Hibernate Relationships (@OneToMany) Workshop

Recap:

In the previous workshop, we have used annotation @ManyToOne to show that there is a relationship between students and a tutor. A student has only one tutor, but a tutor can teach one to many students.

This is the link to the workshop:
https://docs.google.com/document/d/17b21LM3U22SbVAQ4VEKCdq_U68QZImfReSXi2iMK4ig/edit#heading=h.ipdic2rpk58g

Content of this workshop

- Now we want to change the direction of the relation from many to one (in student class) to one to many (in tutor class)
- We create a one to many relation in Tutor class by adding a collection of students as an attribute in this class
- We use Set as the collection of the students
- In the test class, we create several students and a tutor
- Then we retrieve the students in a specific tutor's group
- We add a new student to the group
- We use JoinColumn annotation to create a foreign key in student table
- We use Map instead of Set and test creating, retrieving data and adding a new student.

@OneToMany

With the many to one relation which we did earlier the Student class has information about the tutor. But the Tutor class does not have any information about the students.

We can get this information by having a one to many relationship in tutor class.(And then student will not have any information about the tutor)

We change the direction of the relationship from *many to one* from Student class to Tutor class to *one to many* relationship from tutor to student.

If we want to use one to many, it means that the tutor class should have a collection of students as a group.

For the collection, we can use Set, Map or List. We begin with the Set:

In the Student class we comment out the following:

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

This is because we want to create the relationship between these two tables from the other side.
Naturally you should remove the allocateTutor method, getters and setters for tutor and Tutor in your constructor in the Student class.

This will be the Student class:

```
package se.yrgo.domain;


import jakarta.persistence.*;

@Entity
//@Table(name="TBL_STUDENT")
public class Student
{
   @Id
   @GeneratedValue(strategy= GenerationType.AUTO)
   private int id;
   private String enrollmentID;
   private String name;

   @Column(name="NUM_COURSES")
   private Integer numberOfCourses;


   public Student() {}

   public String getEnrollmentID() {
       return enrollmentID;
   }

   public Student(String name) {
       this.name = name;
```

```
    }


    public String toString() {
        return "name:" + name ;
    }
}
```

Using Set

In Tutor class, add a property for students with type Set<>.

```
private Set<Student>teachingGroup;
```

Add this annotation above the *Set<Student>*

```
@OneToMany
```

Add this property in the constructor of Tutor:

```
this.teachingGroup = new HashSet<Student>();
```

This will be the constructor:

```
public Tutor(String tutorId,String name, int salary) {
    this.tutorId= tutorId;
    this.name=name;
    this.salary= salary;
    this.teachingGroup = new HashSet<Student>();
}
```

A method to add students to this set in the Tutor class

We need a method for adding students to this teaching group:

```
public void addStudentToTeachingGroup(Student newStudent) {
    this.teachingGroup.add(newStudent);
}
```

A get method for this collection of students in the Tutor class

And this method to get the teachingGroup in Tutor class

```
public Set<Student> getTeachingGroup() {
   Set<Student>unmodifiable=
               Collections.unmodifiableSet(this.teachingGroup);
   return unmodifiable;
}
```

Collection has a method named unmodifiableSet which makes a new set. It ensures the encapsulation principle and prevents modifications to the set of students (teachingGroup) from outside the class via the returned set. So we want to expose a collection from a class but we want to ensure that its contents cannot be modified by external code, preventing unintended changes to the internal state of the class.

Creating students and tutor in the main method

In your main method, delete all the code from creating the objects to printing them (Keep sessionFactory, session and transaction)
Add the following:

```
      Tutor newTutor = new Tutor("ABC234", "Natalie Woodward", 387787);
      Student student1 = new Student("Patrik Howard");
      Student student2 = new Student("Marie Sani");
      Student student3 = new Student("Tom Nikson");

      newTutor.addStudentToTeachingGroup(student1);
      newTutor.addStudentToTeachingGroup(student2);
      newTutor.addStudentToTeachingGroup(student3);

      Set<Student>students = newTutor.getTeachingGroup();
      for(Student student: students) {
         System.out.println(student);
      }
```

So in the above, we have created a tutor and 3 students, and we have allocated these students to the teaching group of the tutor.
Add the students and the tutor to the database.

```
           session.save(student1);
```

```
            session.save(student2);
            session.save(student3);
            session.save(newTutor);
```

Drop tables and run the code

Now drop both student and tutor tables. And run the code.

In ij> run: `show tables;`
You see now that we have three tables: **student, tutor and tutor_student** which is a linked table.

