

ECE356 – Database Systems

Lab 1: Accessing Databases from Web Applications



Mohammad Hamdaqa
Lab Instructor
Winter 2015

Learning objectives

- ✓ Understand the Model-View-Controller architecture commonly used in java web applications development
- ✓ Learn how to create a web application that allows the user to input data. The project will be implemented using JSPs, Java Servlets, and JavaBeans
- ✓ Learn how to connect to a MySQL database, using the MySQL JDBC driver, from a web application

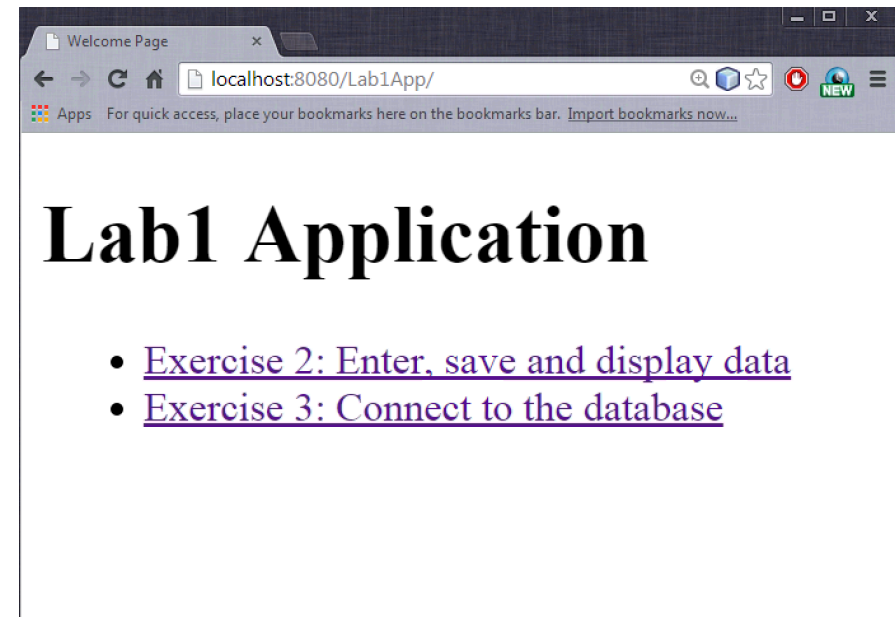
Prerequisite

What you need to know

- Basic knowledge of the following
 - ✓ Java programming language (classes, interfaces, etc.)
 - ✓ HTML language – for static web content

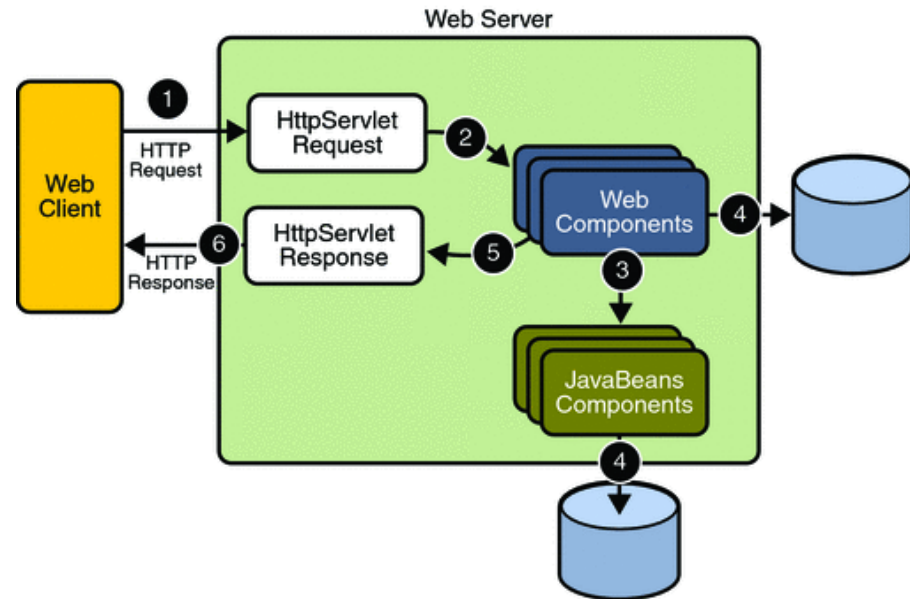
Deliverables

- At the end of this Lab you are expected to demonstrate the application that you built:
 - The web application should look like the screen shot provided on the right hand side
 - Both links should be functional
 - By clicking on the first link (Exercise 2) you should be able to enter data and display it
 - By clicking on the second link (Exercise 3) you should be able to test the database connection



Web Applications

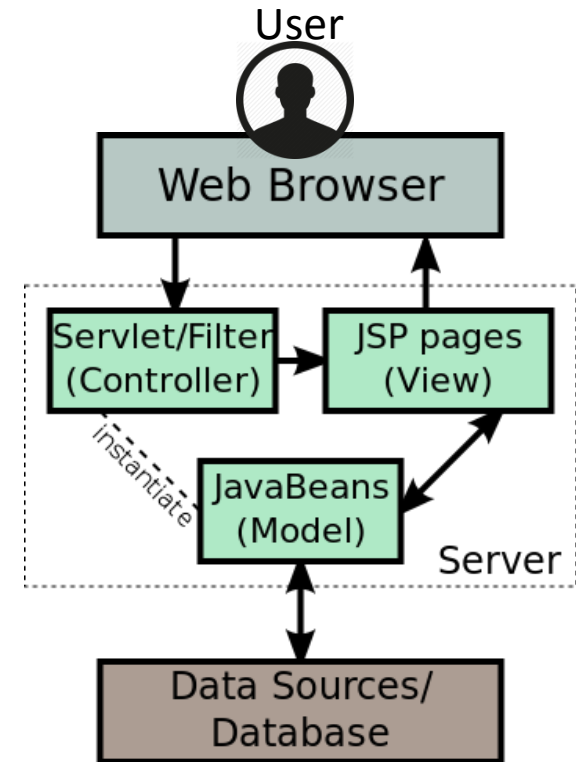
- A web application is a dynamic extension of a **web** or **application server**
- A web application generates interactive web pages containing various types of markup language (HTML, XML, and so on) and dynamic content in response to requests.
- In the Java 2 platform, **web components** provide the dynamic extension capabilities for a **web server**.
- Web components are either **Java servlets**, **JSP pages**, or web service endpoints.
- Web Components can share information in several ways
 - They can use private helper objects (for example, JavaBeans components)
 - They can share objects that are attributes of a public scope
 - They can use a database
 - They can invoke other web resources.



