



Mapping Frameworks in Action

Schwaig b. Nürnberg, 17.10.14



Johannes Schwalb

Senior Developer
Senacor Technologies AG

Wieseneckstr. 26
90571 Schwaig bei Nbg.

Telefon: +49 (911) 4244 - 188
Telefax: +49 (911) 4244 - 100
Mobil: +49 (172) 89 35 005
johannes.schwalb@senacor.com
www.senacor.com



Pius Hübl

Developer
Senacor Technologies AG

Wieseneckstr. 26
90571 Schwaig bei Nbg.

Telefon: +49 (911) 4244 - 302
Telefax: +49 (911) 4244 - 100
Mobil: +49 (151) 67 887 927
pius.huebl@senacor.com
www.senacor.com

1	Motivation – Wozu Mapping?
2	Anwendungsfallbeispiel
3	Handwritten Mapping
4	Auswahl der Mapping-Frameworks
5	Framework 1: Dozer
6	Framework 2: ModelMapper
7	Framework 3: Orika
8	Vergleich Performance
9	Bewertung der ausgewählten Frameworks

„Mit JEE 6 und JPA 2 brauchst du keine DTOs und kein Mapping mehr. Du gibst die Entities direkt raus.“

Adam Bien, Training JEE Einführung, 15.04.2011, Schwaig b. Nürnberg

„Mit JEE 6 und JPA 2 brauchst du keine DTOs und kein Mapping mehr. Du gibst die Entities direkt raus.“

Adam Bien, Training JEE Einführung, 15.04.2011, Schwaig b. Nürnberg

Wirklich?

Sinnvolle Anwendungsfälle für Mapping

... aber was ist mit ...

- ... stabilen Schnittstellen
- ... Annotationen, die der Konsument nicht kennen muss/soll
- ... „geheimen“ Attributen der Entities wie Scoring
- ... „Contract-First“ Modellierung
- ...



Auch wenn es lästig ist...

... um ein Mapping kommt man manchmal nicht herum.

Glücklicherweise gibt es aber etliche Java-Frameworks, die uns diese lästige Standardaufgabe abnehmen (können):

OTOM Morph Smooks Modelbridge
Moo EZ-Morph Dozer
Apache Commons JMapper
BeanUtils Orika
OMapper Transmorph
ModelMapper
Nomin Apache Commons
Convert

Auch wenn es lästig ist...

... um ein Mapping kommt man manchmal nicht herum.

Glücklicherweise gibt es aber etliche Java-Frameworks, die uns diese lästige Standardaufgabe abnehmen (können):

OTOM Morph Smooks Modelbridge
Moo EZ-Morph Dozer
Apache Commons BeanUtils OMapper Orika JMapper
ModelMapper Transmorph
Nomin Apache Commons Convert

