# Exercises: Exam Cheat

This document defines the **exercise assignments** for the ["Databases Advanced – Hibernate" course @ Software University.](https://softuni.bg/trainings/1444/databases-advanced-hibernate-october-2016)

Carlos Ruiz Zafón once wrote “Books are mirrors: you only see in them what you already have inside you.” Since you love his books a lot you decided to write an application about him that will serve you as an **exam cheat**.

## Setup

We have the following classes:

### **Author**

* Id
* Name
* Set<Book>

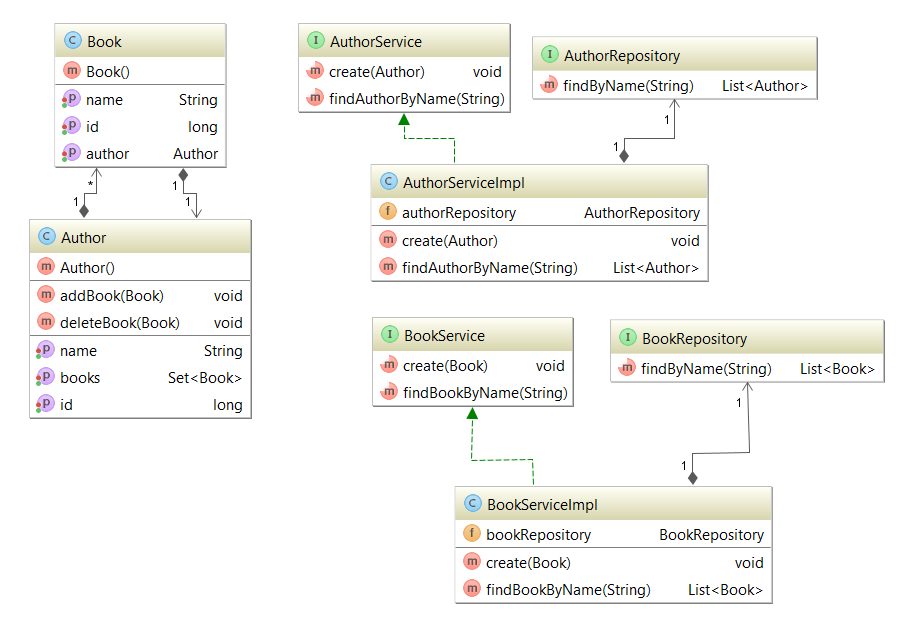
### **Book**

* Id
* Name
* Author

We have to create a repositories and services with the following functionalities:

* Create
* FindByName

Here is the overal structure:



### **Models:**

@Entity  
@Table(name = **"authors"**)  
**public class** Author {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"id"**)  
 **private long id**;  
  
 @Column(name = **"name"**)  
 **private** String **name**;  
  
 @OneToMany(mappedBy = **"author"**, targetEntity = Book.**class**, cascade = CascadeType.ALL)  
 **private** Set<Book> **books**;  
  
 **public** Author() {  
 **this**.setBooks(**new** HashSet<>());  
 }  
   
 **public void** addBook(Book book){  
 **this**.getBooks().add(book);  
 }  
   
 **public void** deleteBook(Book book){  
 **this**.getBooks().remove(book);  
 }  
}

@Entity  
@Table(name = **"books"**)  
**public class** Book {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"id"**)  
 **private long id**;  
  
 @Column(name = **"name"**)  
 **private** String **name**;  
  
 @ManyToOne  
 @JoinColumn(name = **"author\_id"**, referencedColumnName = **"id"**)  
 **private** Author **author**;  
  
 **public** Book() {  
 }  
}

### **Repositories:**

@Repository  
**public interface** AuthorRepository **extends** JpaRepository<Author, Long>{  
  
 List<Author> findByName(String name);  
}

@Repository  
**public interface** BookRepository **extends** JpaRepository<Book, Long> {  
  
 List<Book> findByName(String name);  
}

**Services:**

**public interface** AuthorService {  
  
 **void** create(Author author);  
  
 List<Author> findAuthorByName(String name);  
}

**public interface** BookService {  
  
 **void** create(Book book);  
  
 List<Book> findBookByName(String name);  
}

**Service Implementations:**

@Service  
**public class** AuthorServiceImpl **implements** AuthorService {  
  
 @Autowired  
 **private** AuthorRepository **authorRepository**;  
  
 @Override  
 **public void** create(Author author) {  
 **this**.**authorRepository**.saveAndFlush(author);  
 }  
  
 @Override  
 **public** List<Author> findAuthorByName(String name) {  
 **return this**.**authorRepository**.findByName(name);  
 }  
}