JSP Model 2

Web Application Architecture

- JSP Model 2 is a realization of the Model-View-Controller (MVC) architectural style for building java web applications
- Model 2 is a hybrid approach for serving dynamic content
 - combines the use of servlets and JSP
 - uses JSP to generate the presentation
 - uses servlets to perform process-intensive tasks
- The servlet acts as the controller and is in charge of:
 - request processing
 - creation of any beans used by the JSP
 - deciding, depending on the user's actions, which JSP page to forward the request to
- There is no processing logic within the JSP page itself; it is simply responsible for retrieving any objects or beans that may have been previously created by the servlet



Source:

<http://www.javaworld.com/article/2076557/java-web-development/understanding-javascript-model-2-architecture.html>

Web Application Architecture

- **Web Browser:** Handles the users request and retrieves the information from the server. Browsers can render HTML pages but can not deal with dynamic content.
- **Web Server:** A software application that accepts and supervises the HTTP requests (e.g., Apache Tomcat, Glassfish, etc.)¹
 - Apache Tomcat is a software implementation of the Java Servlet and JavaServer Pages.
 - GlassFish is the reference implementation of Java EE and as such supports Enterprise Java Beans, JPA, JavaServer Faces, JMS, RMI, JavaServer Pages, Servlets, etc.
 - This tutorial uses JavaServer Pages, Servlets, and JavaBeans
 - Java Servlet: allow dynamic content in web pages
 - JSP: high-level abstractions of Java servlets
 - JavaBean: reusable software components (classes) for Java, encapsulating related objects into a single object.
- **Data sources** – Database, ie. MySQL

¹ Both Tomcat and GlassFish are application servers. An application server contains a webserver.
See this link for detail <http://www.tomcatexpert.com/blog/2010/06/09/tomcat-application-server>

Java Servlet

- What Is a Servlet?

“A servlet is a **Java programming language class** used to extend the capabilities of **servers** that host applications accessed by means of a **request-response programming model**. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

The javax.servlet and javax.servlet.http packages provide interfaces and **classes for writing servlets**. All servlets must implement the Servlet interface, which defines **lifecycle methods**. When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as **doGet** and **doPost**, for handling HTTP-specific services.”

<http://docs.oracle.com/javaee/6/tutorial/doc/bnafe.html>

Web Components

Sharing objects as attributes of a public scope

- As other collaborating web components in java EE servlets share information by means of objects that are maintained as attributes of four scope objects.

Scope Object	Class	Accessible From
Web context	<code>javax.servlet.ServletContext</code>	Web components within a web context.
Session	<code>javax.servlet.http.HttpSession</code>	Web components handling a request that belongs to the session.
Request	Subtype of <code>javax.servlet.ServletRequest</code>	Web components handling the request.
Page	<code>javax.servlet.jsp.JspContext</code>	The JSP page that creates the object.

<http://docs.oracle.com/javaee/5/tutorial/doc/bnafo.html>

Java Servlet Example

Gets parameters from request object

```
public class UserDataServlet extends HttpServlet
{
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        try
        {
            UserData ud = new UserData();
            ud.userName = request.getParameter("username");
            ud.favColour = request.getParameter("usercolour");
            HttpSession session = request.getSession(true);
            session.setAttribute("userData",ud);
            // displays following page
            getServletContext().getRequestDispatcher("/user_data_thanks.jsp").forward(request,response);
        } finally {
            out.close();
        }
    }
}
```

The ud object is used to save the information received in request for the duration of the “session”

Keep an eye

Keep an eye

Java Servlet Example

Java Interface HttpSession

- public interface **HttpSession**
 - Provides a way to **identify a user** across more than one page request or visit to a Web site and to store information about that user.
 - ie. Servlets can save objects to sessions, allowing user information to persist across multiple user connections.

Java Interface HttpServletRequest

- public interface **HttpServletRequest** extends [ServletRequest](#)
 - Extends the [ServletRequest](#) interface to provide request information for HTTP servlets
 - A **ServletRequest** object provides data including **parameter name and values, attributes**, and an input stream.

Other Java classes of interest

- ServletContext – methods for servlet to communicate with web container
 - <http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html>
- PageContext
 - <http://docs.oracle.com/javaee/1.4/api/javax/servlet/jsp/PageContext.html>
- HttpServletResponse
 - <http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html>

JSP Example

Implementation of user_data_thanks.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
```

```
<html>
```

```
<head>
```

```
<title>Form 1</title>
```

```
</head>
```

```
<body>
```

```
<h2>User data remains in session.</h2>
```

```
<jsp:useBean id="userData" class="ece356.UserData" scope="session"/>
```

```
Name: <%= userData.getUserName() %><br/>
```

```
Colour: <%= userData.getFavColour() %><br/>
```

```
<p>
```

```
<a href="index.jsp">Start over</a>
```

```
</body>
```

```
</html>
```

Notice that userData is the same object, which holds the values of the object (ud) that has been created in the servlet

JavaBean in JSP

```
<jsp:useBean id="userData" class="ece356.UserData" scope="session"/>
```

The JavaBean named **userData**, an object of type **ece356.UserData**, is defined and assigned the “**session**” scope, that is, the value stored in this object will be available to other web components in this web session.

JavaBean Conventions

- A JavaBean is a Java class following certain conventions:
 - **method naming, construction, and behaviour.**
 - These conventions make it possible to have tools that can use, reuse, replace, and connect JavaBeans.
- The required conventions are as follows:
 - The class must have a public [default constructor](#) (with no arguments). This allows easy instantiation within editing and activation frameworks.
 - The class [properties](#) must be accessible using **get**, **set**, **is** (used for boolean properties instead of get), and other methods (so-called [accessor methods](#) and [mutator methods](#)) according to a standard [naming convention](#). This allows easy automated inspection and updating of bean state within frameworks, many of which include custom editors for various types of properties. Setters can have one or more than one argument.
 - The class should be [serializable](#). [This allows applications and frameworks to reliably save, store, and restore the bean's state in a manner independent of the [VM](#) and of the platform.]

http://en.wikipedia.org/wiki/JavaServer_Pages

JavaBean Example

Implementation of the UserData Class

```
package ece356;

public class UserData {

    String userName;
    String favColour;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String value) {
        userName = value;
    }

