LS/EECS Building Ecommerce Systems

Lab 5: Student Information System R1.0

Objectives

-understand model view control programming model
-learn to use JDBC programming model
-develop full stack MVC application
-create a Data access layer (using Data Access Object pattern)

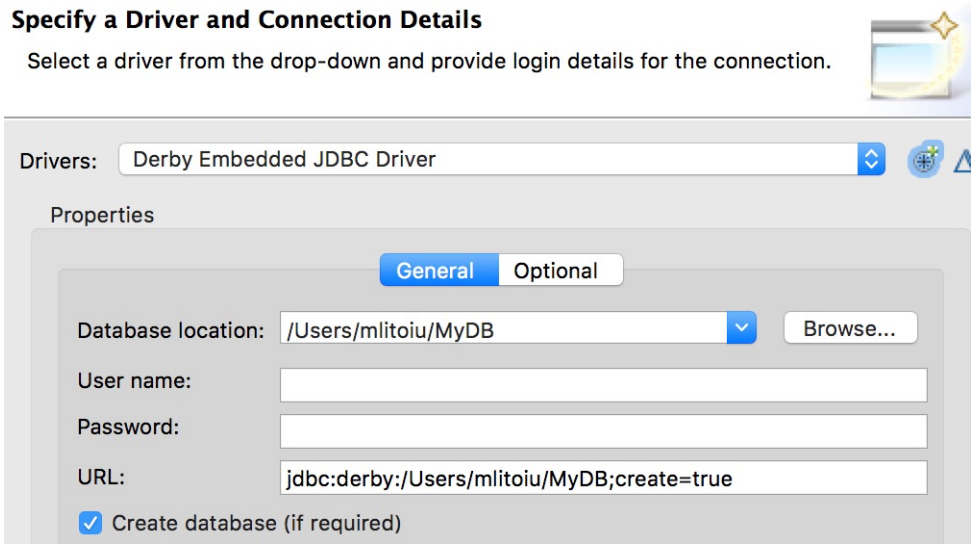
The Project

- Create a new project named *SIS-v1* (Student Information System, release #1).
- The project enables its client to generate reports that returns information from a student records database.
- The client can filter the students included in the report based on their names and credits.

Requirement 1. Installing and Exploring the Database as an *DB Administrator*

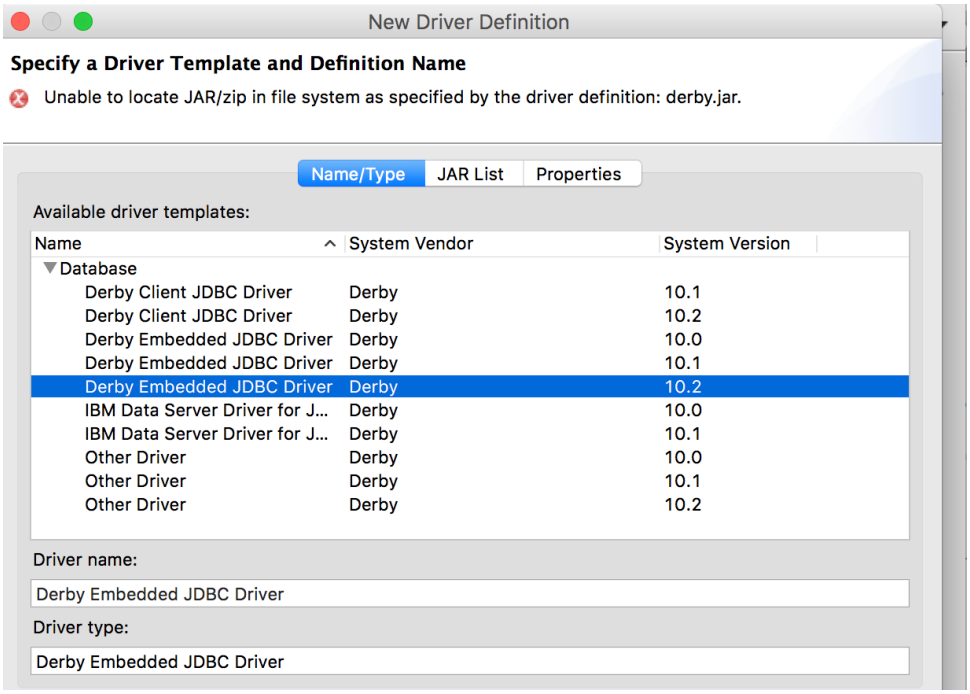
A. Install Apache Derby Embedded Driver (embedded= the database engine runs within the same VM where JDBC runs). Use these instructions:

1. If you have not done it in Lab 1,
    a. Download and unzip Apache derby version 10.14.2.0 (lib version is enough) http://db.apache.org/derby/derby_downloads.html
    b. Unzip it Eclipse directory.
2. Create a Dynamic Web project, SIS-v1
3. In Eclipse, switch into Database Development perspective
4. Select Database Connection->New Connection.
5. From the list of connections, select Derby....and then Next
    5.1. you should see something like this....



5.2 if you do not see "Derby Embedded JDBC Driver" at the top of the wizard, then click on the + sign, top right corner of the wizard.

5.3 in the next wizard page you should be able to add the driver: selects its name and add the "derby.jar" file in the "Jar List" tab ("derby.jar" is in the derby folder, see step 2 above).



6. Select the location of the database in workspace\SIS-1. Chose the name of the database "SIS_DB" (Derby will create a local database). No user id and pswd are needed for this project.

7. in the wizard showed in step 5.1, there is a button "Test Connection." Press that and if everything is ok then the driver is installed.

B. Use Eclipse "Database Development" to create a connection, then "Open SQL Scrapbook" and create two tables STUDENTS and ENROLLMENT

1. *You can use sql statements available in "projectB.sql" to create and populate the tables.*

2. Explore the tables (if no user id is used, Derby stores the tables in APP.TABLES. You can use "Data->Edit" on the table menu to visually edit the tables.

Now, you can issue any valid SQL statement in the scrapbook and it will be executed. Here are a few examples:

```
select * from   STUDENTS;
select SID, CREDIT_TAKEN from STUDENTS where CREDIT_TAKEN >= 100;
select SURNAME, CREDIT_GRADUATE from STUDENTS where CREDIT_TAKEN >
90 and SURNAME like '%Bradley%';
```

```
Use the scrapbook to practice SQL to practice these:
INSERT a student or an enrolment row
DELETE a student or an enrolment row
UPDATE a student or an enrolment row
```

Important Notes:

- **Derby Driver that you use only allow 1 connection at a time. Close the connection you opened above. Select the connection in the editor and close it**

- **Many times, your software crashes before your connection is closed. To fix that, stop and restart the server, that will clear the connection.**

Requirement 2. Use the database as a *Java Programmer.*

a. Set the JDBC, pool connection

Configure your SIS-1 project/application to use a JDBC pool.

How to do it:

1. Add "derby.jar" file to WEB-INF/lib directory. This is needed by Tomcat, when you "Run on Server," Note that this will enable your application to access the database
2. Under META-INF, create "context.xml" file. You can get information about the syntax and semantic of its content from http://tomcat.apache.org/tomcat-8.0-

doc/index.html. For now, it is important to know that this file contains information about the data source and the jdb driver to be used at runtime. Below is an example. Note **Resource** element and its attributes **driverClassName** and **url**. These attributes will depend on the jdbc driver you use and the location of your database. You have to change the url so it points to YOUR Database.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context privileged="true" reloadable="true">
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <Manager pathname="" />
    <Resource name="jdbc/EECS"  factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
            type="javax.sql.DataSource"
            driverClassName="org.apache.derby.jdbc.EmbeddedDriver"
            url="jdbc:derby:/Users/marin/Documents/workspace/SIS_DB"/>
    <ResourceLink global="jdbc/EECS" name="jdbc/EECS" type="javax.sql.DataSource" />

