Hibernate- Mapping-Workshop

Our project and connecting to derby

We will continue with our project from the previous section.

We are using Derby as our database.

In the previous workshop we have created a Student class and with the help of hibernate we could create, delete, insert and update data in the student table.

To start derby in windows:

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

In linux and mac users:

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

Leave this terminal and let the derby serve be up.

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

```
./ij
```

To create a database or connect to an existing database:

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

Now we want to test adding a new field in our Student class and see if it will update the table with a new column.

Adding a new field

Add a field in Student class:

```java
private Integer numberOfCourses;
```

In your main method, delete everything except session and transaction.
Now add a new Student object:

```
Student newStudent = new Student("Ada Svensson");
session.save(newStudent);
```

hbm2ddl.auto in the hibernate.cfg.xml file should be set to ***update***.

Run the code.

In the ij>, run the following:

```
describe student;
```

You should be able to see that a column `numberOfCourses` has been added to the table.

If you run `select * from student;` you see that all the rows (students) in the table have a field numberOfCourses which is null.

Have a different column name in the table

Column name

Sometimes we want to have different names in our table than the ones in our class.

In the ij: first we delete the column numberOfCourses and then we add the column NUM_COURSES:

```
alter table student drop column numberOfCourses;
alter table student add column NUM_COURSES integer;
```

Now we map NUM_COURSES in the table to numberOfCourses in our Student class:

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

You should import jakarta.persistence.Column

Add this line in the constructor: `public Student(String name){`

```
this.numberOfCourses = 10;
```

Now run the code and check the result in the ij to see if you have got 10 numberOfCourses.

Table name

in ij drop the table:

```
drop table student;
```

In Student class do the following:

```
@Entity
@Table(name="TBL_STUDENT")
```

You should import jakarta.persistence.Table

Run the code and check in ij that you have this table

```
show tables;
```

Field access and Property access

So far we were using the field access. Even though we had private properties in our Student class, without accessors and modifiers (get and set methods) hibernate could read directly from them and affect the database.

There is an alternative to mapping properties to the database and it is property access.
In order to use this alternative, we need get and set methods for all the properties in our class.
Note: Check that your get and set methods follow the standard naming in JavaEE (Jakarta EE):

For example:
getName() or getEnrollmentID()
setName() or setEnrollmentID()

@*Id* now is on top of the field id. By this, we say to hibernate that we want to use *field access.*

If we put this annotation above *getId()*, we make a signal to hibernate that we want property access. All the other annotations must also move to above the get methods.
Hibernate looks at the name of get methods, for example *getName()* and creates a field name in the table.

@Transient

It can happen that we have some get methods or fields in our class that we do not want to affect our table (creating a correspondent column in the table)
We can solve this problem by adding:

`@Transient` from jakarta.persistence library
above the method or above the field name.

For example:

```
@Transient
public int getAge(int age) {
   return age;
}
```

Or if you have field access:

```
@Transient
private String getEmail;
```

After this workshop, I recommend changing to field access again.