Workshop

In this workshop:

- We create a project that is using a database (derby) to store the data.
- We use Hibernate to connect to this database.
- We look at how we can start derby and how we can create a database.
- We configure our domain class: Student for hibernate.
- We write a class to do the configurations for hibernate with the help of hibernate.cfg.xml file. The configuration consists of creating a session and transaction.
- We test creating a new student and persist it into the database.
- We test retrieving data from the database.

Our project

We want to create a project which uses a sql database to store the data in a database.
In this project we are using Derby as the database and Hibernate to connect to this database.

Download the code and add the project to your IDE

Download the given HibernateProject.zip . Unzip it and copy it under your IDE workspace.

Instructions to use Derby

Download the Apache Derby database.. We want to use derby as our database.
Navigate to the folder, and then to the bin folder.

Windows users:
There is a file `NetworkServerControl.bat` which will be run to start the derby for Windows users.
Run this in cmd:

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

Mac and linux users:
There is another file `NetworkServerControl` for linux and mac users to start the database.
Run this in terminal:

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

By this command the derby server will be up and running on port 50000.
Leave this terminal or cmd and let the derby server be up and running

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

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

ij is an interactive shell and there we can create databases, tables, etc.
We should connect to a database or if the database does not exist, create one.

Create database

This is the command to create or connect to a database in the ij interactive shell:

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

The part: ‘create=true’ is for the first time that we create the database. Next time we can skip it.

Create table in Derby

Now you are connected to the “hibernate” database. Still in the ij interactive shell, we create a table with the name student:

```
create table student(id integer generated always as identity, name
varchar(200), tutorName varchar (200) );
```

Id is the primary key.

The above command created a table with the name student and three fields.

We insert one student in our table. Just pick a name and id:

```
insert into student (name,tutorName) values('Edward Bourne' , 'Diamond
Cameron');
```

To test it:

```
select * from student;
```

It should return a row with:

```
ID          |NAME                 |tutorName
------------------------------------------------
1           |Edward Bourne        |Diamond Cameron
```

The command to see the tables in the database is:

```
show tables;
```

We can close ij using exit

```
exit;
```

Note: what we did in the above (creating a table and inserting a row, was just to show how we can create a table and insert rows in derby). To begin with Hibernate, we need only to create a database, we don't need to create a table. Hibernate does it for us.

Back to our project in intellij we look at the class Student in our project, its attributes and constructors.

Configure the domain class for Hibernate

In order to be able to use hibernate, we need to do three things:

1- @Entity

in jakarta.persistence library is an annotation(a marker, a label) that should be above the class name. you need to import it to your code:
import jakarta.persistence.Entity;

2- A No-Argument constructor

```
public Student() {}
```

3-An attribute nominated as ID (primary key)

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)  //This line is optional
private int id;
```

Create class HibernateTest to test our hibernate

Now we create a new class in package *se.yrgo.test* with the name: *HibernateTest*

1. We want a main method in this class which has an object of class Student and prints it.

   (You need to add a toString method in your Student class)

```
Student newStudent= new Student("Nick Fame", "Diamond Cameron");
System.out.println(newStudent);
```

Persisting data in the database

Now we want to save the new student in our database.

In order to create a session object, we need to do the following:

Configure Hibernate

We need three things to save the data in our database in hibernate. (persistence)

1. SessionFactory
2. hibernate.cfg.xml file
3. Transactions

First we create the *SessionFactory* only one time for the whole project. In our main method, after creating the student, we add these lines:

```
SessionFactory sf = getSessionFactory();
Session session = sf.openSession();
```

(getSessionFactory is a method that we are going to write)

NOTE: When you write the above lines, you get a compiling error in the IDE, but it is temporary, we are going to fix it soon.

SessionFactory

SessionFactory is a helper class and getSessionFactory() is a helper method which we want to write.

We need this line in our class before the main method:

```
private static SessionFactory sessionFactory = null;
```

And this is the getSessionFactory method: (the standard way to create a sessionFactory)

```
   private static SessionFactory getSessionFactory() {
        if(sessionFactory ==null) {
            Configuration configuration = new Configuration();
            configuration.configure();

            sessionFactory = configuration.buildSessionFactory();
        }
        return sessionFactory;
    }
```

Configuration is in *org.hibernate.cfg* package.

Transactions

Now we add the Transactions:

In the main method after the session add:

```
Transaction tx = session.beginTransaction();
```

And in the end of main method, after saving the student in session:

```
tx.commit();
session.close();
```

To save the newStudent in the database we write:
`session.save(newStudent);`
between the transaction block.
This will be an example of the whole HibernateTest class:

```
package se.yrgo.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import se.yrgo.domain.Student;

public class HibernateTest {
      private static SessionFactory sessionFactory = null;

      public static void main(String[] args) {
            Student newStudent= new Student("Nick Fame", "Diamond
Cameron");
            System.out.println(newStudent);
            SessionFactory sf = getSessionFactory();
            Session session = sf.openSession();

            Transaction tx = session.beginTransaction();

            session.save(newStudent);

            tx.commit();
            session.close();
      }


   private static SessionFactory getSessionFactory() {
        if(sessionFactory ==null) {
            Configuration configuration = new Configuration();
            configuration.configure();

            sessionFactory = configuration.buildSessionFactory();
        }
        return sessionFactory;
     }

}
```

hibernate.cfg.xml

Now go to the configuration xml file: hibernate.cfg.xml
Under the session-factory, there are two properties:

```
<property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
<property name="connection.url">jdbc:derby://localhost:50000/hibernate</property>
```

which configures our database.

`<property name="show_sql">true</property>` //true for development, false for production.

`<property name="hbm2ddl.auto">create</property>`

will create the tables automatically from the java classes with the necessary annotations.

`<mapping class="se.yrgo.domain.Student"/>`
By adding this property, hibernate does not need to check every class that is loaded in the memory.

This file is under src/main/resources
We have the following in the pom file so that maven can find it.

```
<resource>
   <directory>src/main/resources</directory>
</resource>
```

Running the code

Run the code. (derby server should be running)

Now in the ij interactive shell write ***show tables;***

and then `select * from student;`
Now you should have one student in the table.

You can get the id of the student by this line:

```
System.out.println("The student has an id  of: " + newStudent.getId());
```

You should add *getId()* in your Student class too.

Retrieve an object by using get method

To find an object or select a row in the table, we can use the ‘get’ method in Session class:

```
Student myStudent = (Student)session.get(Student.class, 1);
```

‘1’ here is the id of the student that we want to retrieve. You should check your database and provide an “id” that you have in your database.
(Student) is a type cast.

```
System.out.println(myStudent + " is the student");
```

With this line we get only one student, not all the students. If we want to get all the students, we write a query.

Create more students

Continue with creating two more students in the main and persist them. Check the database to see if they are in the table.

Then find the students by their id and print them.