    public void setFavColour(String value) {
        favColour = value;
    }

    public String getFavColour() {
        return favColour;
    }
}
```


Hands-on exercises

Starting code for the exercises can be found under this url:

<https://ece.uwaterloo.ca/~ece356/lab1>

Lab 1 – Three Exercises

Exercise 1:

- Install NetBeans (**Note: You can skip this step if you are using the computers in the lab**)
- Create your first “Hello World” web project

Exercise 2:

- Modify your web app, adding a page where the user can enter data. The application will then save this data and display it in subsequent pages.

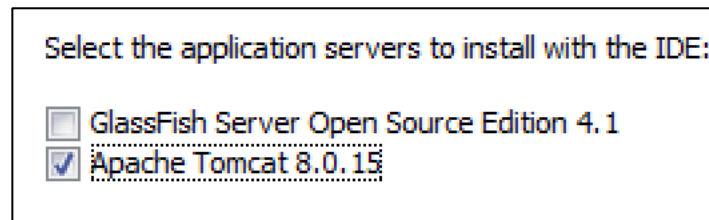
Exercise 3:

- Connect to the MySQL database from your web application.

Exercise 1:

NetBeans Installation

- Install the Java Development Kit
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Install Netbeans 8.0.2
 - <http://netbeans.org/downloads/>
 - Download the “Java EE option” from the available “NetBeans IDE Download Bundles”.
 - Select you preferred application server (e.g., “Apache Tomcat”) for your web server during the installation. You can select both of them, and decide which one to use for your application later on.

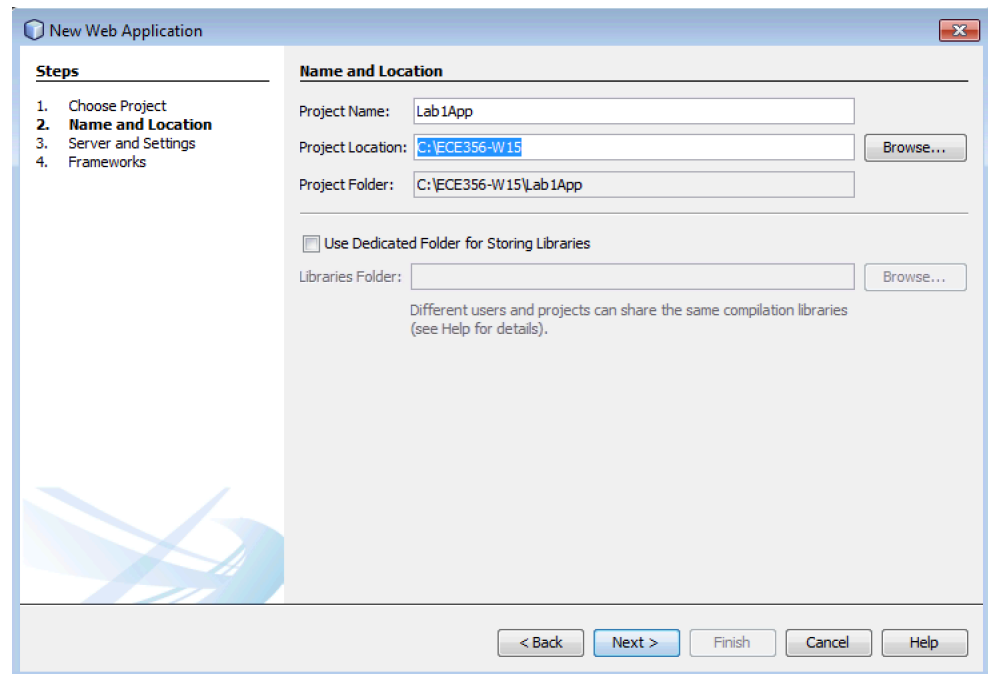
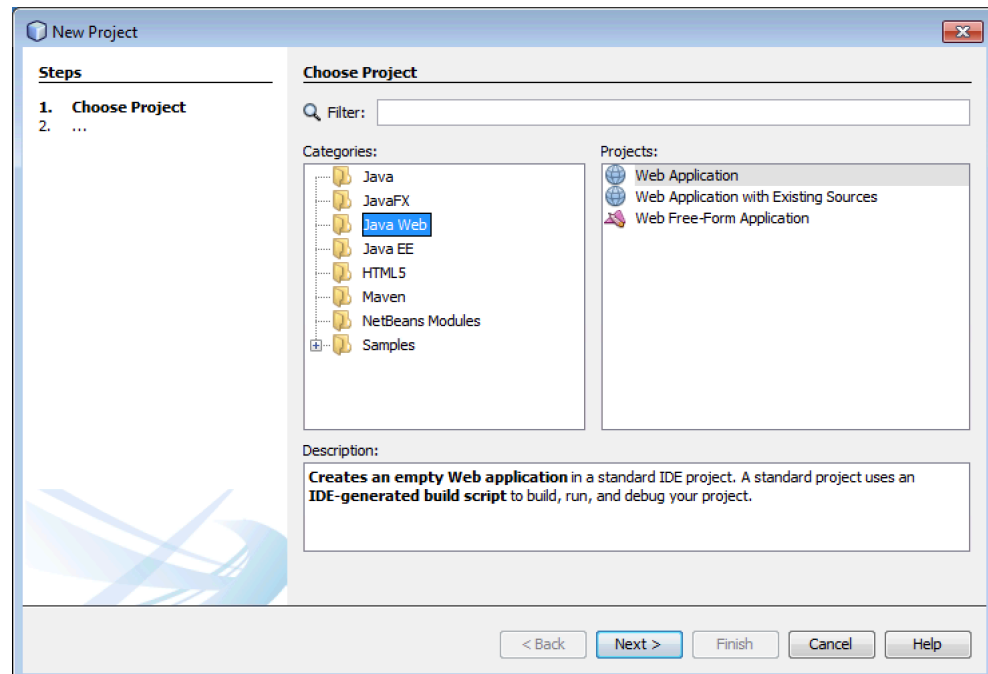


- Select the place where your JDK is installed
 - e.g., C:\Program Files\Java\jdk1.8.0_25

Exercise 1:

Create A “Hello World” Web Project

- Start NetBeans
- Create a Web Project
 - File/New Project
- **Select Java Web**
- Click Next
- **Choose the name and the location of the project**
 - e.g., Name: Lab1App
- Click Next
- Select a server (e.g., Apache Tomcat)
- Click Finish

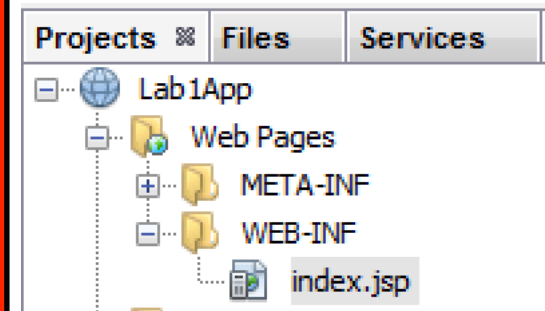


Exercise 1:

Create A “Hello World” Web Project

- Notice that Netbeans has generated a file called index.jsp. This is the default start page for your applicationx that contains HTML code to display “Hello World” when the page is invoked from the server.
- A JSP is an HTML page containing a reference to Java servlets, or, java server side applets.
- If you are new to JSP technology and html you will need to consult online tutorials to learn about this technology. You could use the sources provided at the end of this tutorial.

Note. Depending on the server and Java EE version that you specified when you created the project, the IDE might generate index.html as the default welcome page for the web project. You can use the New File wizard to generate an index.jsp file to use as the welcome page, in which case you should delete the index.html file.



Exercise 1:

The Generated index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

Exercise 1:

Running Your Application

- Make sure the server is running
 - Go to the ``services`` tab
 - Click on servers
 - Right click on the server you selected for your project
 - Click run
- Run your first web project
 - Go back to the ``projects`` tab
 - Right click on your project
 - Click on run
- The project will compile and a webpage will be open that displays “Hello World!”
- Bug fix for Tomcat 8 on windows,
 - In case the Tomcat Server did not start and you got this error
 - '127.0.0.1' is not recognized as an internal or external command
 - Bug fix
 - <http://stackoverflow.com/questions/26485487/error-starting-tomcat-from-netbeans-127-0-0-1-is-not-recognized-as-an-inter>

Exercise 2:

Enter, save and display data

- Objectives:
 - Modify this web application so that it will allow the user to enter some data in another JSP. The application will then display this data and save it.
- You will need to:
 - **Step1:** add several JSPs to enter and display the data entered.
 - user_data_form.jsp -> will be used to enter the data
 - user_data_thanks.jsp and user_data_thanks2.jsp -> will be used by the to display the values entered by the user
 - **Step2:** Add a JavaBean class to store the data entered by the user.
 - **Step3:** Add a servlet to process, save the data entered, and display this data in subsequent JSPs.

Exercise 2:

Step1: adding JSP /enter data

- Add a JSP called `user_data_form.jsp` under the folder “Web Pages”
- Add html code to create a form such as the one shown in the next slide.
- This jsp allows the user to enter some data. The data is then passed to a servlet (UserDataServlet) for additional processing. The parameter ‘action’ in the html form command indicates that UserDataServlet is the servlet to be executed when the ‘submit’ button is pressed:

