Hibernate

in action

Configuring Hibernate

What do we need to implement hibernate and create a simple project:

- We need to
  - have a domain class and the same data in a table in the database.
  - choose a database
  - configure the domain class for the hibernate
  - create a class to configure the hibernate in order to make the connection to the database.
  - configure the hibernate.xml file.
- And we will create a row of data , retrieve all the data, update a row and remove a row in our database via this class.

Configuring Hibernate Example

To create an example hibernate project, we will proceed with the following:

- have a Student class and a student table in the database.
- use Apache Derby as the database
  - Download Derby
  - Create database
  - Create table student
- configure Student class for the hibernate
- create a class to configure the hibernate in order to make the connection to the database.
- configure the hibernate xml file.
- create a student, retrieve all the students, update a student and remove a student via this class.

Apache Derby

Apache Derby, an Apache DB subproject, is an open source relational database implemented entirely in Java and available under the Apache License, Version 2.0.

Some key advantages include:

- Derby has a small footprint -- about 3.5 megabytes for the base engine and embedded JDBC driver.
- Derby is based on the Java, JDBC, and SQL standards.
- Derby is easy to install and use.

Connect to Derby

First we need to connect to our database. There are two files in the bin folder of derby which are called NetworkServerControl.bat which will be used for windows system and NetworkServerControl.sh which will be used for linux and mac system:

C:\>cd db-derby-10.12.1.1-bin/bin

Windows:

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

Linux:

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

Create a database in Derby

Then we need to create a database. To do that:
Open interactive shell ij:

C:\>cd db-derby-10.12.1.1-bin/bin

Run ij:

In Windows(cmd): Run ij.bat

In Linux and Mac: Run ./ij

Create a database:

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

Hibernate is the name of the database that we create. You can choose something else.

Configure Domain class for Hibernate

In order to be able to use hibernate in our code, we need to do three things.

The class that we want to map to our table needs these three things to be included (in our case this class would be the Student class):

- 1- @Entity

  in jakartapersistence library is an annotation(a marker, a label) that should be above the class name.

- 2- A No-Argument constructor

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

Configure Domain class for Hibernate

- 3-An attribute (property) nominated as ID (primary key)

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

Configure Hibernate in our project

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

1- SessionFactory, 2-Transaction and 3- hibernate.cfg.xml file.

SessionFactory:

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

In the main method:

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

//getSessionFactory() is a method that we write ourselves:

Configure Hibernate in our project

//getSessionFactory() is a method that we write ourselves:

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

Transaction

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();
```

Example code

```
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();
     }
```

Configure Hibernate in our project

hibernate.cfg.xml file:

- hibernate.cfg.xml file contains database related configurations and session related configurations.
- Database configuration includes jdbc connection url, DB user credentials, driver class and hibernate dialect.
- It is placed under src/main/resource folder.

Configure Hibernate in our project

DBMS configuration:

```
<property
name="connection.driver_class">org.apache.derby.jdbc.ClientDriver
</property>

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

Configure Hibernate in our project

Sql configurations:

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

Domain configuration:

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

Create data and persist in the table

We create a session with the help of SessionFactory

We call the ***save*** method on the session. This will persist the data in the database:

session.save(newStudent);

This method (save) works like *insert* in sql language.

Create data and persist in the table

```
public static void main(String[] args) {

    Student newStudent= new Student("Nick Fame", "Diamond Cameron");

    SessionFactory sf = getSessionFactory();

    Session session = sf.openSession();

    Transaction tx = session.beginTransaction();

    session.save(newStudent);

    tx.commit();

    session.close();

}
```

Retrieve data from the table

We call the method ***get*** on the session:

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

This will return the student by its id. It is like

```
select * from student  where id=?
```

So we are retrieving a student with Id number 1.

Update data

To update data, we can retrieve the data from the database first and when we have saved it in an object, we can use the java set method to update the data.

For example in the below, we get the student with id number 3 from the database and then we save it as a student object.

Then we call the setTutor method to update the tutor in this row.

```
Student student = (Student) session.get(Student.class, 3);

student.setTutor("David Graveyard");
```

Delete data

Again we will retrieve the specific row (item) that we want to delete from the database by using the *get* method.

Then we call the ***delete*** method on the session which takes the student object as the argument and deletes it from the database.

```
Student myStudent = (Student) session.get(Student.class, 4);

session.delete(myStudent);
```