1 Motivation – Wozu Mapping?

2 **Anwendungsfallbeispiel**

3 Handwritten Mapping

4 Auswahl der Mapping-Frameworks

5 Framework 1: Dozer

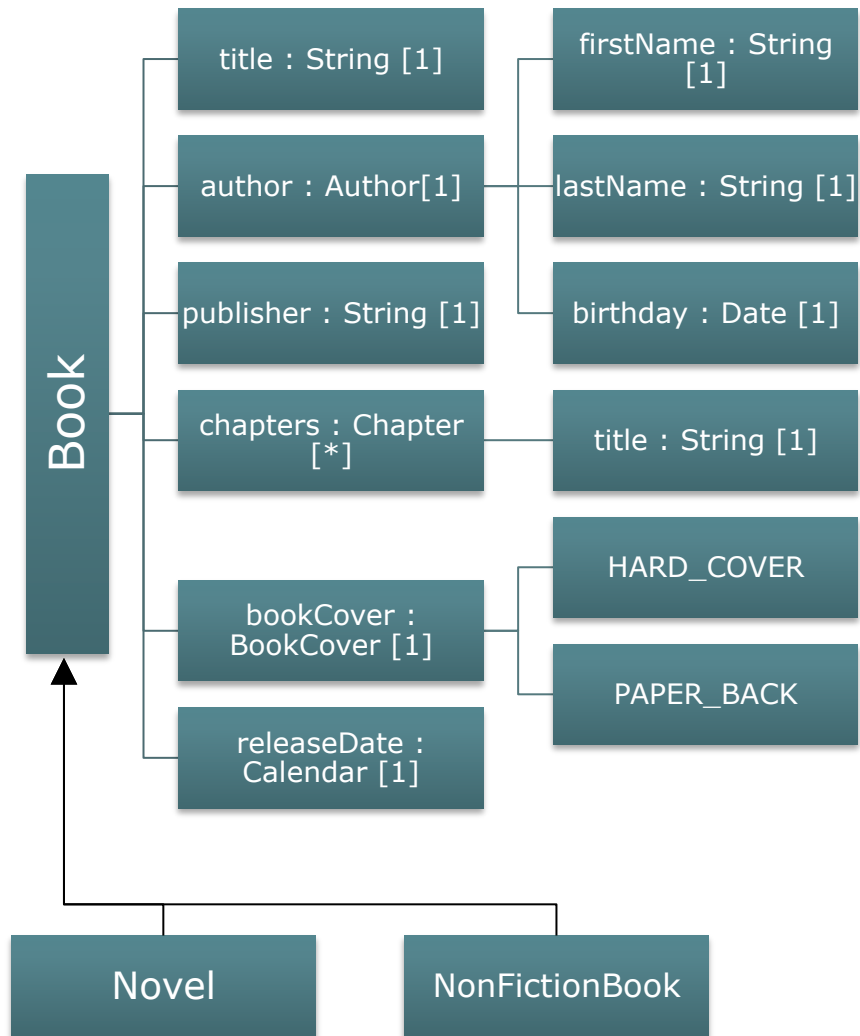
6 Framework 2: ModelMapper

7 Framework 3: Orika

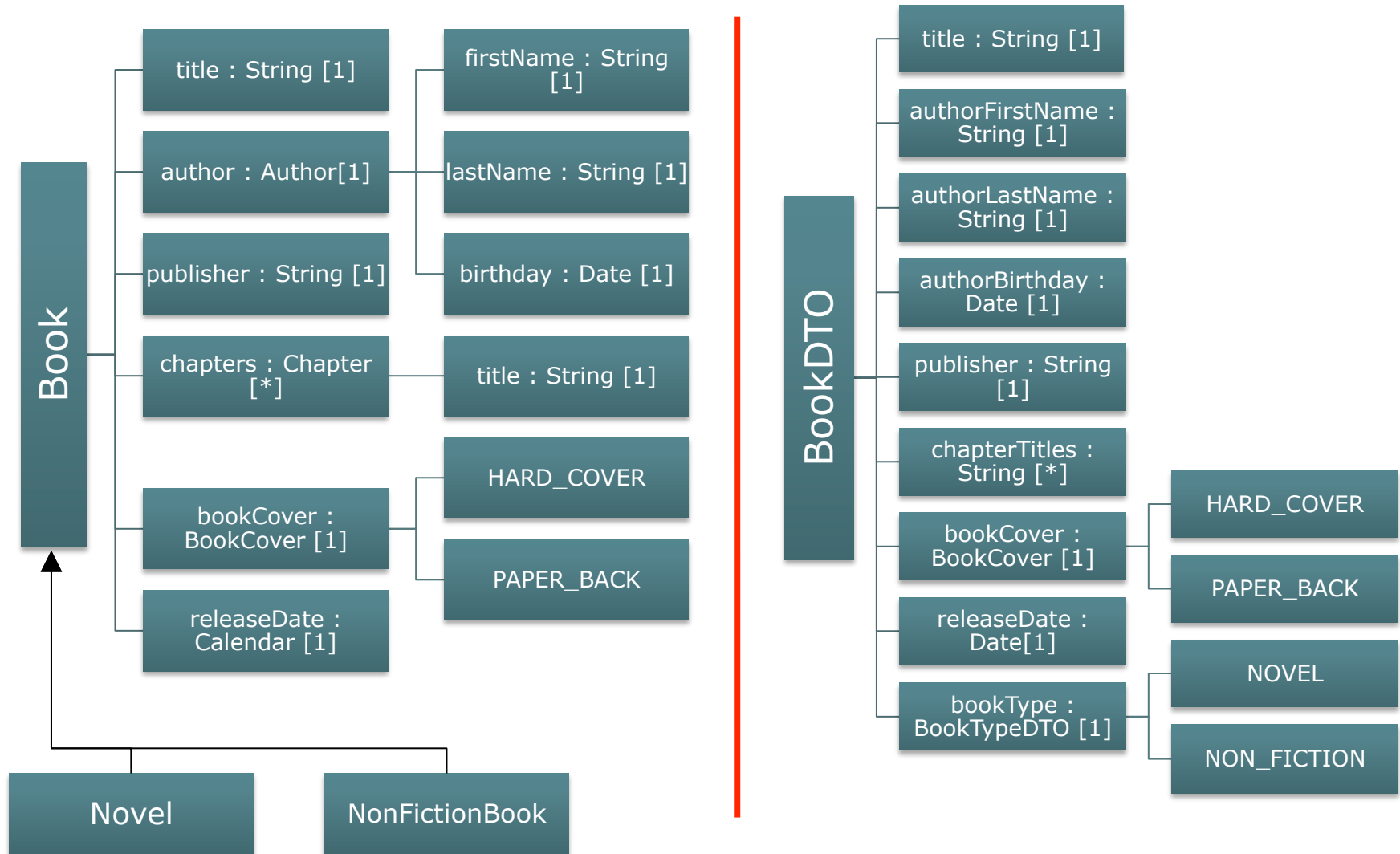
8 Vergleich Performance

9 Bewertung der ausgewählten Frameworks

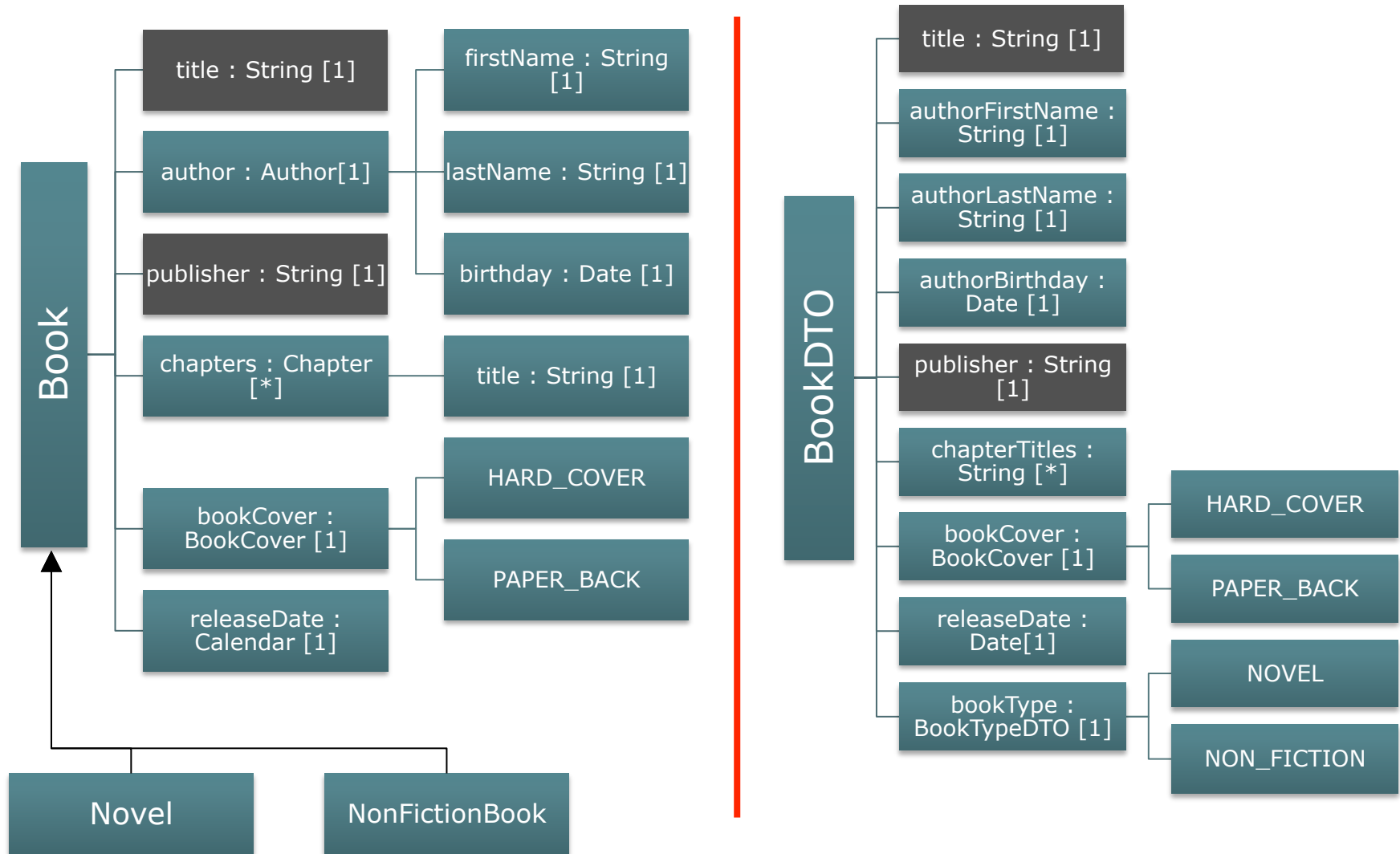
Beispiel: Mapping Book



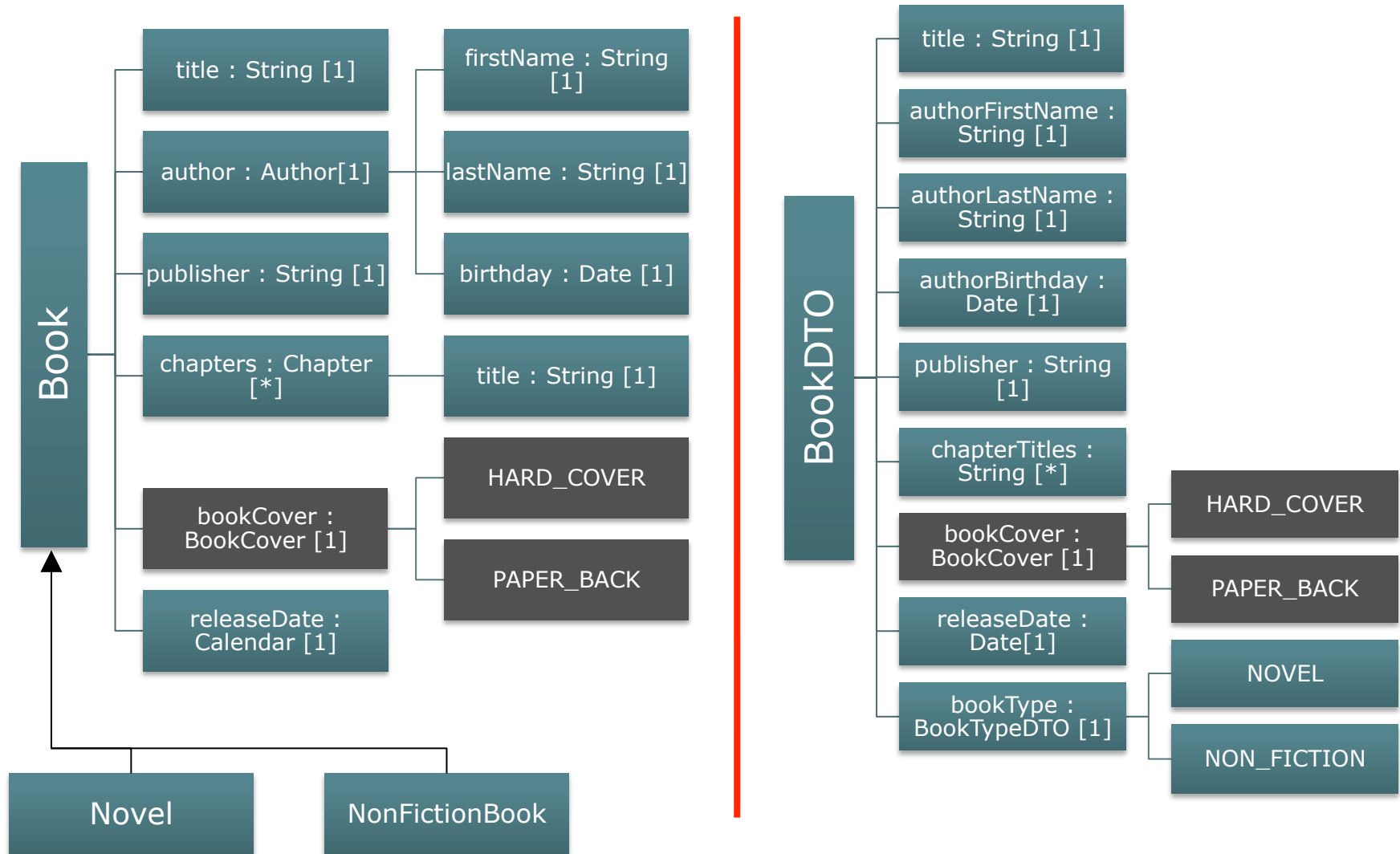
Beispiel: Mapping Book <-> BookDTO



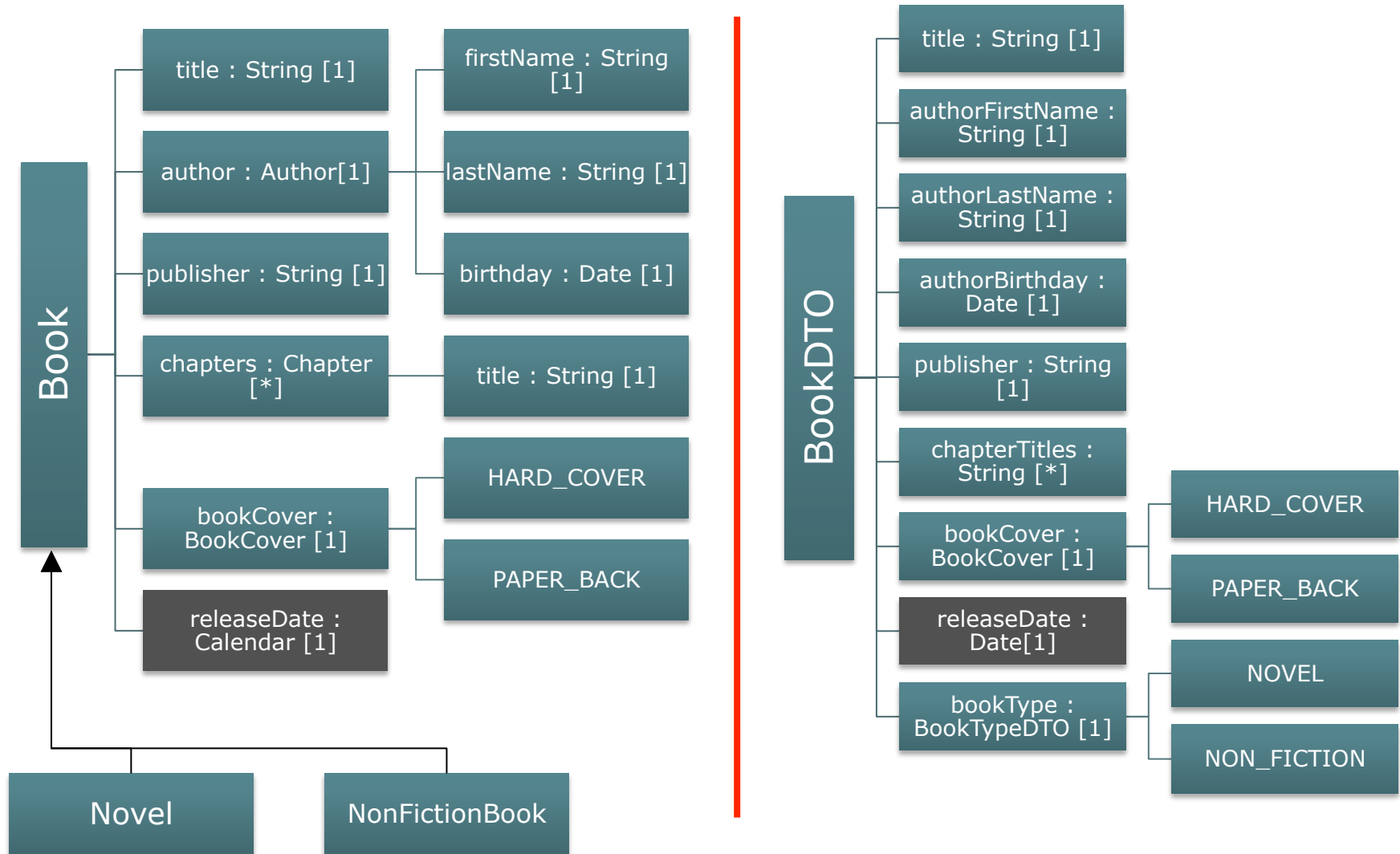
Gruppe 1 (Simple Mapping)