```
<form method="post" action="UserDataServlet">
```

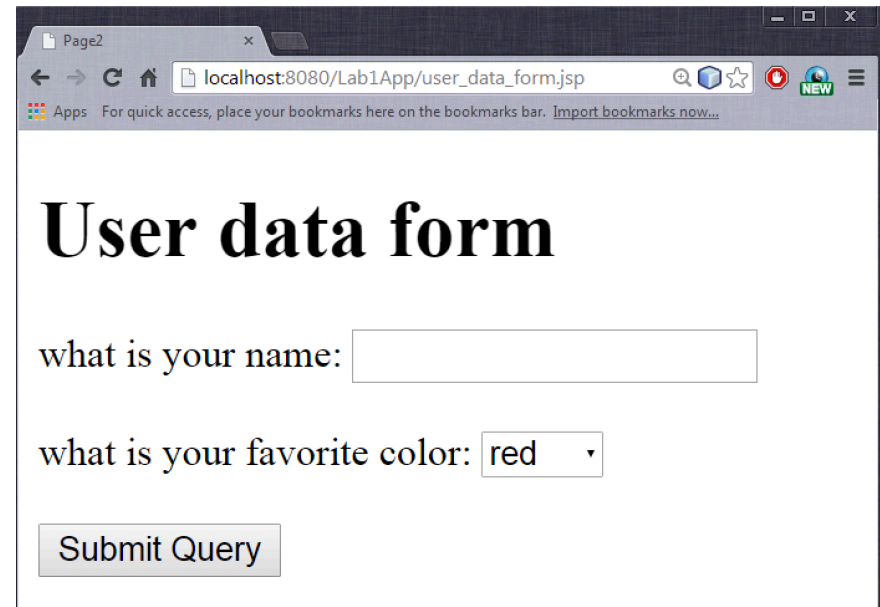
- You can use the Palette (Ctrl-Shift-8) to design the form
 - Drag a form component and drop it after the header

Note: You will create the UserDataServlet in the next step of this lab.

Exercise 2:

Step1: adding JSP / enter data

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html;
  charset=UTF-8">
  <title>Page2</title>
</head>
<body>
  <h1>User data form</h1>
  <form action="UserDataServlet" method="POST">
    what is your name: <input type="text"
    name="username" value="" /><br>
    what is your favorite color: <select name="usercolour">
      <option>red</option>
      <option>green</option>
      <option>blue</option>
    </select><br>
    <input type="submit" value="Submit Query" />
  </form>
</body>
</html>
```



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/Lab1App/user_data_form.jsp'. The page content includes a heading 'User data form', a text input field for 'what is your name:', a dropdown menu for 'what is your favorite color:' with 'red' selected, and a 'Submit Query' button.

The HTML code needed and the expected outcome

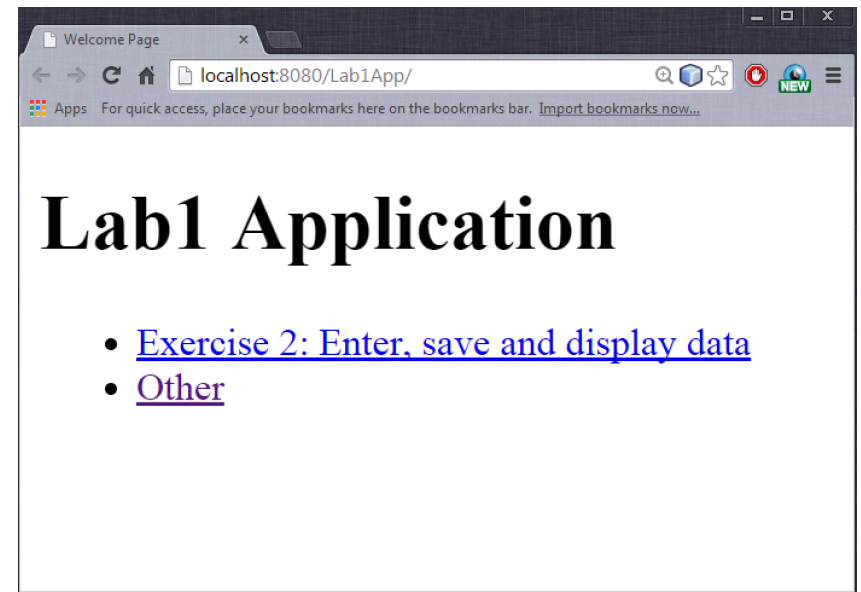
Exercise 2:

Step1: adding JSP / Modify index.jsp

- Modify index.jsp to allow the user to access the page that you have just created. After your changes index.jsp must contain code to display the page shown in the next slide.

Suggestion: practice using HTML List , and HTML Links <a> to display user_data_form.jsp

- On the side is an example of what your index.jsp could look like



index.jsp

Exercise 2:

Step1: adding JSP / to display the entered values

The servlet uses JSPs to display the values entered by the user. These JSPs are provided in the start code, named [user_data_thanks.jsp](#) and [user_data_thanks2.jsp](#).

Download these two JSPs and add them to your project under the folder named “Web Pages”.

Exercise 2:

Step2: JavaBean class used by Servlet

UserData.java

- The servlet ***UserDataServlet.java*** makes use of a JavaBean class .
- A JavaBean class is a reusable software component that encapsulates associated properties and methods, allowing controlled access to these properties.
- The JavaBean class that you will be using, called [***UserData.java***](#).

Download [***UserData.java***](#) and add it to the your ece356 package.

Exercise 2:

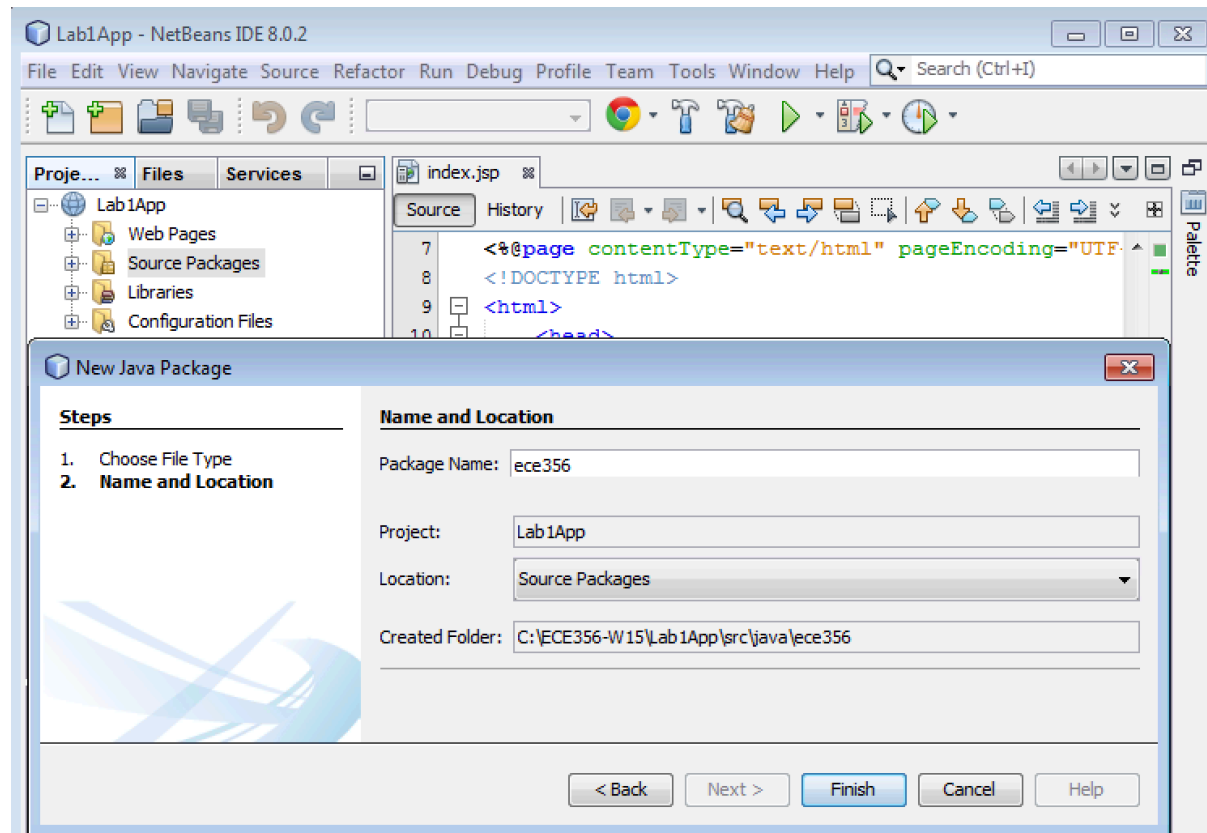
Step3: Adding a Servlet

- You will now add the servlet `UserDataServlet.java` which defines the action performed when the user clicks on the “submit” button of your form.

Exercise 2: Adding a Servlet

Creating a Package for your Servlets

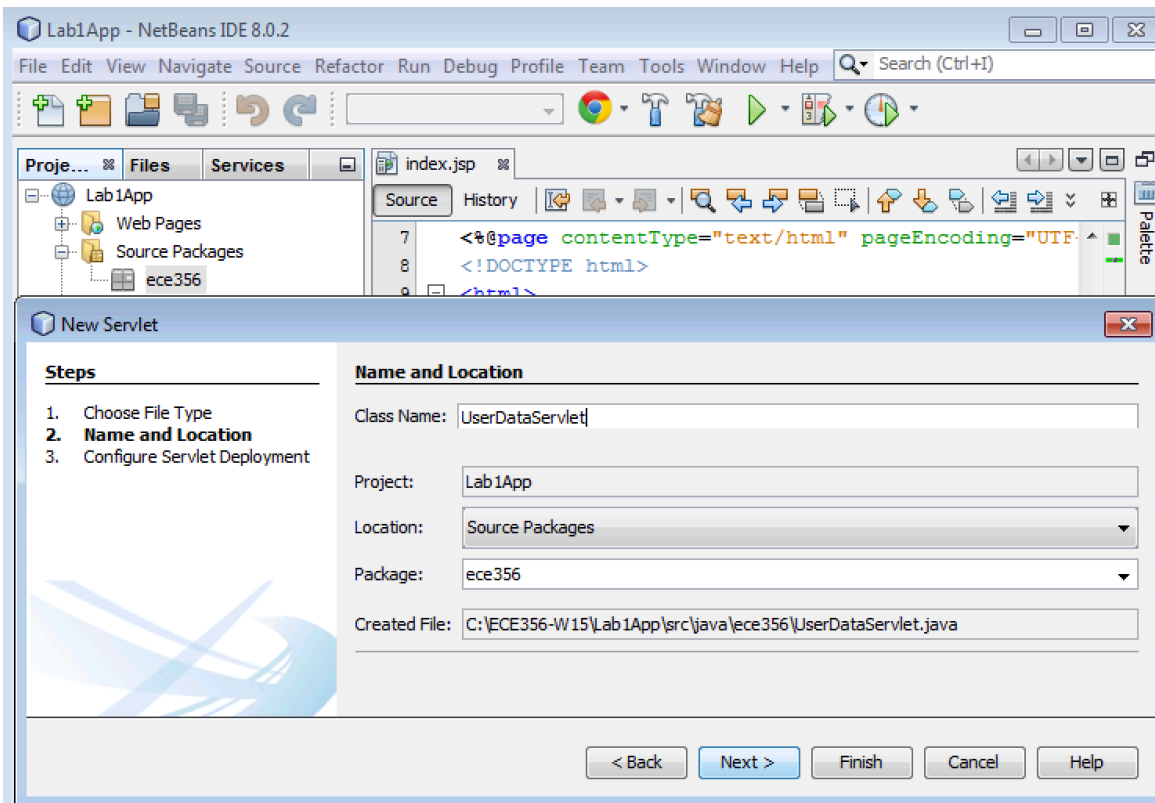
- In order to keep your application components organized you will first create a package which will contain all your servlets.
- Add a new package to your app (use option that appears in the right mouse click menu).



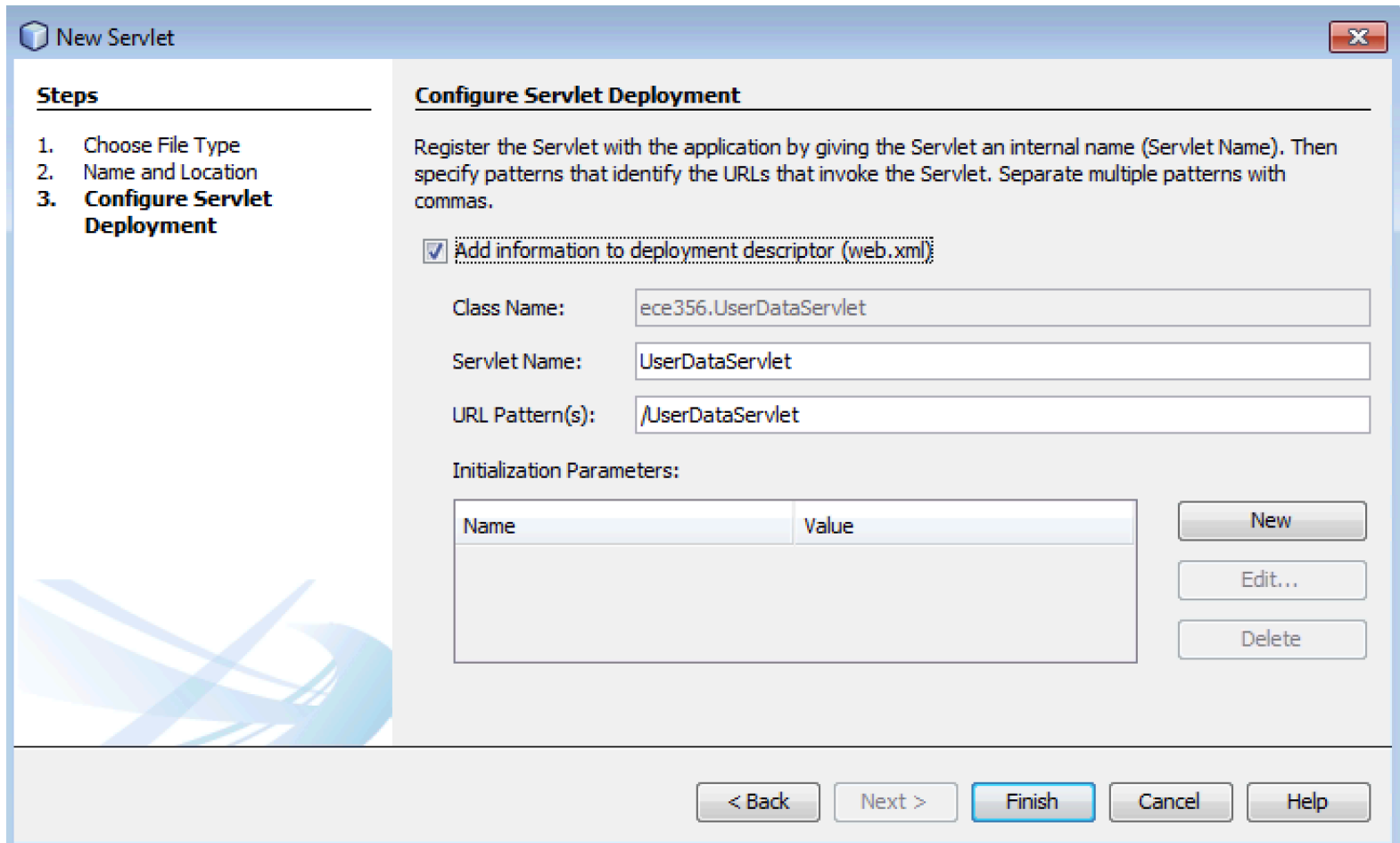
Exercise 2: Adding a Servlet under package

