Hibernate Many to Many relation workshop

Recap

We have learned how to create a many to one and one to many relation using Hibernate. Now we want to create a many to many relation between two classes which will create three tables for us. One table for each class and the third table will be the relational table between these two tables.

Derby

Windows users:
In bin folder:
Run this via cmd:

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

Mac and linux users:
In bin folder:
Run this in terminal:

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

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

```
./ij   for mac and linux
```

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

@ManyToMany relationship

We will be using *sets* again. So change everything from List to Set or use the copy of the code with ‘set’.
If we want to use a many to many relationship, we should use a collection on each side of the relationship. Then we should add the *@ManyToMany* annotation on each collection.
We add *mappedBy* annotation on either side but only on one side.
A link table is always used and we do not need a class for it.

Content of this Workshop

- We create a Subject class in our project
- In the main method, we create and persist several subjects
- We make a many to many relation between Subject and Tutor by adding @ManyToOne annotations in both classes.
- We create methods in both classes to add an object to the collection
- We test our relationship in the main method
- We combine the two methods so we do not have to use both methods when adding an object to the collection.

Create Subject class

To test the ManyToMany relationship, we create another class in our project: *Subject* class. Subject class has these private attributes:

- *id: int*
- *subjectName: String*
- *numberOfSemesters: int*

These attributes are private so you need to add some methods to make them accessible. Put all the necessary annotations and create necessary construction for hibernate. Create a constructor with all these two attributes as arguments: subjectName and numberOfSemesters.

This would be the class Subject:

```
package se.yrgo.domain;

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


@Entity
public class Subject {

	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	private int id;

	private String subjectName;
```

```
	private int numberOfSemesters;


	public Subject() {}

	public Subject(String subjectName, int numberOfSemesters) {
		this.subjectName = subjectName;
		this.numberOfSemesters=numberOfSemesters;
	}

	public String subjectName() {
		return subjectName;
	}
}
```

Add Subject class to the hibernate.cfg.xml

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

Adding ManyToMany annotation in Subject class

Now we want to add this ManyToMany annotation to make a relationship between *subject* and *tutor* classes.

In Subject class we add:

```
@ManyToMany
private Set<Tutor>tutors;
```

Instantiate tutors in the constructor which has two parameters subjectName and numberOfSemesters:

```
public Subject(String subjectName, int numberOfSemesters) {
	this.subjectName = subjectName;
	this.numberOfSemesters=numberOfSemesters;
	this.tutors = new HashSet<Tutor>();
}
```

We need a method to add a new tutor to the group of teachers for the subject:

```
public void addTutorToSubject(Tutor tutor) {
```

```
      this.tutors.add(tutor);
}
```

Adding ManyToMany annotation in Tutor class

Now in Tutor class add the followings:

```
@ManyToMany(mappedBy="tutors")
private Set<Subject>subjectsToTeach;
```

A new method to add a subject to the collection of subjects

```
public void addSubjectsToTeach(Subject subject) {
      this.subjectsToTeach.add(subject);
}
```

Instantiate *subjectsToTeach* in the constructor:

```
this.subjectsToTeach = new HashSet<Subject>();
```

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

Changes in the main method

This is the code that we already have in the main method. Keep the code and if you don't have add it in your main method.

```
            Tutor newTutor = new Tutor("ABC234", "Natalie Woodward",
                 387787);
            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");


            session.save(student1);
            session.save(student2);
```

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

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

We create two Subject objects too and we add them to our database.

```
            Subject subject1 = new Subject("Math", 3);
            Subject subject2 = new Subject("Science", 6);

            session.save(subject1);
            session.save(subject2);
```

Now add:

```
newTutor.addSubjectsToTeach(subject1);
newTutor.addSubjectsToTeach(subject2);

subject1.addTutorToSubject(newTutor);
subject2.addTutorToSubject(newTutor);
```

And we add a new teacher and insert it in the database:

```
Tutor secondTutor = new Tutor("FJK", "Ben Aflek", 3585895);
session.save(secondTutor);
```

We allocate subject 2 to this teacher:

```
secondTutor.addSubjectsToTeach(subject2);
```

And we do it in the other direction too:

```
subject2.addTutorToSubject(secondTutor);
```

Run the code and check the database

Now in your database shell (ij), drop the tables and run the code.

in ij> run

```
show tables;
```

Now we have three student, tutor, subject tables and an extra table subject_tutor which is a link table.
And if your run

```
select * from subject_tutor;
```

you will get three lines indicating subjectsToTeach_id and tutors_id for this two tutors, one can teach Math and science and the other one can teach Math.

Combining methods

Everything works well. But the problem is that everytime, we must add them in both directions. We can combine these two methods in one of the methods so we call only that method each time:

In Tutor class in *addSubjectsToTeach* method we add this line:

```
subject.getTutors().add(this);
```

We must create the method *getTutors* in *Subject* class:

```
public Set<Tutor> getTutors() {
     return this.tutors;
}
```

In the main method remove these lines:

```
subject1.addTutorToSubject(newTutor);
subject2.addTutorToSubject(newTutor);
```

We do not need them anymore, because *addSubjectsToTeach*does both sides of the relationship now.

Now with the method *addTutorToSubject*, we can either delete it or change this method too in a way that has both directions. And then we can decide which of these methods we want to use. So in *Subject* class, we change the method *addTutorToSubjec*t by adding:

```
tutor.getSubjects().add(this);
```

And in the Tutor class, we add the method getSubjects as following:

```
public Set<Subject> getSubjects() {
	return this.subjectsToTeach;
}
```

It is up to us now which one we want to call in our main method.
Now delete all the tables in ij shell and run the code again and check the result in the database shell.