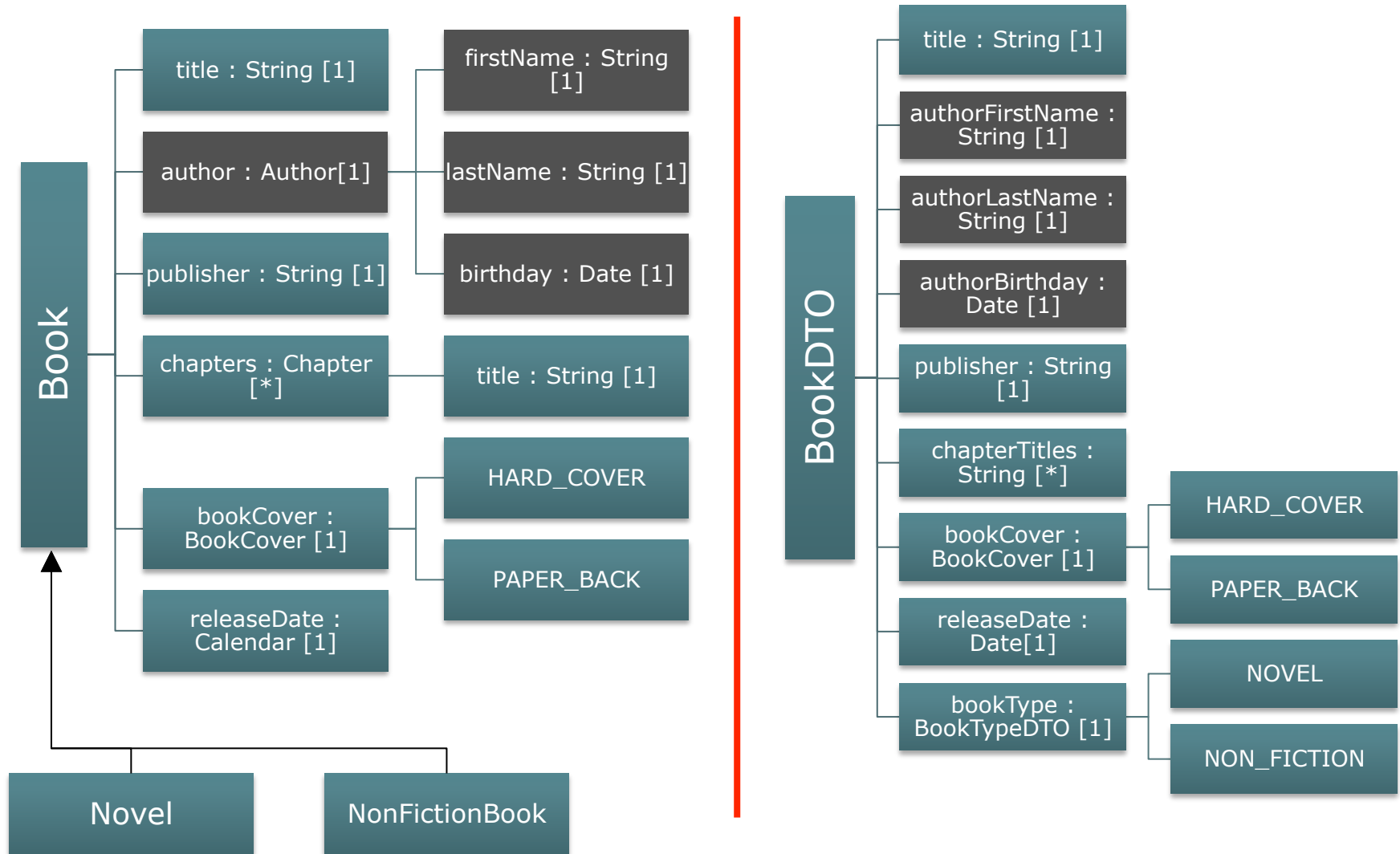
Gruppe 2 (Enum Mapping)



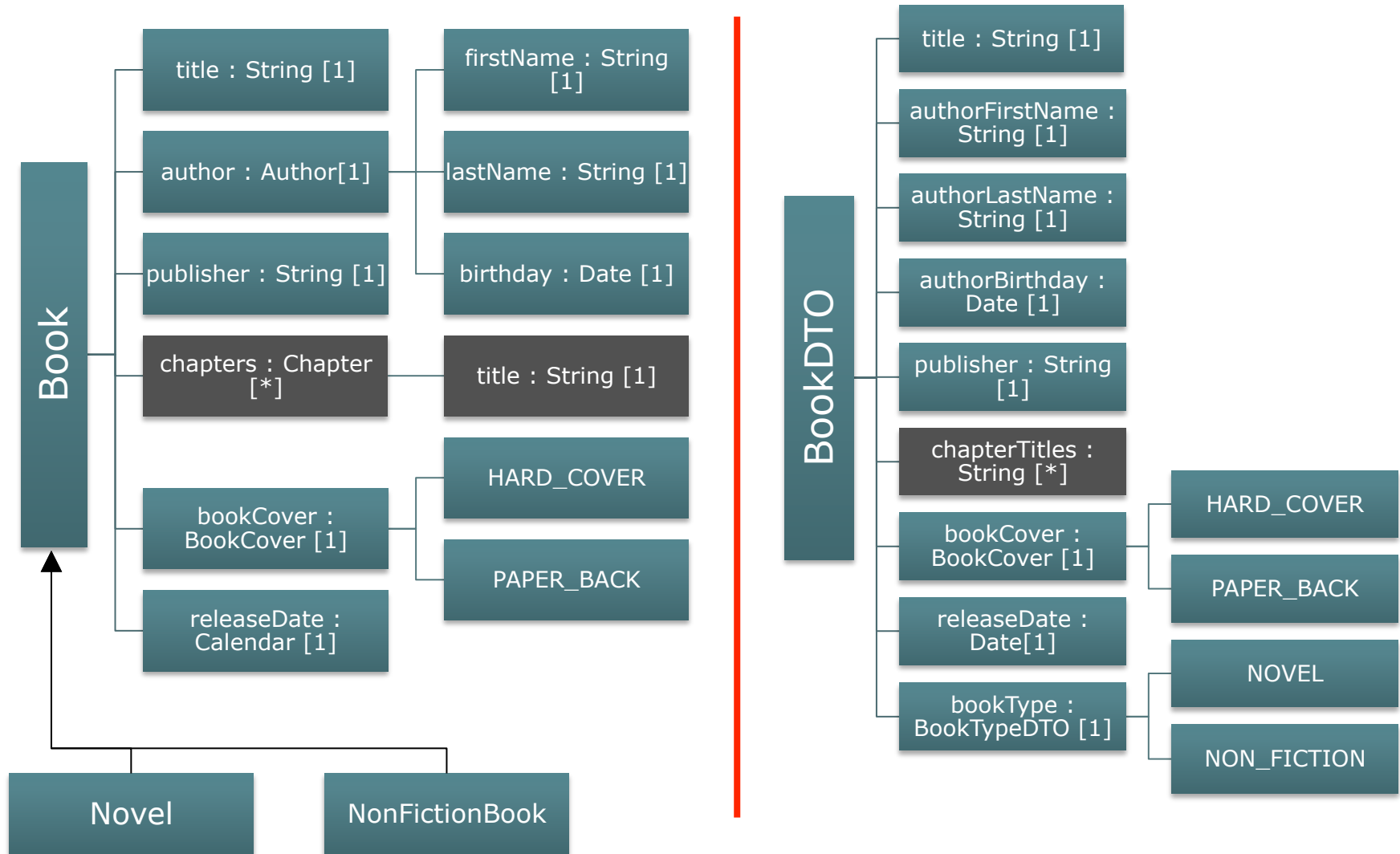
Gruppe 3 (Simple Type Conversion z.B.: Date <-> Calendar)



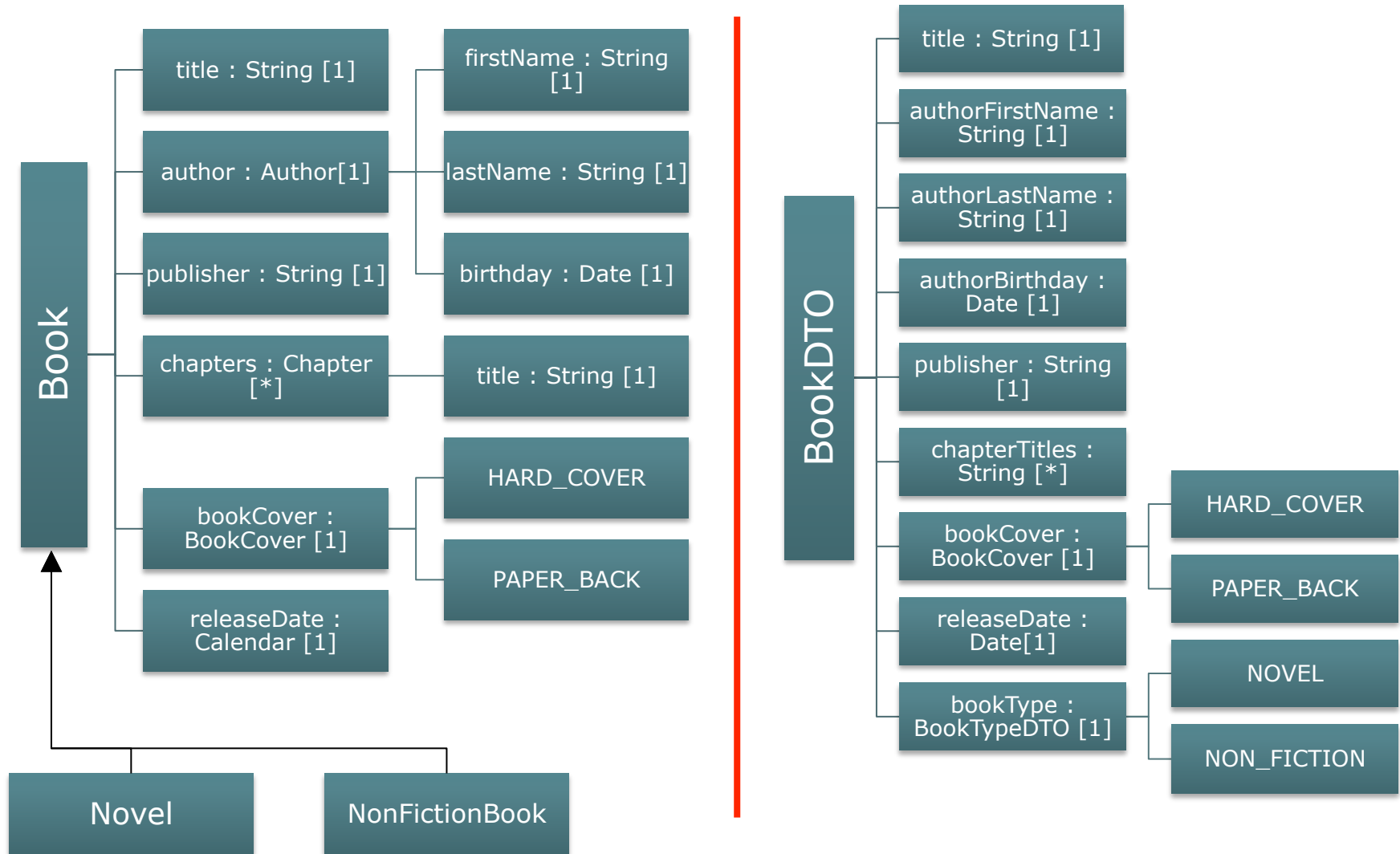
Gruppe 4 (Deep Object Mapping)



