigener Mapping-Code (2)

Eigene Mapping-Klasse
muss dasselbe Komponentenmodell wie
der MapStruct-Mapper verwenden
oder einen Default-Konstruktor
anbieten.

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


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

}
```

Exceptions beim Mapping

```
public class MeinAdresseMapper {
    public AdresseDTO adresseToAdresseDto(Adresse adresse)
        throws AdresseException, SchwereException {
        …
    }
}
```

Nicht deklarierte Checked-Exceptions werden als RuntimeExceptions weiter geworfen

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

    KundeDTO kunde2KundeDto(Kunde k) throws AdresseException;

}
```

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);
}
```

Mehrere Quell-Objekte – Beans mixen

```
@Mapper
public interface KundeMapper {

    @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

null-Mapping anpassen

- Bei null-Quell-Wert wird im Ziel standardmäßig null gesetzt.

- Mit

```
nullValueMappingStrategy =
    NullValueMappingStrategy.RETURN_DEFAULT
```

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

- Verfügbar bei @Mapper etc.

Default-Werte und Konstanten

```
@Mapper
public interface KundeMapper {

    @Mapping(target="name", default="undefiniert")
    @Mapping(target="kundennummer", default="-1")

    @Mapping(
      target="beschreibung", constant="mein String")

    KundeDTO kunde2KundeDto(Kunde kunde);

}
```

Auch für Zahlen, formatierte Datumswerte und selbstgeschriebene String2xxx-Mapper

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

Formatierung (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);

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

}
```

Ebenso keyDateFormat

Arrays und Collections

```
@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);
}
```

Enum-Werte abbilden

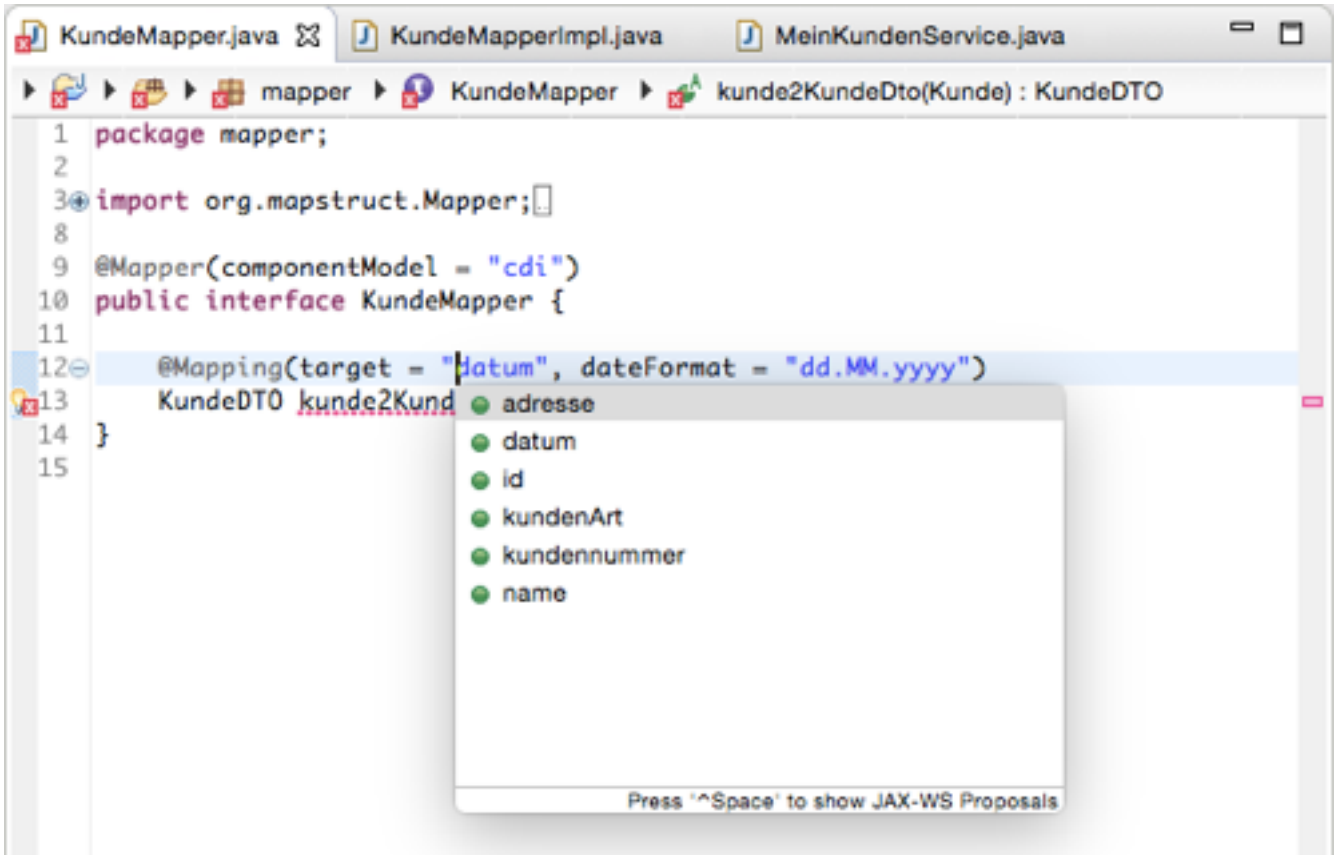
```
@Mapper
public interface KundeMapper {

    KundeDTO kunde2KundeDto(Kunde kunde);

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

}
```

Viele weitere Möglichkeiten

- Anpassbar:
  - Namen der generierten Packages und Impl-Klassen
  - Namen der verwendeten Zugriffsmethoden (Getter/Setter/Adder etc.)
- Java-Expressions zur Target-Wert-Berechnung.
- Mehrdeutige Target-Bean-Typen per Qualifier auswählbar.
- Hooks für den Mapping-Lebenszyklus
  - Decorators für Mapper (CDI, Spring)
  - @BeforeMapping, @AfterMapping

Eclipse-Plugin

- *https://github.com/mapstruct/mapstruct-eclipse/*
- Derzeit Code-Completion und einige Quick-Fixes.
- Refactoring-Support ist geplant.



MapStruct – Fazit

- Hervorragend produktiv einsetzbar!
  - Und wer Code-Generatoren nicht traut, kann zur Not den generierten Mapping-Code einchecken und von Hand weiterentwickeln.

- Generierter Code ist performant, einfach und *lesbar.*
- Flexible (Custom-)Mappings.
- Einfache Einbindung in div. Komponentenmodelle.
- Gute Dokumentation.

Neugierig geworden?

http://mapstruct.org/

😀

Danke!

MapStruct Home News Documentation Download Getting help Development G+1 2

Fork me on GitHub

Development

This page gives your all the information you need to get started with hacking on MapStruct. MapStruct is a very young open source project and we're looking forward to any contribution!

1. Team

MapStruct is brought to you by a helpful international community of open source enthusiasts. The following persons are contributing to the project:

Gunnar Morling is the original author of MapStruct and leads the project.

He is a long-time Java developer and open-source committer. He is part of the Hibernate team at Red Hat, where he works on Hibernate OGM, Validator and Search. 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, maintaining the Surefire plugin.

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) were he believes MapStruct can be a true asset, reducing the amount of repetitive, error-prone work.

Contents

Fragen?

Vielen Dank! 😀

thomas@muchsoft.com

www.javabarista.de

@thmuch