Run these:

```
select * from student;
describe student;
```

You see that there is no foreign key created. But we have the tutor_student which has primary keys from both the student and tutor tables in rows.

```
select * from tutor_student;
```

Retrieving the students from a specific tutor's group

Now comment out all the code from creating tutor to the end of the loop in the main method and Add: (run select * from tutor; in ij and see which id the tutor got)

```
Tutor myTutor = session.get(Tutor.class, 1);
Set<Student>students = myTutor.getTeachingGroup();
for(Student s: students) {
      System.out.println(s);
}
```

This will print out all the students that belong to this tutor's teaching group.
name:Marie Sani
name:Tom Nikson
name:Patrik Howard

Adding a new student to the group

Now we want to add a new student to this teaching group.

```
Student student4 = new Student("Julia Ericcson");
session.save(student4);
myTutor.addStudentToTeachingGroup(student4);
```

Above, we have created a new student. We published it to our database, and we added this new student to the tutor's teaching group.

Run the code, and see the result in ij. (By running a select from student tables, then a select from tutor and finally a select from tutor_student)

JoinColumn annotation to create a foreign key in student table

As said before, there are no foreign keys in the tables, and instead we have a relation table. But if we want to change from this table to foreign key we can do as below:

In Tutor class add the annotation JoinColum with a name to the Set of students:

```
@OneToMany
@JoinColumn(name="TUTOR_FK")
Set<Student>teachingGroup;
```

Drop tables and run the code with the new feature `@JoinColumn`

Now do as following:

```
drop table tutor_student;
drop table student;
drop table tutor;
```

Remove or comment out the following code

```
Tutor myTutor = (Tutor)session.get(Tutor.class, 1);
Set<Student>students = myTutor.getTeachingGroup();
for(Student s: students) {
      System.out.println(s);
}

Student student4 = new Student("Julia Ericcson");
```

```
session.save(student4);
myTutor.addStudentToTutorGroup(student4);
```

In your main method add or uncomment the following:

```
Tutor newTutor = new Tutor("ABC234", "Natalie Woodward", 387787);
Student student1 = new Student("Patrik Howard");
Student student2 = new Student("Marie Sani");
Student student3 = new Student("Tom Nikson");
session.save(student1);
session.save(student2);
session.save(student3);
session.save(newTutor);
```

```
newTutor.addStudentToTeachingGroup(student1);
newTutor.addStudentToTeachingGroup(student2);
newTutor.addStudentToTeachingGroup(student3);

Set<Student>students = newTutor.getTeachingGroup();
for(Student student: students) {
   System.out.println(student);
}
```

Run the code again and see the result in ij by:

```
show tables;
```

Now there are only two tables.

```
describe student;
```

And we have a foreign key (tutor_fk) in the student table.

```
select id,name,tutor_fk from student;
```

Using Map

Have a copy of your Set code somewhere, because we would go back to Set after doing the map and list.

We have used Set to create a collection of students. We can use Map or List too. Now we want to use Map instead

Changes in Tutor class

Change the property in the Tutor class to:

```
private Map<String,Student>teachingGroup;
```

You know that a map has a pair of key values. Here the key would be *enrollmentId* which is String and value is *Student*.

Change all the Set in the Tutor class to Map:

In constructor:

```
this.teachingGroup = new HashMap<String,Student>();
```

In addStudent method:

```
this.teachingGroup.put(newStudent.getEnrollmentID(),newStudent);
```

And change the method *getTeachingGroup()* to:

```
public Map<String,Student>getTeachingGroup(){}
```

and inside the method:

```
Map<String, Student>unmodifiable= Collections.unmodifiableMap(this.teachingGroup);
```

So this will be the method:

```
public Map<String,Student>getTeachingGroup(){
   Map<String, Student>unmodifiable=
Collections.unmodifiableMap(this.teachingGroup);
   return unmodifiable;
}
```

After the annotation OneToMany in Tutor class add:

```
@MapKey(name="enrollmentID")
```

And this needs:

```
import jakarta.persistence.MapKey;
```

Changes in Student class

In the Student class, add enrollmentID in the constructor.

```
public Student(String name, String enrollmentID) {
   this.name = name;
    this.enrollmentID= enrollmentID;
}
```

Changes in the main method

In the main method:
Change the lines for creating the students to:

```
      Student student1 = new Student("Patrik Howard","1-HOW-2017");
      Student student2 = new Student("Marie Sani", "2-SAN-2018");
      Student student3 = new Student("Tom Nikson", "3-NIK-2019");
```

We are adding *enrollmentID* too.
Remove the set code.

Running the code and checking the database

In database drop tables student and tutor.
Run the code . And check if you have data in your tables.

More tests- Retrieve the students for a specific tutor

Now we want to do the query part in our main method.

Comment out the code from creating objects to adding them to the database.
Instead add the following code for querying:
(check your own tutor table and find the proper id)

```
Tutor myTutor = (Tutor)session.get(Tutor.class, 1);
Map<String,Student>students = myTutor.getTeachingGroup();
```

```
for(Student student: students.values()) {
    System.out.println(student);
}
```

Notice that we have changed Set to Map.
Run the code and see what you get when it prints them.

More tests- Retrieve a specific student from the students collection

```
Student s = students.get("3-NIK-2019");
System.out.println("found the student  " + s);
```

Run the code and see the result.