- Add a new servlet to your package (use option that appears in the right mouse click menu).

Important: Make sure you click the option “Add information to deployment descriptor (web.xml)” which appears in the second page of the servlet definition. If you miss this step your servlet will not be found when the web application attempts to invoke it.



Exercise 2: Add servlet under this package



The screenshot shows the 'New Servlet' dialog box with the 'Configure Servlet Deployment' step selected. The 'Steps' list on the left includes 'Choose File Type', 'Name and Location', and 'Configure Servlet Deployment'. The main area contains instructions to register the servlet and a checkbox for adding deployment descriptor information, which is checked. Below this are text fields for 'Class Name' (ece356.UserDataServlet), 'Servlet Name' (UserDataServlet), and 'URL Pattern(s)' (/UserDataServlet). An 'Initialization Parameters' table is empty, and buttons for 'New', 'Edit...', and 'Delete' are to its right. At the bottom are 'Back', 'Next >', 'Finish', 'Cancel', and 'Help' buttons.

New Servlet

Steps

1. Choose File Type
2. Name and Location
3. **Configure Servlet Deployment**

Configure Servlet Deployment

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

☒ Add information to deployment descriptor (web.xml)

Class Name: ece356.UserDataServlet

Servlet Name: UserDataServlet

URL Pattern(s): /UserDataServlet

Initialization Parameters:

Name	Value
------	-------

New Edit... Delete

< Back Next > Finish Cancel Help

Exercise 2: UserDataServlet.java

Now that you have added the JavaBean, you can complete the code for your servlet.

The code in this servlet will:

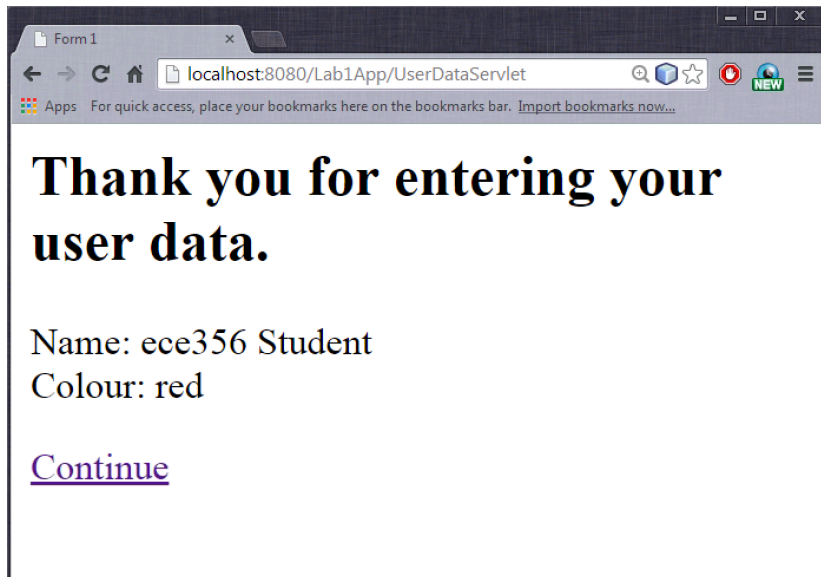
1. Get the parameters entered in the form and store them in the corresponding fields in an object of type *UserData*.
 - **Hint:** see online documentation for class **HttpServletRequest**, method **getParameter()**
2. Save the object *UserData* to the session request
 - **Hint:** see online documentation for class **HttpSession**, method **setAttribute()**
3. Display the values entered in the form using the provided jsp, *user_data_thanks.jsp*.
 - **Hint:** see online documentation for **getServletContext().getRequestDispatcher(..).forward(..);**

Try to write the code with no help, if you give up revisit slide 10

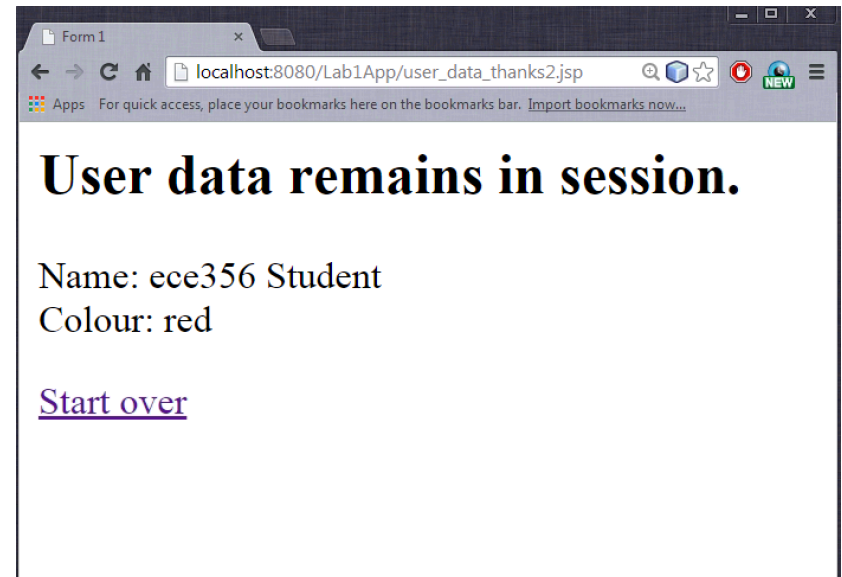
Exercise 2: Testing your web application

- Test your web application
 - Before you test, make sure that you have added to the project the needed components (JavaBean and JSPs) which were provided in the start code.
 - Steps for testing:
 1. Start your application, using “Run Project”, and click the option for “Exercise 2”.
 2. Enter name and color in first jsp
 3. See how this info is displayed in the second jsp
 4. When you click on “continue” in the second jsp, you will see that the info is also available to the third jsp, since the servlet has saved the info in the session object of the web application.