@Service  
**public class** BookServiceImpl **implements** BookService {  
  
 @Autowired  
 **private** BookRepository **bookRepository**;  
  
 @Override  
 **public void** create(Book book) {  
 **this**.**bookRepository**.saveAndFlush(book);  
 }  
  
 @Override  
 **public** List<Book> findBookByName(String name) {  
 **return this**.**bookRepository**.findByName(name);  
 }  
}

**Writing Data:**

@Override  
**public void** run(String... strings) **throws** Exception {  
  
 *//Author* Author author = **new** Author();  
 author.setName(**"Carlos Ruiz Zafón"**);  
 *//Book 1* Book marina = **new** Book();  
 marina.setName(**"Marina"**);  
 marina.setAuthor(author);  
 *//Book 2* Book shadowOfTheWind = **new** Book();  
 shadowOfTheWind.setName(**"Shadow of the wind"**);  
 shadowOfTheWind.setAuthor(author);  
 *//Add books* author.addBook(marina);  
 author.addBook(shadowOfTheWind);  
 **authorService**.create(author);  
}

## JSON

In order to parse in and from JSON we need an external library called **GSON.** Here is the dependency:

<**dependency**>  
 <**groupId**>com.google.code.gson</**groupId**>  
 <**artifactId**>gson</**artifactId**>  
 <**version**>2.6.2</**version**>  
</**dependency**>

## Write JSON

**Use @Expose annotation to choose which fields you want exported.**

@Entity  
@Table(name = **"authors"**)  
**public class** Author {  
  
 @Expose  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"id"**)  
 **private long id**;  
  
 @Expose  
 @Column(name = **"name"**)  
 **private** String **name**;  
  
 @Expose  
 @OneToMany(mappedBy = **"author"**, targetEntity = Book.**class**, cascade = CascadeType.***ALL***)  
 **private** Set<Book> **books**;

}

@Entity  
@Table(name = **"books"**)  
**public class** Book {  
  
 @Expose  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"id"**)  
 **private long id**;  
  
 @Expose  
 @Column(name = **"name"**)  
 **private** String **name**;

**//here we dont want to export the author due to recursive call**  
 @ManyToOne  
 @JoinColumn(name = **"author\_id"**, referencedColumnName = **"id"**)  
 **private** Author **author**;

}

**Code to export to JSON**

Gson gson = **new** GsonBuilder().excludeFieldsWithoutExposeAnnotation().setPrettyPrinting().create();  
String authorJson = gson.toJson(author);  
File outputFile = **new** File(**"src/main/resources/files/output/json/author.json"**);  
**try** (  
 BufferedWriter bufferedWriter = **new** BufferedWriter(**new** FileWriter(outputFile));  
) {  
 bufferedWriter.write(authorJson);  
} **catch** (FileNotFoundException e){  
 e.printStackTrace();  
}

**Result**

{  
 **"id"**: 1,  
 **"name"**: **"Carlos Ruiz Zafón"**,  
 **"books"**: [  
 {  
 **"id"**: 1,  
 **"name"**: **"Shadow of the wind"** },  
 {  
 **"id"**: 2,  
 **"name"**: **"Marina"** }  
 ]  
}

## Read JSON

### **File to read:**

{  
 **"id"**: 2,  
 **"name"**: **"Ivan Vazov"**,  
 **"books"**: [  
 {  
 **"id"**: 3,  
 **"name"**: **"Under the Yoke"** },  
 {  
 **"id"**: 4,  
 **"name"**: **"Epic of the Forgotten"** },  
 {  
 **"id"**: 5,  
 **"name"**: **"Uncles"** }  
 ]  
}

**Code to import from JSON**

Gson gson = **new** GsonBuilder().excludeFieldsWithoutExposeAnnotation().setPrettyPrinting().create();  
File inputFile = **new** File(**"src/main/resources/files/input/json/vazov.json"**);  
StringBuilder jsonData = **new** StringBuilder();  
**try** (BufferedReader bfr = **new** BufferedReader(**new** FileReader(inputFile));  
) {  
 String line;  
 **while** ((line = bfr.readLine()) != **null**) {  
 jsonData.append(line);  
 }  
} **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
}  
Author vazov = gson.fromJson(jsonData.toString(), Author.**class**);  
vazov.getBooks().stream().forEach(b -> b.setAuthor(vazov));  
**this**.**authorService**.create(vazov);

## XML

In order to parse in and from XML we need an external library called **JAXB.** Here is the dependency:

<**dependency**>  
 <**groupId**>com.sun.xml.bind</**groupId**>  
 <**artifactId**>jaxb-impl</**artifactId**>  
 <**version**>2.2.7</**version**>  
</**dependency**>