</Context>
```

b. Write your first jdbc program to test your configuration file and your runtime server settings

Create a *dao.StudentDAO.java* class, with a method that will connect to the database SIS-DB, retrieve all STUDENTS and prints each row to the server console.
To understand each line below, check the course notes and examples…

```
import java.sql.*;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public class StudentDAO {
  public void readAndPrintTableToConsole() throws SQLException {
     try {
              DataSource ds = (DataSource) (new InitialContext()).lookup("java:/comp/env/jdbc/EECS");
              Connection con = ds.getConnection();
              Statement stmt = con.createStatement();

              ResultSet rs = stmt.executeQuery("SELECT * FROM STUDENTS");
              while(rs.next()){
                        String em= rs.getString("SID");
                        String fname = rs.getString("SURNAME");
                        System.out.println("\t" + em+ ",\t" + fname+ "\t ");
             }//end while loop

          con.close();
   } catch (NamingException e) {
       e.printStackTrace();
   }
}
```

To test the above code on the web server (Tomcat), you have to invoke it from a servlet.

- Create a servlet *ctrl.Sis.java* and in the *doGet* method, invoke the main method above, such as:

```
try {
        StudentDAO sd=new StudentDAO();
```

```
            sd.readAndPrintTableToConsole();
        }catch (Exception e) {
            e.printStackTrace();
        }
```

- Run the servlet and you should see the student list on printed on the console.

If you Your environment and your first JDBC program work correctly, you can move to the next requirement.

Requirement 3. SIS Web Application: Model View Controller

Develop a reporting application that

- enables its client to generate reports that return information from a student records database.
- The application has to follow the MVC architecture

The application has one View that shows a query form and the results of the query, rendered as a table, as below (The exact rendering is not required):

Student Information System

Home About

SIS-v1

Name Prefix: Rodrig

Minimum Credit Taken: 20

ReportAjax

Student Name	Credits taken	Credits toGraduate
Philip, Rodriguez	81	90
Irene, Rodriguez	84	90

How to implement it. Note that you can implement the views using JSP or simply using Java Script. In the following we use Java Script, you might want to review Lab 4 and Lecture 4.

3.1 Create a form SisApp.html, choose the HTML 5 template in Eclipse. This is one View component of our webapp. It presents a form (with POST submission) containing two input fields (with labels *Surname Prefix* and *Minimum Credit taken*). The form should have an appropriate legend and one submit button captioned *Report.*

- Note that when the Report button is clicked then the result should appear below the form
- Note also that the name prefix applies to the **SURNAME** only and the search for it must be case sensitive.

Start with a simple form, with no AJAX or JS validation, test it with no inputs, just trigger the test code you created in II.b

3.2 Create a bean.StudentBean.java class

This class in the `bean` package is just a simple data structure that holds information about one student and it maps the table STUDENT. It has four private attributes (sid, name, credit_taken, and credit_graduate), a constructor that sets them and four accessor (getter) / mutators (setter) methods. *Hint: You can just declare the attributes and ask Eclipse to generate the rest using its Source menu.*

3.3 Create a bean.EnrollmentBean.java

This class in the `bean` package is just a simple data structure that holds information about one course as in the table ENROLMENT. It has three private attributes (cid, students, and credit), a constructor that sets them and four accessor (getter) / mutators (setter) methods. *Hint: You can just declare the attributes and ask Eclipse to generate the rest using its Source menu. Students is the list of student IDs who are currently enrolled in the course.*

3.4 Modify dao.StudentDAO.java

This class in the `model` package is responsible for communicating with the DBMS. It is instantiated from the model and it contains one attribute, the data source that can grab connections from the pool. See the course notes to understand how a pooled connection can be used. Check also the test examples above. The essentials are listed below:

a. The constructor has to lookup for the data source. Note the name **jdbc/EECS** (it is from context.xml)