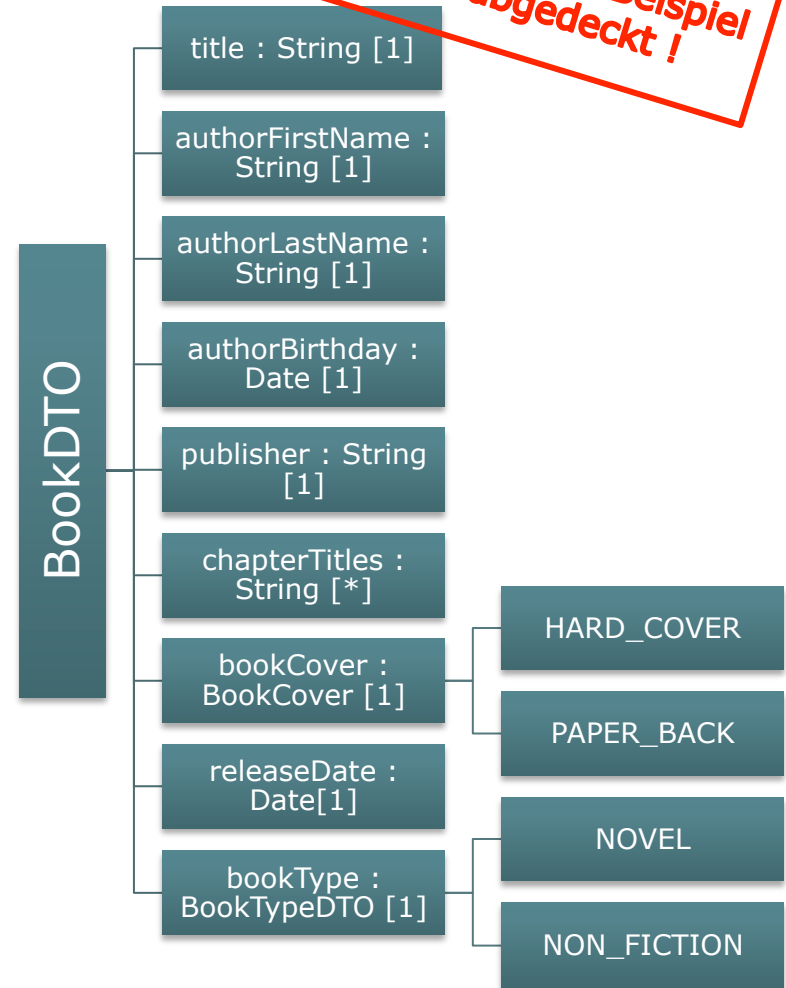
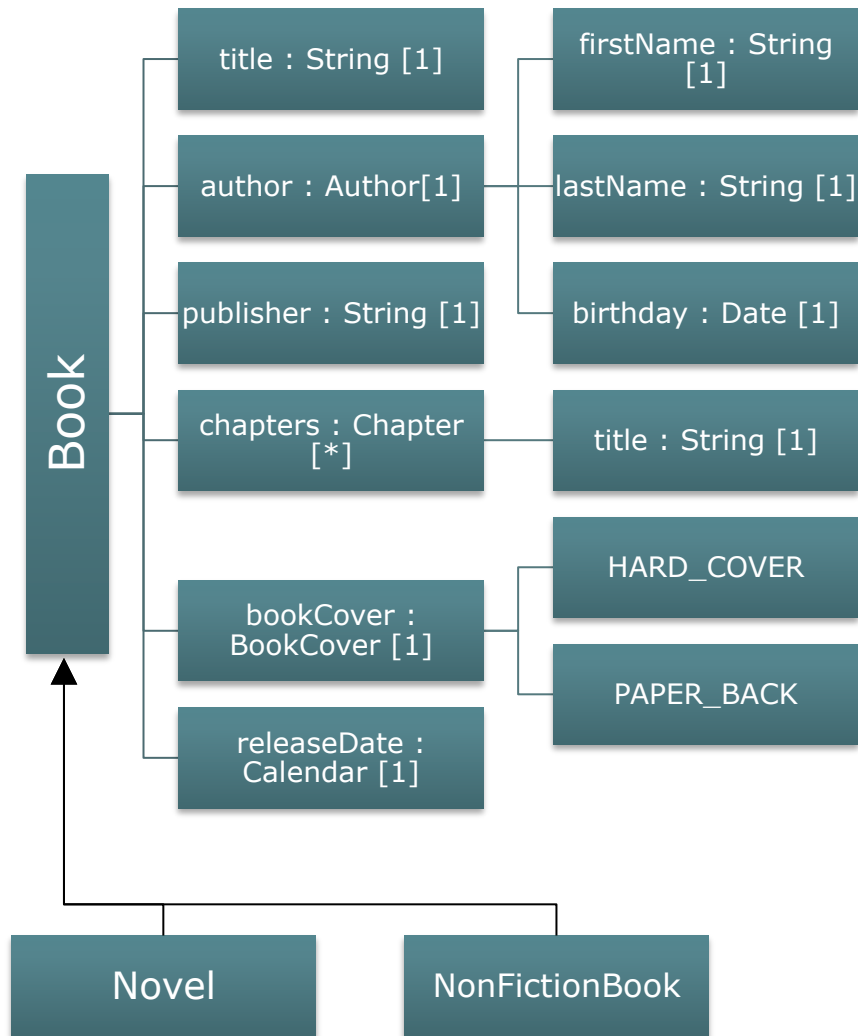
Gruppe 5 (Collection Mapping)



Gruppe 6 (Inheritance Mapping)

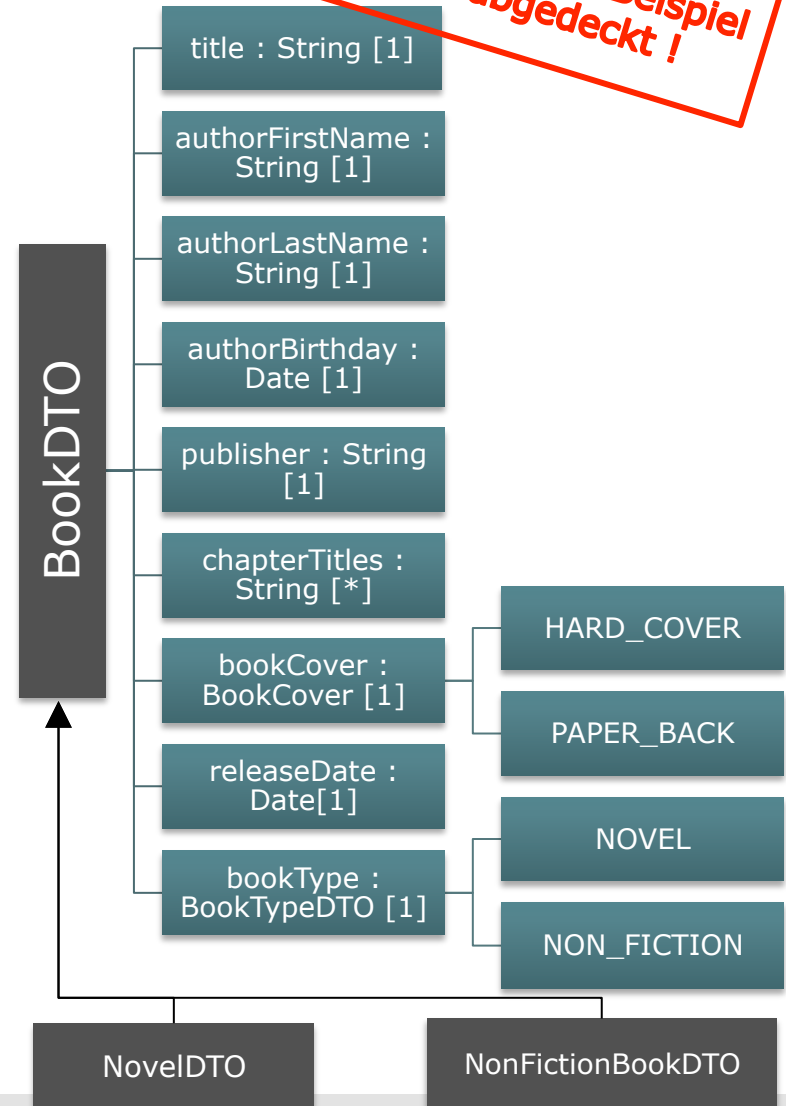
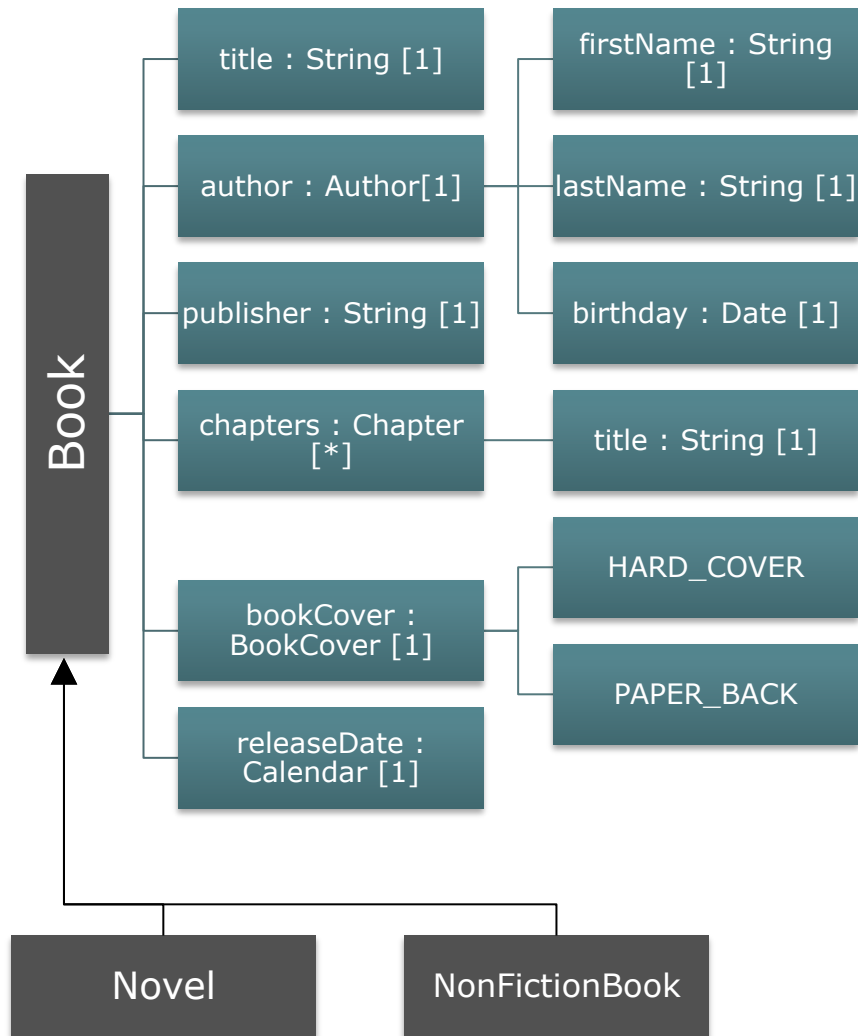


