Hibernate Relationships (@Many-To-One) Workshop

Recap:

In the previous workshops, we have created a project called HibernateProject.
In this project, we have a Student class. By using Hibernate, we could create a student table in our database, and we could manipulate the student table from Java (adding, deleting, retrieving and updating data).
We could even change the name of the table and the variables.
We have learned how to use field access and property access.

Instructions to use Derby

```
NetworkServerControl.bat -p 50000 start
```

In linux and mac users:

```
./NetworkServerControl -p 50000 start
```

Open another terminal and go to the bin directory again and from there run:
`ij.bat` or

```
./ij
```

```
connect 'jdbc:derby://localhost:50000/hibernate; create=true' ;
```

In this workshop:

- We continue with the same project.
- We will create a many to one relationship between classes Student and Tutor (we will create a class Tutor and accordingly it will create a table tutor in our database)

@Many-To-One

In Student class, we have a field which is *String tutorName*. First of all we change the type of this variable to Tutor which is a class that we will write.

Many students can be tutored by the same *tutor*. But a student can have only one tutor.
So this is a *many to one* relationship.

Create Tutor class

Create a class named *Tutor* with these attributes and constructor:

private String tutorId
private String name
private int salary
A constructor with the above properties as arguments.
Accessors (get methods)
A toString() method

Now we do the hibernate configurations for this class. And the necessary modifications in Student class.

Hibernate configuration means to have:
@*Entity* above the name of the class.
A private int id with @*Id* annotation (and generating it automatically via database)
An empty constructor.

This is the complete Tutor class:

```
package se.yrgo.domain;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Tutor {
	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	private int id;
	private String tutorId;
	private String name;
	private int salary;

	public Tutor() {}

	public Tutor(String tutorId,String name, int salary) {
		this.tutorId= tutorId;
		this.name=name;
```

```
            this.salary= salary;
      }

      public String getName() {
            return name;
      }
}
```

Changes in the Student class

Remove `@Table(name="TBL_STUDENT")` from above the name of the class

Now we should change tutorName to tutor and the type from String to Tutor in Student class:

```
private Tutor tutor;
```

(If you get a compiler error on Tutor, just leave it. We fix it very soon by adding the annotation).

Also change the type of tutor and the name of the field (tutor instead of tutorName) in the constructor, get method and toString method.

If you have getters and setters for tutorName, delete them too.

Add annotation ManyToOne

Add the @ManyToOne annotation to the Tutor field in the Student class:

```
@ManyToOne
private Tutor tutor;
```

You should import `jakarta.persistence.ManyToOne`

This annotation works here as a foreign key.

Add these two methods in your Student class

There are two methods that we need to add to our class Student:

```
public void allocateTutor(Tutor tutor) {
      this.tutor=tutor;
}

public String getTutorName() {
      return this.tutor.getName();
}
```

This is the complete Student class after changes:

```
package se.yrgo.domain;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Student{
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;
    private String enrollmentID;
    private String name;
    @ManyToOne
    private Tutor tutor;

    public Student() {}

    public Student(String name, Tutor tutor) {
      this.name = name;
      this.tutor = tutor;
    }

    public Student(String name) {
      this.name = name;
      this.tutor = null;
    }

    public int getId() {
```

```
      return id;
    }

    public void allocateTutor(Tutor tutor) {
      this.tutor=tutor;
    }

    public String getTutorName() {
      return this.tutor.getName();
    }
}
```

Changes in hibernate.cfg.xml

We should add the following to the hibernate.cfg file:

```
<mapping class="se.yrgo.domain.Tutor"/>
```

In HibernateTest class

In the HibernateTest class remove all the code except the sessionFactory, session and transaction code.
Add the following code between the beginTransaction and commit:

```
Tutor tutor = new Tutor("ABC123" ,"Edward", 30000);
Student student = new Student("Sara Hedborn");
student.allocateTutor(tutor);
System.out.println(student.getTutorName());
```

Persisting data

Now we want to save the data that we have created in our database:

```
session.save(student);
session.save(tutor);
```

Now run the code.

Running the code and checking in ij shell

Run the code. Check your tables in ij and the schema of both tables.

```
show tables;
describe tutor;
describe student;
```

You see that in the student table, we have a column *tutor_id* which is a foreign key and this is made because of the ***@ManyToOne*** annotation we added above tutor property in Student class.

Actually we can override the default name by adding:

```
@ManyToOne
@JoinColumn(name="TUTOR_FK")
private Tutor tutor;
```

To see the result, we should drop the table tutor, but we cannot. Why is it so?
To drop the tutor table, we should first drop the student table because there is dependency between these two tables in the student table.

Now run the code and see the changes in the table student.

In ij retrieve *name* and *tutor_fk* from *student* table and *id* and *name* from *tutor*.

More tests- Retrieve a student by id

Warning: The ids that are used in this workshop such as 1 or 2, are not the same as in your database. Find the right ids in your tables and use them instead.

Now in your main method, comment out or delete all the code that you wrote for creating, printing and saving in the database. Only session and transaction code should be there.
Add the following:

```
Student studentFromDatabase = session.get(Student.class, 1);
System.out.println(studentFromDatabase);
```

It returns the name of the student with id=1

We add a print line:

```
System.out.println(studentFromDatabase.getTutorName());
```

which prints out the name of the tutor for this student.

More tests- Get the name of the teacher for a specific student

(You need to have a getter method for the tutor in your Student class):

```
	public Tutor getTutor() {
		return tutor;
	}
```

Now we want to find the name of the teacher for a specific student:

```
Tutor tutorForStudentFromDatabase = studentFromDatabase.getTutor();
System.out.println(tutorForStudentFromDatabase.getName());
```

Run the code and see the result.

Update the tutor for a student

We want to update the tutor for a student. So we get the name of the teacher in our table and allocate this teacher to a student:

```
Tutor tutorFromDatabase = session.get(Tutor.class, 1);
```

This line will return a teacher with id=1

- check your tutor table and see which id you have if you don't have id number 1.

```
Student newStudent= new Student("Susan Lindberg");
newStudent.allocateTutor(tutorFromDatabase);
session.save(newStudent);
```

With the line above, we allocate or change the tutor for Susan Lindberg to a new tutor (`Edward`).

We can even change the tutor to null:

```
Student student2=session.get(Student.class, 2);
```

```
student2.allocateTutor(null);
```