```
public StudentDAO() throws ClassNotFoundException{
    try {
     ds = (DataSource) (new InitialContext()).lookup("java:/comp/env/jdbc/EECS");
    } catch (NamingException e) {
                    e.printStackTrace();
    }
}
```

b. One method in this class must have the following header and sample body:

```
public Map<String, StudentBean> retrieve(String namePrefix, int credit_taken) throws
SQLException{
        String query = "select * from students where surname like '%" + namePrefix +"%'
        and credit_taken >= " + credit_taken;
         Map<String, StudentBean> rv = new HashMap<String, StudentBean>();
        Connection con = this.ds.getConnection();
        PreparedStatement p = con.prepareStatement(query);
        ResultSet r = p.executeQuery();
        while (r.next()){
          String name = r.getString("GIVENNAME") + ", " + r.getString("SURNAME");
          String cseID = r.getString("SID");
          int credit_taken = r.getInt("CREDIT_TAKEN");
          int credit_graduate = r.getInt("CREDIT_GRADUATE");
          rv.put(cseID, new StudentBean(cseID, name, credit_taken, credit_graduate));
        }
        r.close();
        p.close();
        con.close();
        return rv;
}
```

To test the method, you can call it from the "Sis" servlet, with some arbitrary parameters. Note the method returns a Map, you need to convert to String and either print it out to console or append it to the servlet response.

3.5 dao.EnrollmentDAO.java

Similar to StudentDAO, this class is responsible to retrieve data from Enrollment table.

3.6 model.SisModel.java

This class in the `model` package is the main model class. It has attributes that hold an instance of `StudentDAO and EnrollmentDAO` (initialize it in the class constructor). In addition, it has the methods:

```
public Map<String, StudentBean> retriveStudent(String namePrefix, String
credit_taken) throws Exception

public Map<String, EnrollmentBean> retriveEnrollment() throws Exception
```

that perform validation (possible by invoking private methods) and then invoke the appropriate methods in dao.`StudentDAO/dao.EnrollmentDAO`.

Note that the validation logic must also sanitize the input to prevent SQL injection.

3.7 Modify ctrl.Sis.java

You now can comment out the test code from II.b, retrieve the parameters from rquest and distinguish between a first call and a call from the form. You can use a button parameter, "reportAjax" to make the distinction. Its init method must instantiate the model and store it in the servlet context and handle any exception by proper re-throwing and logging. Its doGet and doPost methods are very similar to (if not simpler than) those of OsapCalc. Note that no session scope storage is needed for this.

3.8 Render the table in the html page.

Now you can make your application to display the results as a table, below the form.

First, you need to tell the button to call your function reportAjax()

```
<button type="submit" value="ReportAjax" id="report" name="reportAjax"
onclick="reportAjax('/SIS-1/Sis?reportAjax=true')">ReportAjax</button>
```

Write the function reportAjax() to compose a GET method, by adding the form input values to the query sting and send it to the servlet. The code is similar to Lab 4. Then, you get the responseText and insert it to the end of the table, again, similar to Lab 4.

To render the table, you can return an html <table> element from the servlet, with the header and rows populated. Here is an example of what you can do at the server side:

```
public class View {
  public String studentAsHTML(SisModel model, String namePrefix, String credit_taken) {
            String html ="";
            Map<String, StudentBean> hmap=null;
            try {
                    hmap=model.retriveStudent(namePrefix, credit_taken);
            }catch (Exception ex){
          System.out.print(ex);
      }

 html="<table border=\"1\"> <thead> <td>Student Name </td> <td>Credits taken </td> <td>Credits
toGraduate </td> </thead>";

      Iterator iterator = hmap.entrySet().iterator();
      while (iterator.hasNext()) {
       Map.Entry me = (Map.Entry) iterator.next();
       StudentBean sb=(StudentBean)me.getValue();
       html+="<tr>"+
         "<td>"+sb.getName()+"</td>"+
           "<td>"+sb.getCredit_taken()+"</td>"+
         "<td>"+sb.getCredit_graduate()+"</td>"+
           "</tr>";
       }
            html+="</thead></table>";
      return html;
}
}
```