Gruppe 6 (Inheritance Mapping)

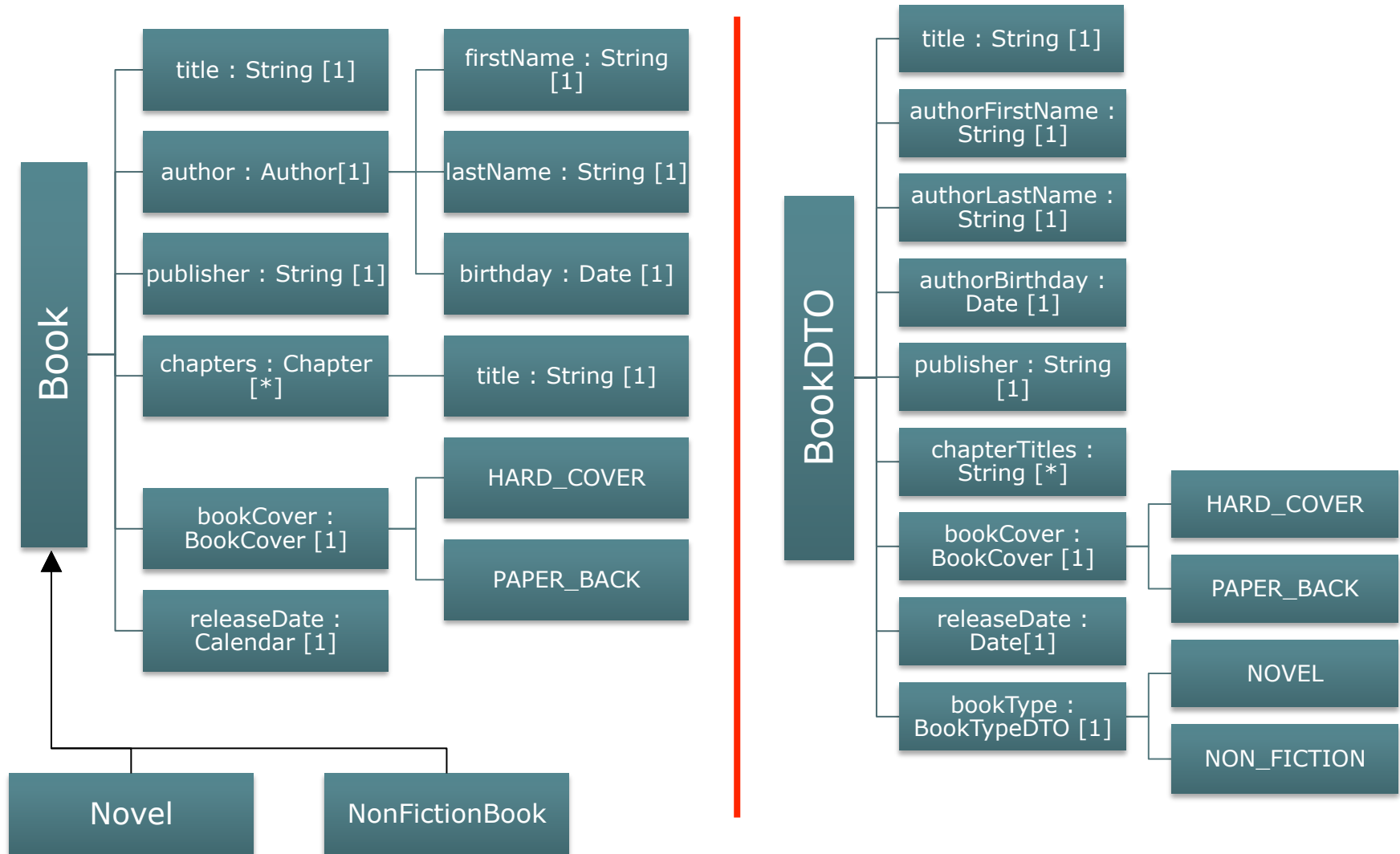


Wird durch das Beispiel
nicht abgedeckt !

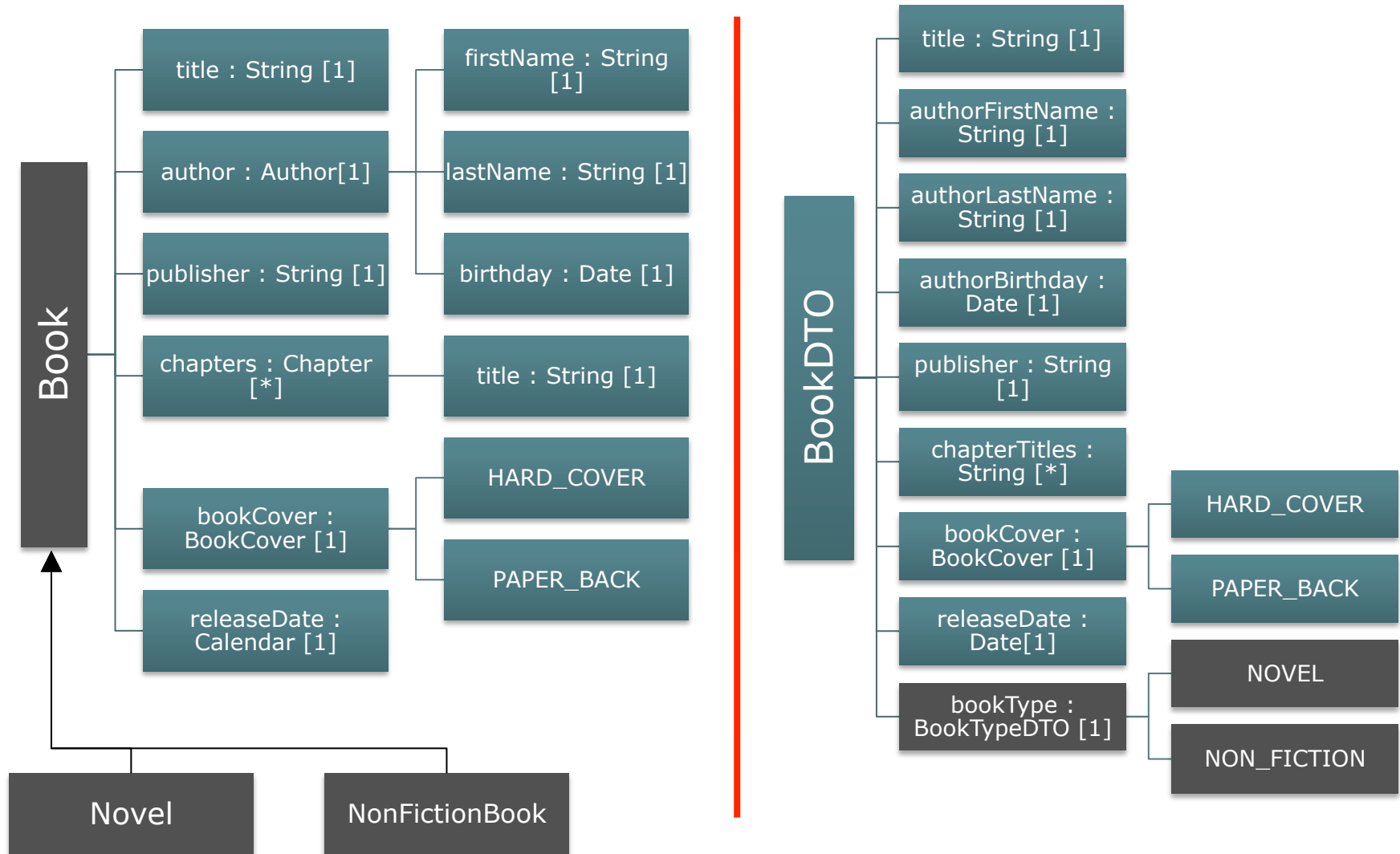
Gruppe 6 (Inheritance Mapping)



Gruppe 6 (Inheritance Mapping)



Gruppe 7 (Inheritance Enum Mapping)