Exercise 2: Testing your web application



user_data_thanks.jsp



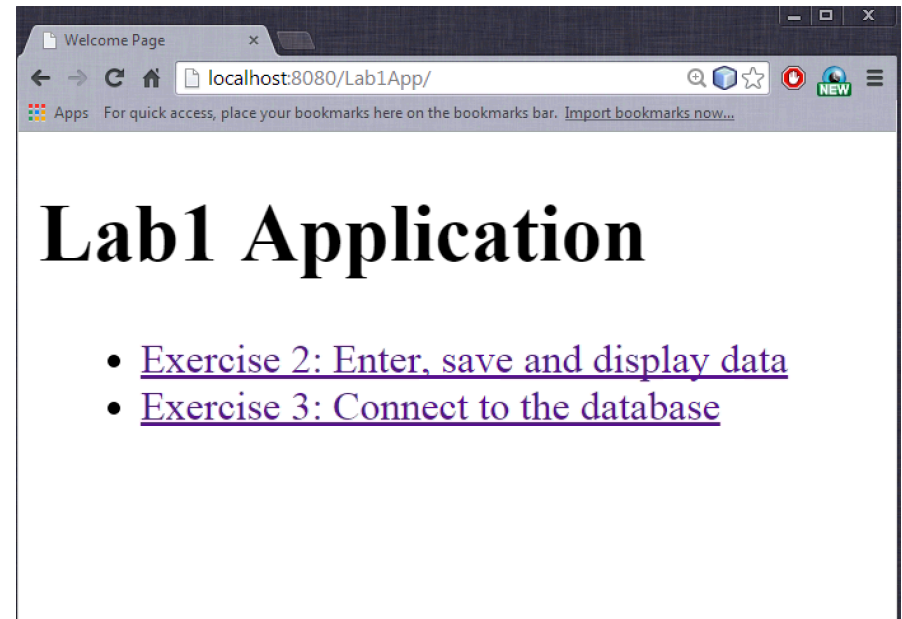
user_data_thanks2.jsp

Exercise 3: Connect to the MySQL database from your web application

- In this exercise you will learn
 - How to add libraries which allow you to interface with a database to your project.
 - How to add software classes to interact with the database following the MVC pattern.

Exercise 3: Add option to connect to the MySQL database in the server

- Modify the first page of your application, index.jsp:
 - Add a link that allows the user to connect to the database
 - When this link is clicked, your web application will execute the servlet DBTestServlet.java (you will write this servlet in the next part of this lab)
 - The following is snapshot of index.jsp after these modifications



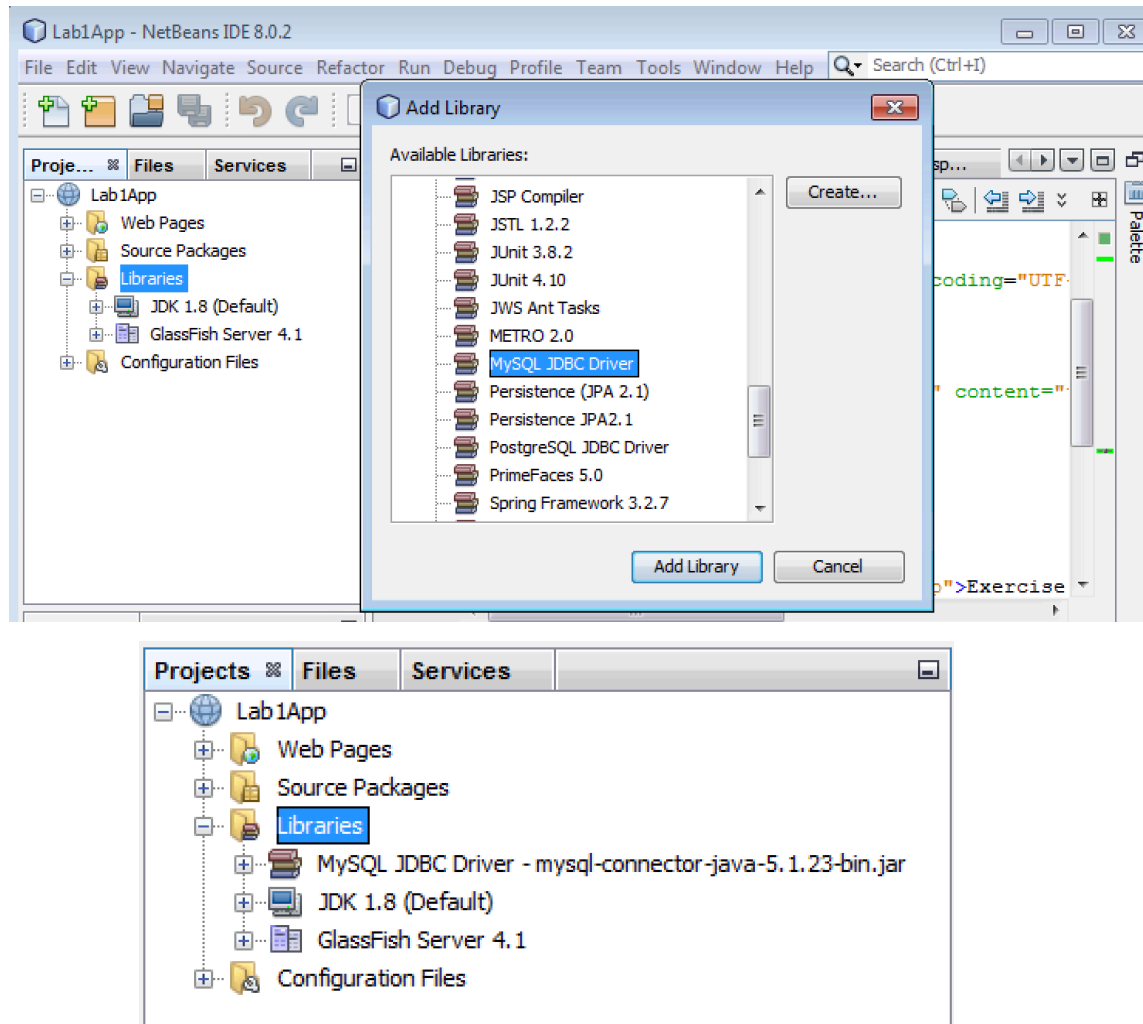
index.jsp

Exercise 3: Model-View-Controller architecture(MVC)

- Your web project uses the MVC software architecture pattern. The MVC pattern includes:
 - JSP to receive and present data to the user (View)
 - Servlet to implement navigation and business logic (Controller)
 - JavaBean – use to encapsulate related properties in a single java object (Model)
 - Data Access Object (DAO) - object that provides an abstract interface to a database(Model)

Exercise 3: Adding MySQL JDBC driver to project

- MySQL provides connectivity for client applications developed in the Java programming language through a JDBC driver, which is called MySQL Connector/J.
- To add the library for the MySQL Connector/J, to your web project
 - Add a new Library
 - select the “MySQL JDBC Driver” from the list of available libraries
 - Your “Projects” tab should now show the MySQL JDBC Driver jar file.



