Association Mapping (AMP)

Exercise AMP.1 – Adding an Association

The Setup:

In this exercise we will be extending the code from **Lab2-Hibernate-1** to create a basic entity relationship between the **Car** class and a new **Owner** class.

Copy the **Lab2-Hibernate-1** project and change the directory name to **Lab4-Associations-1**. Be sure to also rename the <artifactId> inside the **pom.xml**. Or create everything from scratch(suggested).

The Exercise:

Create a new class **Owner** with the following attributes:

```
public class Owner {
      private Long id;
      private String name;
      private String address;
}
```

Start by generate the needed getters, setters, and constructors, and adding the persistence annotations. If you want you can test your class by creating an Owner object and persisting it to the database.

Then add a uni-directional ManyToOne association from Car to Owner that cascades on persistence In other words, when a Car is persisted, Hibernate should automatically also persist its Owner.

Update the main method in **App.java** in so that it creates two cars, and associates an owner with each one before persisting it. Then when retrieving the cars also print the details of each owner.

Collection Mapping (CMP)

Exercise CMP.1 – Mapping Collections

The Setup:

The main objective of this exercise is to practice mapping the different types of collections.

Start this exercise by creating a new project called **Lab4-Collections**. Remember to add the hibernate, mysql, and log4j dependencies to the pom.xml.

The exercise consists out of 3 parts. You will have to create the entity classes used for each part. Remember to use the code-generating features of your IDE.

The Database:

During this exercise, or the next exercise the database may already have an Employee or Student table, while we're trying to add a new (different) Employee or student. This can cause an error something like what is shown below:

```
org.hibernate.exception.SQLGrammarException: could not insert:
[collections.map.Student]
        ...
        ...
Caused by: java.sql.SQLException: Column not found: ID in statement [/*
insert ordering.Book */ insert into Book (title, id) values (?, ?)]
        ...
```

This error is caused by the difference between the Employee and Student classes from this exercise and another exercises. You can fix this error by dropping the database and recreating it.

```
DROP DATABASE cs544;
CREATE DATABASE cs544;
```

The Exercise:

Be sure to also update App.java with some code to test each of the following mappings.

a) Create a **Bidirectional OneToMany**/**ManyToOne** association between **Employee** and **Laptop** using a **Set**.

b) Create a **Unidirectional OneToMany** between **Passenger** and **Flight** using a **List**.

Please note that 'TO' and 'FROM' are SQL keywords and should either be escaped with backticks like `to` or changed to a different column name using the @Column annotation

c) Create a **Unidirectional OneToMany** association between **School** and **Student** using a **Map**, where **studentId** is used as the key for the map (in this scenario studentid needs to be an assigned value, not generated).

Exercise AMP.2 – More Associations

The Setup:

The main objective of this exercise is to practice creating different types of associations.

Start this exercise by creating a new project called **Lab4-Associations-2**. Remember to add the dependencies to the project's pom.xml file.

The exercise consists out of 6 parts; you will have to create an association between two entity classes in each. These entity classes are not provided, which means you will have to create them as needed for the different parts. Remember to use the code-generating features provided by your IDE when making these entity classes.

Be sure to update the main method for each of the following exercises so that you can test and see the results of the mapping in the database.

The Exercise:

a) Create a **Bidirectional OneToMany** association between **Department** and **Employee** using annotations.

b) Create an **Optional Unidirectional ManyToOne** association between **Book** and **Publisher** using annotations and without using NULL fields in the database

c) Create a **Bidirectional ManyToMany** association between **Student** and **Course** using annotations. Be sure to make studentid values application assigned (not generated)!

d) Create a **Unidirectional OneToMany** association between **Customer** and **Reservation** using annotations.

e) Create a **Unidirectional ManyToOne** association between **Reservation** and **Book** using annotations.

f) Create a **Bidirectional ManyToOne** association between **Employee** and **Office** using annotations.