1 Motivation – Wozu Mapping?

2 Anwendungsfallbeispiel

3 Handwritten Mapping

4 Auswahl der Mapping-Frameworks

5 Framework 1: Dozer

6 Framework 2: ModelMapper

7 Framework 3: Orika

8 Vergleich Performance

9 Bewertung der ausgewählten Frameworks

Der Code

```
public BookDTO mapEntity2DTO(Book book) {
    if (book == null) {
        return null;
    }
    BookDTO bookDTO = new BookDTO();

    bookDTO.setTitle(book.getTitle());
    bookDTO.setPublisher(book.getPublisher());

    if (book.getReleaseDate() != null) {
        bookDTO.setReleaseDate(book.getReleaseDate().getTime());
    }

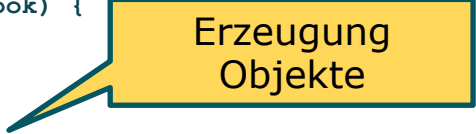
    Author author = book.getAuthor();
    if (author != null) {
        bookDTO.setAuthorFirstName(author.getFirstName());
        bookDTO.setAuthorLastName(author.getLastName());
        bookDTO.setAuthorBirthday(author.getBirthday());
    }

    if (book.getChapters() != null) {
        List<String> chapterTitles = new ArrayList<String>(book.getChapters().size());
        for (Chapter chapter : book.getChapters()) {
            chapterTitles.add(chapter.getTitle());
        }
        bookDTO.setChapterTitles(chapterTitles);
    }

    . . .
}
```


Der Code

```
public BookDTO mapEntity2DTO(Book book) {  
    if (book == null) {  
        return null;  
    }  
    BookDTO bookDTO = new BookDTO();  
  
    bookDTO.setTitle(book.getTitle());  
    bookDTO.setPublisher(book.getPublisher());  
  
    if (book.getReleaseDate() != null) {  
        bookDTO.setReleaseDate(book.getReleaseDate().getTime());  
    }  
  
    Author author = book.getAuthor();  
    if (author != null) {  
        bookDTO.setAuthorFirstName(author.getFirstName());  
        bookDTO.setAuthorLastName(author.getLastName());  
        bookDTO.setAuthorBirthday(author.getBirthday());  
    }  
  
    if (book.getChapters() != null) {  
        List<String> chapterTitles = new ArrayList<String>(book.getChapters().size());  
        for (Chapter chapter : book.getChapters()) {  
            chapterTitles.add(chapter.getTitle());  
        }  
        bookDTO.setChapterTitles(chapterTitles);  
    }  
    ...  
}
```



Der Code

```
public BookDTO mapEntity2DTO(Book book) {  
    if (book == null) {  
        return null;  
    }  
    BookDTO bookDTO = new BookDTO();  
  
    bookDTO.setTitle(book.getTitle());  
    bookDTO.setPublisher(book.getPublisher());  
  
    if (book.getReleaseDate() != null) {  
        bookDTO.setReleaseDate(book.getReleaseDate().getTime());  
    }  
  
    Author author = book.getAuthor();  
    if (author != null) {  
        bookDTO.setAuthorFirstName(author.getFirstName());  
        bookDTO.setAuthorLastName(author.getLastName());  
        bookDTO.setAuthorBirthday(author.getBirthday());  
    }  
  
    if (book.getChapters() != null) {  
        List<String> chapterTitles = new ArrayList<String>(book.getChapters().size());  
        for (Chapter chapter : book.getChapters()) {  
            chapterTitles.add(chapter.getTitle());  
        }  
        bookDTO.setChapterTitles(chapterTitles);  
    }  
    ...  
}
```

Erzeugung
Objekte

Umwandlung von
Date in Calendar

Der Code

```
public BookDTO mapEntity2DTO(Book book) {  
    if (book == null) {  
        return null;  
    }  
    BookDTO bookDTO = new BookDTO();  
  
    bookDTO.setTitle(book.getTitle());  
    bookDTO.setPublisher(book.getPublisher());  
  
    if (book.getReleaseDate() != null) {  
        bookDTO.setReleaseDate(book.getReleaseDate().getTime());  
    }  
  
    Author author = book.getAuthor();  
    if (author != null) {  
        bookDTO.setAuthorFirstName(author.getFirstName());  
        bookDTO.setAuthorLastName(author.getLastName());  
        bookDTO.setAuthorBirthday(author.getBirthday());  
    }  
  
    if (book.getChapters() != null) {  
        List<String> chapterTitles = new ArrayList<String>(book.getChapters().size());  
        for (Chapter chapter : book.getChapters()) {  
            chapterTitles.add(chapter.getTitle());  
        }  
        bookDTO.setChapterTitles(chapterTitles);  
    }  
    ...  
}
```

Erzeugung
Objekte

Umwandlung von
Date in Calendar

Null-Check nötig

Der Code

```
public BookDTO mapEntity2DTO(Book book) {  
    if (book == null) {  
        return null;  
    }  
    BookDTO bookDTO = new BookDTO();  
  
    bookDTO.setTitle(book.getTitle());  
    bookDTO.setPublisher(book.getPublisher());  
  
    if (book.getReleaseDate() != null) {  
        bookDTO.setReleaseDate(book.getReleaseDate().getTime());  
    }  
  
    Author author = book.getAuthor();  
    if (author != null) {  
        bookDTO.setAuthorFirstName(author.getFirstName());  
        bookDTO.setAuthorLastName(author.getLastName());  
        bookDTO.setAuthorBirthday(author.getBirthday());  
    }  
  
    if (book.getChapters() != null) {  
        List<String> chapterTitles = new ArrayList<String>(book.getChapters().size());  
        for (Chapter chapter : book.getChapters()) {  
            chapterTitles.add(chapter.getTitle());  
        }  
        bookDTO.setChapterTitles(chapterTitles);  
    }  
    ...  
}
```

Erzeugung
Objekte

Umwandlung von
Date in Calendar

Null-Check nötig

Umwandlung von
Collections

Der Code

```
public BookDTO mapEntity2DTO(Book book) {  
    if (book == null) {  
        return null;  
    }  
    BookDTO bookDTO = new BookDTO();  
  
    bookDTO.setTitle(book.getTitle());  
    bookDTO.setPublisher(book.getPublisher());  
  
    if (book.getReleaseDate() != null) {  
        bookDTO.setReleaseDate(book.getReleaseDate().getTime());  
    }  
  
    Author author = book.getAuthor();  
    if (author != null) {  
        bookDTO.setAuthorFirstName(author.getFirstName());  
        bookDTO.setAuthorLastName(author.getLastName());  
        bookDTO.setAuthorBirthday(author.getBirthday());  
    }  
  
    if (book.getChapters() != null) {  
        List<String> chapterTitles = new ArrayList<String>(book.getChapters().size());  
        for (Chapter chapter : book.getChapters()) {  
            chapterTitles.add(chapter.getTitle());  
        }  
        bookDTO.setChapterTitles(chapterTitles);  
    }  
    ...  
}
```

Erzeugung
Objekte

Umwandlung von
Date in Calendar

Null-Check nötig

Umwandlung von
Collections

... und das ist
noch nicht alles!

Übersicht: Handwritten Mapping

Vorteile:

- **Handwritten Mapping kann jeder Java-Entwickler leicht nachvollziehen.**
- **Debugging und Refactoring wird häufig durch die IDE unterstützt.**
- **Mapping von Enums auf Klassenhierarchie ist leicht zu erstellen.**

Nachteile

- **Änderungen müssen unter großem Aufwand nachgepflegt werden.**
- **Vergessene Null-Checks führen häufig zu Laufzeitfehlern.**
- **Trotz IDE-Unterstützung ist ein Refactoring sehr mühsam.**
- **Führt zu viel Code.**

- 1 Motivation - Wozu Mapping?
- 2 Anwendungsfallbeispiel
- 3 Handwritten Mapping
- 4 Auswahl der Mapping-Frameworks**
- 5 Framework 1: Dozer
- 6 Framework 2: ModelMapper
- 7 Framework 3: Orika
- 8 Vergleich Performance
- 9 Bewertung der ausgewählten Frameworks

Welches Mapping-Framework kann was leisten?

Framework	Version	Datum	Simple	Enum	Type Conv.	Deep Object	Collection	Inheritance	Inheritance Enums
			G1	G2	G3	G4	G5	G6	G7
Modelmapper	0.7.2	26.06.14	x	x	x	x	o	o	o
Orika	1.4.5	14.03.14	x	x	x	x	x	x	o
Dozer	5.5.1	22.04.14	x	x	x	x	x	x	o
OMapper	2.0	18.06.13	x						
JMapper	1.2.0	16.02.13	x	o	o	o		o	o
Moo	2.0	23.05.14	x						

- 1 Motivation - Wozu Mapping?
- 2 Anwendungsfallbeispiel
- 3 Handwritten Mapping
- 4 Auswahl der Mapping-Frameworks
- 5 Framework 1: Dozer**
- 6 Framework 2: ModelMapper
- 7 Framework 3: Orika
- 8 Vergleich Performance
- 9 Bewertung der ausgewählten Frameworks

Steckbrief: Dozer

- **Mapping Definition: XML, Annotation, API**
- **Aktuelle Version: 5.5.1 (22.04.2014)**
- **Durchführung Mapping:**
 - Dozer arbeitet mit Reflection.
 - Komplexe Mappings können via XML, Annotationen oder einer API definiert werden.
- **Best Practices:**
 - Verwende Dozer als Singleton.
 - Nutze kleinere XML Dateien statt einer XML Datei zur Komponentisierung.

Der Code (Java)

```
public DozerMapper() {
    List<String> myMappingFiles = new ArrayList<String>();
    myMappingFiles.add("dozer-mapping-config.xml");

    mapper = new DozerBeanMapper();
    mapper.setMappingFiles(myMappingFiles);

}

@Override
public BookDTO mapEntity2DTO(Book book) {
    if (book == null) {
        return null;
    } else {
        return mapper.map(book, BookDTO.class);
    }
}
```

Der Code (XML)

```
...
<mapping map-null="false">
    <class-a bean-factory=
        "com.senacor.knowledgetalks.mappingframeworks.mappers.dozer.CustomDozerBookDT02BookBeanFactory">
        com.senacor.knowledgetalks.mappingframeworks.entities.Book</class-a>

    <class-b bean-factory=
        "com.senacor.knowledgetalks.mappingframeworks.mappers.dozer.CustomDozerBook2BookDT0BeanFactory">
        com.senacor.knowledgetalks.mappingframeworks.dtos.BookDT0</class-b>

    <field>
        <a>title</a>
        <b>title</b>
    </field>

    <field>
        <a>Author.firstName</a>
        <b>authorFirstName</b>
    </field>
    <field>
        <a>Author.lastName</a>
        <b>authorLastName</b>
    </field>

    <field>
        <a>Author.birthday</a>
        <b>authorBirthday</b>
    </field>
...
```

Der Code (XML)

Factories für Inheritance
Enum Mapping

```
...
<mapping map-null="false">
  <class-a bean-factory=
    "com.senacor.knowledgetalks.mappingframeworks.mappers.dozer.CustomDozerBookDT02BookBeanFactory">
    com.senacor.knowledgetalks.mappingframeworks.entities.Book</class-a>

  <class-b bean-factory=
    "com.senacor.knowledgetalks.mappingframeworks.mappers.dozer.CustomDozerBook2BookDT0BeanFactory">
    com.senacor.knowledgetalks.mappingframeworks.dtos.BookDT0</class-b>

  <field>
    <a>title</a>
    <b>title</b>
  </field>

  <field>
    <a>Author.firstName</a>
    <b>authorFirstName</b>
  </field>
  <field>
    <a>Author.lastName</a>
    <b>authorLastName</b>
  </field>

  <field>
    <a>Author.birthday</a>
    <b>authorBirthday</b>
  </field>
...