Exercise 3: UserDBAO.java

- Servlet DBTestServlet uses the services of a data access object(DAO) called to [UserDBAO.java](#) to interact with the database. The code for [UserDBAO.java](#) has been provided.
- Modify the method TestConnection() in the UserDBAO.java so that the database, user id, and password corresponds to your course database in the ecweb server. I.e. if your uw userid is **ece356_test**, your parameters should look as follows:

```
public static final String url = "jdbc:mysql://ecweb.uwaterloo.ca:3306/";  
public static final String user = "user_ece356_test";  
//use your UW username instead of ece356_test  
public static final String pwd = "user_ece356_test";  
// specify your password here, initially same as userid
```

Note (First Lab Only): the ecweb server is temporarily down. Use the following parameters for the first Lab.

```
Server: snorkel.uwaterloo.ca  
Database: ece356db_wgolab  
User: user_ece356_test  
Password: user_ece356_test
```

Exercise 3: Add DBTestServlet.java

- Add a new Servlet to package ece356, named **DBTestServlet.java**. Make sure to choose the option “Add information to deployment descriptor (web.xml)” on the in the second page of the servlet definition.
- You will later edit this servlet and add the java code needed to connect to the DB as part of the method processRequest()

Exercise 3: DBTestServlet.java processRequest() method

- Modify method processRequest() in DBTestServlet.java to connect to the database.
 - Add call to TestConnection method in UserDBAO.java:
UserDBAO.testConnection();
 - The statement above may throw an exception when the attempt to connect is not successful. Your code should handle this exception. You can proceed to the next slide if you need an example.

Exercise 3: DBTestServlet.java

processRequest() method

- Handling exceptions – setting the value of variable with name of jsp to be displayed:

String url;

```
try {  
    // Call static method using class name  
    UserDBAO.testConnection();  
    // Set the name of jsp to be displayed if  
    // connection succeeds  
    url = "/dbtest.jsp";  
} catch (Exception e) {  
    // Save the exception info in the request object so it can be displayed.  
    request.setAttribute("exception", e);  
  
    // Set the name of jsp to be displayed if  
    // connection fails  
  
    url = "/error.jsp";  
}
```

DBTestServlet

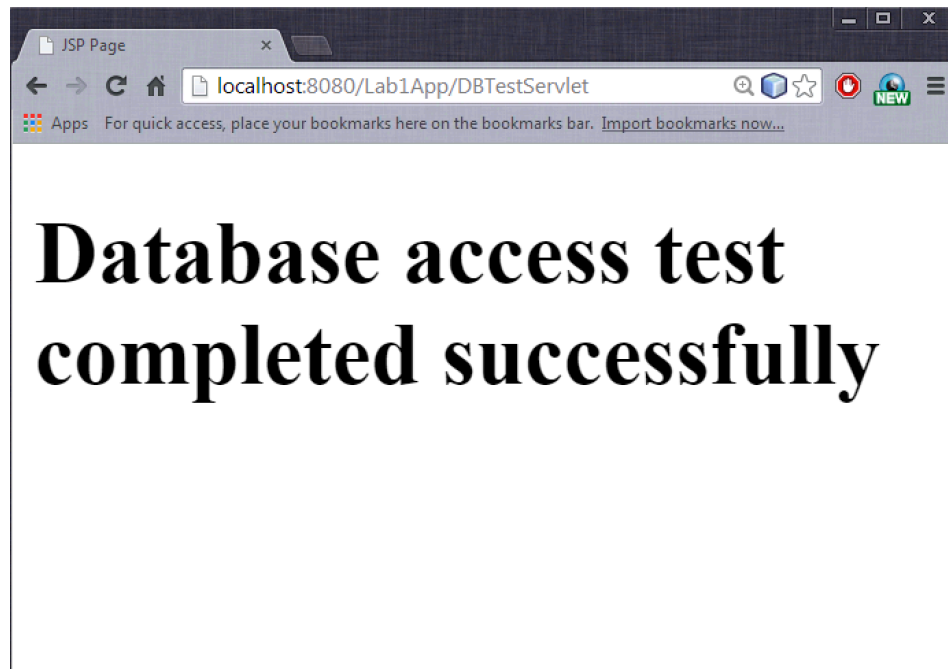
processRequest()

Displaying the suitable jsp from the servlet (name of jsp is now stored in variable “url”):

```
getServletContext().getRequestDispatcher(url).forward(request, response);
```

Exercise 3: Connection successful

- Create a jsp named **dbtest.jsp**. This jsp displays a message that indicates that the connection is successful



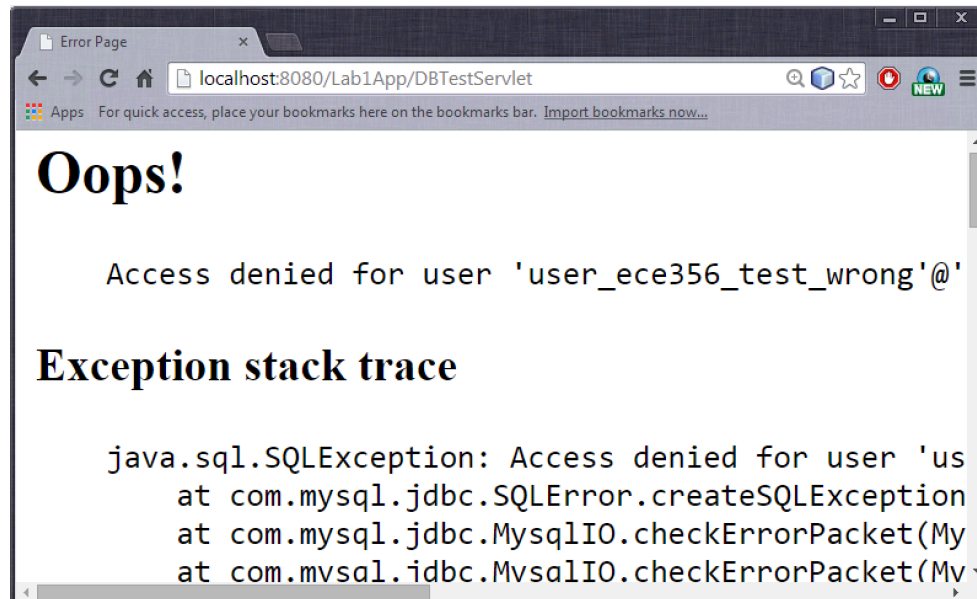
dbtest.jsp

Exercise 3: Test Connection

- Test both possible cases:
 - Connection successful
 - Connection unsuccessful
 - Test for several possible errors. For example, invalid user id, or password, invalid server name.

Exercise 3: Connection unsuccessful

- Add error.jsp, which has been provided.
- Notice that the code in error.jsp:
 - Uses request.getAttribute() to get the value of attribute “exception” (this attribute was saved to the request object in DBTestServlet.java)
 - Prints error message and stack trace in this attribute



error.jsp

Summary

In this lab we covered the following:

- Created a web project with NetBeans
- Used Java JSP Model 2, which is a realization of Model-View-Controller architecture.
 - Used JSPs and a HTML form to collect data from the user.
 - Used a JavaBean class to represent user data.
 - Stored and accessed data from a JavaBean in JSPs and servlets.
 - Connected to a database from a servlet, using a DAO object to interface with the database.

References

Java, JSP and JDBC

[The Java Tutorials](#)

[The Java Tutorial - Trail: JDBC\(TM\) Database Access](#)

[Introduction to Java Server Pages\(JSP\)](#)

Web Application Development

[The Java EE 6 Tutorial](#)

[Tutorial - Creating a Simple Web Application Using a MySQL Database \(using NetBeans\)](#)

References

- Reference for html
www.w3schools.com
- Reference for MVC and JSPs:
<http://netbeans.org/kb/docs/javaee/ecommerce/design.html?print=yes>
- Ref for DAO :
<http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>