**Annotations:**

* **@XmlRootElement – Root of the XML. It will be the beginning tag.**
* **@XmlAccessorType(XmlAccessType.FIELD) – put your annotations on fields only**
* **@XmlAttribute – makrs the field as an attribute**
* **@XmlElement – makrs the field as an element**
* **@XmlElementWrapper(name = "books") – wraps the array of books with tag “books”**
* **@XmlTransient – the field won’t be exported/imported**

## Write XML

@XmlRootElement  
@XmlAccessorType(XmlAccessType.***FIELD***)  
@Entity  
@Table(name = **"authors"**)  
**public class** Author **implements** Serializable {  
  
 @XmlAttribute  
 @Expose  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"id"**)  
 **private long id**;  
  
 @XmlElement  
 @Expose  
 @Column(name = **"name"**)  
 **private** String **name**;  
  
 @XmlElementWrapper(name = **"books"**)  
 @XmlElement  
 @Expose  
 @OneToMany(mappedBy = **"author"**, targetEntity = Book.**class**, cascade = CascadeType.***ALL***)  
 **private** Set<Book> **books**;

}

@XmlRootElement  
@XmlAccessorType(XmlAccessType.***FIELD***)  
@Entity  
@Table(name = **"books"**)  
**public class** Book **implements** Serializable {  
  
 @XmlAttribute  
 @Expose  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"id"**)  
 **private long id**;  
  
 @XmlElement  
 @Expose  
 @Column(name = **"name"**)  
 **private** String **name**;  
  
 @XmlTransient**//Excluded due to recursive call**  
 @ManyToOne  
 @JoinColumn(name = **"author\_id"**, referencedColumnName = **"id"**)  
 **private** Author **author**;

}

**Code to export to XML**

JAXBContext jaxbContext = JAXBContext.*newInstance*(author.getClass());  
Marshaller jaxbMarshaller = jaxbContext.createMarshaller();  
jaxbMarshaller.setProperty(Marshaller.***JAXB\_FORMATTED\_OUTPUT***, Boolean.***TRUE***);  
File outputXMLFile = **new** File(**"src/main/resources/files/output/xml/author.xml"**);  
jaxbMarshaller.marshal(author, outputXMLFile);

**Result**

*<?***xml version="1.0" encoding="UTF-8" standalone="yes"***?>*<**author id="1"**>  
 <**name**>Carlos Ruiz Zafón</**name**>  
 <**books**>  
 <**books id="1"**>  
 <**name**>Marina</**name**>  
 </**books**>  
 <**books id="2"**>  
 <**name**>Shadow of the wind</**name**>  
 </**books**>  
 </**books**>  
</**author**>

## Read XML

### **File to read:**

*<?***xml version="1.0" encoding="UTF-8" standalone="yes"***?>*<**author id="3"**>  
 <**name**>Karl May</**name**>  
 <**books**>  
 <**books id="7"**>  
 <**name**>Hantu Llano Estacado</**name**>  
 </**books**>  
 <**books id="8"**>  
 <**name**>Winnetou I: Kepala Suku Apache</**name**>  
 </**books**>  
 </**books**>  
</**author**>

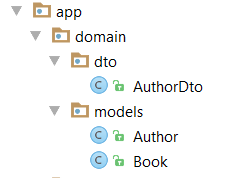
**Code to import XML**

JAXBContext jaxbContext = JAXBContext.*newInstance*(Author.**class**);  
Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();  
File outputXMLFile = **new** File(**"src/main/resources/files/input/xml/author.xml"**);  
Author authorFromXML = (Author) unmarshaller.unmarshal(outputXMLFile);  
authorFromXML.getBooks().stream().forEach(b -> b.setAuthor(authorFromXML));  
**this**.**authorService**.create(authorFromXML);

## DTO

DTO – data transfer object. Serves to transfer data from one point to another.

Example: AuthorDto will carry only the name because we may need to send only the name to JSON/XML.



**public class** AuthorDto **implements** Serializable {  
   
 **private** String **name**;  
  
 **public** String getName() {  
 **return this**.**name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
}

## EntityToDto

List<Author> authors = **this**.**authorService**.findAuthorByName(**"Carlos Ruiz Zafón"**);  
List<AuthorDto> authorDtos = **new** ArrayList<>();  
**for** (Author a : authors) {  
 AuthorDto authorDto = **new** AuthorDto();  
 authorDto.setName(a.getName());  
 authorDtos.add(authorDto);  
}

## DtoToEntity

AuthorDto authorDto = **new** AuthorDto();  
authorDto.setName(**"Elin Pelin"**);  
Author authorEntity = **new** Author();  
author.setName(authorDto.getName());  
**this**.**authorService**.create(authorEntity);