```

Bewertung: Dozer

- **Vorteile**
 - Automatisches Mapping bei Namensgleichheit.
 - Auswählbar ob nur eine Referenz kopiert werden soll (copy-by-Ref).
 - Integration mit Spring ist möglich.
- **Nachteile:**
 - Kein automatisches DeepMapping innerhalb von Collections.
 - Nur das Mapping via XML ist gut dokumentiert, aber zusätzliche Verwaltung von XML-Dateien nötig.
 - Mapping von null führt zu Fehlern.

- 1 Motivation - Wozu Mapping?
- 2 Anwendungsfallbeispiel
- 3 Handwritten Mapping
- 4 Auswahl der Mapping-Frameworks
- 5 Framework 1: Dozer
- 6 Framework 2: ModelMapper**
- 7 Framework 3: Orika
- 8 Vergleich Performance
- 9 Bewertung der ausgewählten Frameworks

Übersicht: ModelMapper

- **Mapping Definition: API**
- **Aktuelle Version: 0.7.2**
- **Durchführung Mapping:**
 - Modelmapper führt das Mapping per Reflection durch.
 - Beim ersten Mapping eines Objekts A auf ein Objekt B/einen Typen B wird für den Vorgang eine TypeMap angelegt.
 - Jedes weitere Mapping von $A \Rightarrow B$ verwendet die Informationen aus der TypeMap und fügt ggf. neue hinzu.
 - Für komplexere Problemstellungen stehen Converter und Provider zur Verfügung
- **Best Practices:**
 - Verwende für ein Mapping von $A \Rightarrow B$ stets die gleiche Instanz des Mappers
 - Verwaltung der Mapper als „Multiton“

Der Code

```
public BookDTO mapEntity2DTO(Book book) {  
    if (book == null) {  
        return null;  
    }  
  
    BookDTO result = this.modelMapper.map(book, BookDTO.class);  
  
    if(book instanceof Novel) {  
        result.setBookType(BookTypeDTO.NOVEL);  
    } else if(book instanceof NonFictionBook) {  
        result.setBookType(BookTypeDTO.NON_FICTION);  
    }  
  
    for(Chapter chapter : book.getChapters()) {  
        if(result.getChapterTitles() == null) {  
            result.setChapterTitles(new ArrayList<String>(book.getChapters().size()));  
        }  
        result.getChapterTitles().add(chapter.getTitle())  
    }  
  
    return result;  
}
```

In vielen Fällen ist
nur diese eine Zeile
notwendig...

Bewertung: ModelMapper

- **Vorteile:**
 - Automatisches Mapping bei Namensgleichheit
 - Loose Mapping Strategy: Automatisches Mapping bei Namensähnlichkeit
 - Deep Object Mapping quasi unsichtbar
 - Refactoring-sichere API
 - Integration mit Spring, GSon, Jackson...
- **Nachteile:**
 - Erstes Mapping langsam (Lernphase)
 - In einigen Fällen komplexe Konfiguration mit Providern und Convertern notwendig

- 1 Motivation - Wozu Mapping?
- 2 Anwendungsfallbeispiel
- 3 Handwritten Mapping
- 4 Auswahl der Mapping-Frameworks
- 5 Framework 1: Dozer
- 6 Framework 2: ModelMapper
- 7 Framework 3: Orika**
- 8 Vergleich Performance
- 9 Bewertung der ausgewählten Frameworks

Übersicht: Orika

- **Mapping Definition: API oder Annotation**
- **Aktuelle Version: 1.4.5 (14.03.2014)**
- **Durchführung Mapping:**
 - Orika generiert aus den Meta-Daten der Klassen Mapping Objekte.
 - Die Mapping Objekte werden dann dazu verwendet direkt zwischen den beiden Objektgraphen zu mappen.
 - Orika mappt gleiche Objektstrukturen automatisch.
 - Einfache Notation für Objekt-Transformationen.
- **Best Practices:**
 - Verwende die MapperFactory als Singleton.
 - Vermeide den automatischen MapperLookUp und verwende BoundMapperFactory.

Der Code

```
public void init(){
    MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();

    mapperFactory.registerObjectFactory(new BookDTOFactory(), TypeFactory.valueOf(BookDTO.class));
    mapperFactory.registerObjectFactory(new BookFactory(), TypeFactory.valueOf(Book.class));

    mapperFactory.classMap(Book.class, BookDTO.class)
        .field("title", "title")
        .field("author.firstName", "authorFirstName")
        .field("author.lastName", "authorLastName")
        .field("author.birthday", "authorBirthday")
        .field("publisher", "publisher")
        .field("releaseDate", "releaseDate")
        .field("chapters{title}", "chapterTitles{}")
        .field("bookCover", "bookCover")

        .register();

    mapperEntity2DTO = mapperFactory.getMapperFacade(Book.class, BookDTO.class);
    mapperDTO2Entity = mapperFactory.getMapperFacade(BookDTO.class, Book.class);
}

public BookDTO mapEntity2DTO(Book book) {
    return mapperEntity2DTO.map(book);
}
```

Der Code

```
public void init(){
    MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();

    mapperFactory.registerObjectFactory(new BookDTOFactory(), TypeFactory.valueOf(BookDTO.class));
    mapperFactory.registerObjectFactory(new BookFactory(), TypeFactory.valueOf(Book.class));

    mapperFactory.classMap(Book.class, BookDTO.class)
        .field("title", "title")
        .field("author.firstName", "authorFirstName")
        .field("author.lastName", "authorLastName")
        .field("author.birthday", "authorBirthday")
        .field("publisher", "publisher")
        .field("releaseDate", "releaseDate")
        .field("chapters{title}", "chapterTitles{}")
        .field("bookCover", "bookCover")

        .register();

    mapperEntity2DTO = mapperFactory.getMapperFacade(Book.class, BookDTO.class);
    mapperDTO2Entity = mapperFactory.getMapperFacade(BookDTO.class, Book.class);
}

public BookDTO mapEntity2DTO(Book book) {
    return mapperEntity2DTO.map(book);
}
```

Factories für Inheritance
Enum Mapping

Der Code

```
public void init(){
    MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();

    mapperFactory.registerObjectFactory(new BookDTOFactory(), TypeFactory.valueOf(BookDTO.class));
    mapperFactory.registerObjectFactory(new BookFactory(), TypeFactory.valueOf(Book.class));

    mapperFactory.classMap(Book.class, BookDTO.class)
        .field("title", "title")
        .field("author.firstName", "authorFirstName")
        .field("author.lastName", "authorLastName")
        .field("author.birthday", "authorBirthday")
        .field("publisher", "publisher")
        .field("releaseDate", "releaseDate")
        .field("chapters{title}", "chapterTitles{}")
        .field("bookCover", "bookCover")

        .register();

    mapperEntity2DTO = mapperFactory.getMapperFacade(Book.class, BookDTO.class);
    mapperDTO2Entity = mapperFactory.getMapperFacade(BookDTO.class, Book.class);
}

public BookDTO mapEntity2DTO(Book book) {
    return mapperEntity2DTO.map(book);
}
```

Factories für Inheritance
Enum Mapping

Verwendung der
Punkt-Notation für
Deep Object Mapping

Der Code

```
public void init(){
    MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();

    mapperFactory.registerObjectFactory(new BookDTOFactory(), TypeFactory.valueOf(BookDTO.class));
    mapperFactory.registerObjectFactory(new BookFactory(), TypeFactory.valueOf(Book.class));

    mapperFactory.classMap(Book.class, BookDTO.class)
        .field("title", "title")
        .field("author.firstName", "authorFirstName")
        .field("author.lastName", "authorLastName")
        .field("author.birthday", "authorBirthday")
        .field("publisher", "publisher")
        .field("releaseDate", "releaseDate")
        .field("chapters{title}", "chapterTitles{}")
        .field("bookCover", "bookCover")

        .register();

    mapperEntity2DTO = mapperFactory.getMapperFacade(Book.class, BookDTO.class);
    mapperDTO2Entity = mapperFactory.getMapperFacade(BookDTO.class, Book.class);
}

public BookDTO mapEntity2DTO(Book book) {
    return mapperEntity2DTO.map(book);
}
```

Factories für Inheritance
Enum Mapping

Verwendung der
Punkt-Notation für
Deep Object Mapping

Verwendung von {} zur
Manipulation von Collections

Der Code

```
public void init(){
    MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();

    mapperFactory.registerObjectFactory(new BookDTOFactory(), TypeFactory.valueOf(BookDTO.class));
    mapperFactory.registerObjectFactory(new BookFactory(), TypeFactory.valueOf(Book.class));

    mapperFactory.classMap(Book.class, BookDTO.class)
        .field("title", "title")
        .field("author.firstName", "authorFirstName")
        .field("author.lastName", "authorLastName")
        .field("author.birthday", "authorBirthday")
        .field("publisher", "publisher")
        .field("releaseDate", "releaseDate")
        .field("chapters{title}", "chapterTitles{}")
        .field("bookCover", "bookCover")

        .register();

    mapperEntity2DTO = mapperFactory.getMapperFacade(Book.class, BookDTO.class);
    mapperDTO2Entity = mapperFactory.getMapperFacade(BookDTO.class, Book.class);
}

public BookDTO mapEntity2DTO(Book book) {
    return mapperEntity2DTO.map(book);
}
```

Der Code

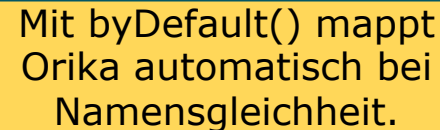
```
public void init(){
    MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();

    mapperFactory.registerObjectFactory(new BookDTOFactory(), TypeFactory.valueOf(BookDTO.class));
    mapperFactory.registerObjectFactory(new BookFactory(), TypeFactory.valueOf(Book.class));

    mapperFactory.classMap(Book.class, BookDTO.class)
//        .field("title", "title")
//        .field("author.firstName", "authorFirstName")
//        .field("author.lastName", "authorLastName")
//        .field("author.birthday", "authorBirthday")
//        .field("publisher", "publisher")
//        .field("releaseDate", "releaseDate")
//        .field("chapters{title}", "chapterTitles{}")
//        .field("bookCover", "bookCover")
        .byDefault()
        .register();

    mapperEntity2DTO = mapperFactory.getMapperFacade(Book.class, BookDTO.class);
    mapperDTO2Entity = mapperFactory.getMapperFacade(BookDTO.class, Book.class);
}

public BookDTO mapEntity2DTO(Book book) {
    return mapperEntity2DTO.map(book);
}
```



Mit `byDefault()` mappt Orika automatisch bei Namensgleichheit.

Bewertung: Orika

- **Vorteile:**

- `byDefault()`-Mapping verkürzt die Mapping-Definitionen.
- Mapping-Definitionen gelten in beide Richtungen.
- Einfache Behandlung von DeepObject Mapping.
- Intuitive Notation zur Behandlung von Datenstrukturen (z.B Collections, Maps, ...).
- Integration mit Spring ist möglich.

- **Nachteile:**

- Bei der Mapping-Definition müssen die Attribute als String benannt werden, je nach IDE nicht refactoring-safe.
- Erstes Mapping langsam (Lernphase)

- 1 Motivation - Wozu Mapping?
- 2 Anwendungsfallbeispiel
- 3 Handwritten Mapping
- 4 Auswahl der Mapping-Frameworks
- 5 Framework 1: Dozer
- 6 Framework 2: ModelMapper
- 7 Framework 3: Orika
- 8 Vergleich Performance**
- 9 Bewertung der ausgewählten Frameworks

Mapping Performance - Ergebnis der Evaluation

Mapping von 1000 randomisierten Objekten.

Test von beiden Richtungen (Entity -> DTO und DTO -> Entity)

Testcomputer Apple MacBookPro (2,3GHz Quad-Core;16 GB DDR3 RAM;256 GB SSD)

Angegebene Zeit entspricht dem Mapping von einem Element in NanoSekunden

Mapper	Entity -> DTO	DTO -> Entity
Handwritten	3.100 ns	14.780 ns
Orika	28.613 ns	26.514 ns
Modellmapper	96.908 ns	37.753 ns
Dozer	946.877 ns	569.694 ns

Mapping Performance - Ergebnis der Evaluation

Mapping von 10000 randomisierten Objekten.

Test von beiden Richtungen (Entity -> DTO und DTO -> Entity)

Testcomputer Apple MacBookPro (2,3GHz Quad-Core;16 GB DDR3 RAM;256 GB SSD)

Angegebene Zeit entspricht dem Mapping von einem Element in NanoSekunden

Mapper	Entity -> DTO	DTO -> Entity
Handwritten	491 ns	3.522 ns
Orika	4.975 ns	7.675 ns
Modellmapper	30.755 ns	20.322 ns
Dozer	317.625 ns	266.036 ns

Mapping Performance - Ergebnis der Evaluation

Mapping von 100000 randomisierten Objekten.

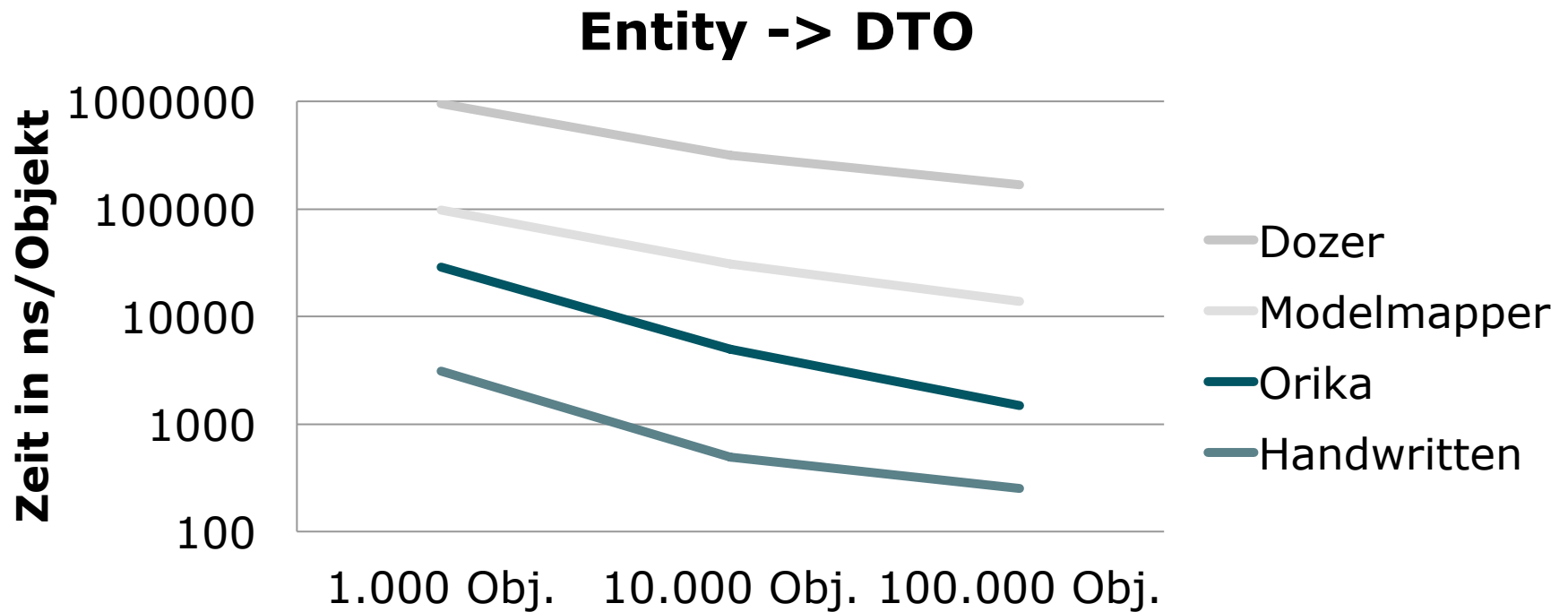
Test von beiden Richtungen (Entity -> DTO und DTO -> Entity)

Testcomputer Apple MacBookPro (2,3GHz Quad-Core;16 GB DDR3 RAM;256 GB SSD)

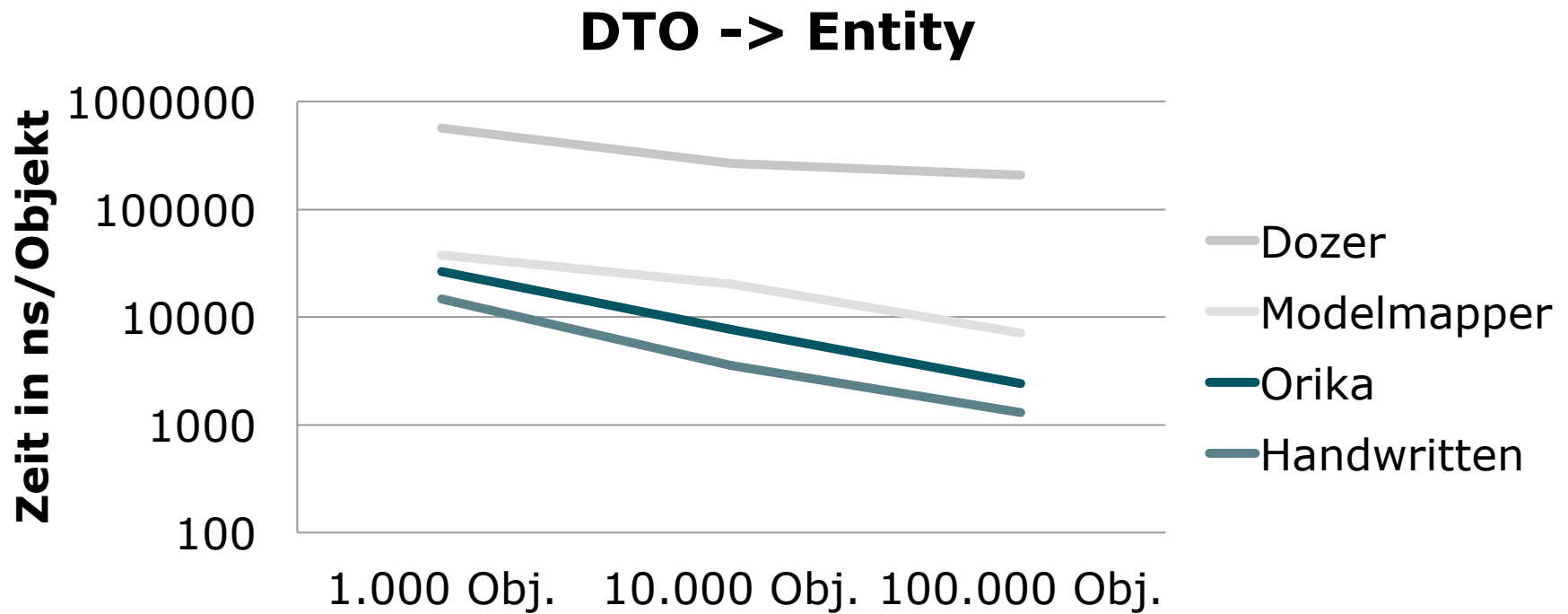
Angegebene Zeit entspricht dem Mapping von einem Element in NanoSekunden

Mapper	Entity -> DTO	DTO -> Entity
Handwritten	250 ns	1.295 ns
Orika	1.473 ns	2.407 ns
Modellmapper	13.813 ns	7.104 ns
Dozer	167.818 ns	207.261 ns

Mapping Performance - Ergebnis der Evaluation



Mapping Performance - Ergebnis der Evaluation



- 1 Motivation - Wozu Mapping?
- 2 Anwendungsfallbeispiel
- 3 Handwritten Mapping
- 4 Auswahl der Mapping-Frameworks
- 5 Framework 1: Dozer
- 6 Framework 2: ModelMapper
- 7 Framework 3: Orika
- 8 Vergleich Performance
- 9 Bewertung der ausgewählten Frameworks**

Welches Framework können wir empfehlen?

ORIKA

- ... beste Performance
- ... mappt (fast) alles problemlos
- ... einfache, eingängige Mappings

MODELMAPPER

- ... refactoring-sichere API
- ... bleibt bei kleinen Änderungen stabil

Noch Fragen?

