Java EE 6: Develop Web Components with Servlets & JSPs

Activity Guide

D77750GC10
Edition 1.0
March 2013
D81230

ORACLE

Authors

Edgar Alberto Martinez Cruz, Eduardo Moranchel Rosales

This book was published using: Oracle Tutor

Table of Contents

Practices for Lesson 1: Environment Set Up

Chapter 1

Practices for Lesson 1

Practices Overview

In these practices, you will set up the Oracle's WebLogic Server and deploy a web project into your configured server using NetBeans.

Practice 1-1: Set Up WebLogic to Run Using NetBeans

Overview

In this practice, you set up WebLogic Server and deploy the example project using NetBeans. Using the deployed application, you can review the structure of a NetBeans Java Web Project.

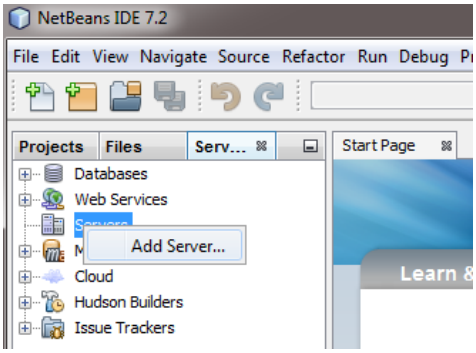
Assumptions

- WebLogic Server is already downloaded on the lab machines and is unzipped and configured at `d:\weblogic\wls`.
- There is a WebLogic domain created at `d:\weblogic\domain`.
- The username for the domain administration is `weblogic`.
- The password for the domain administration is `welcome1`.

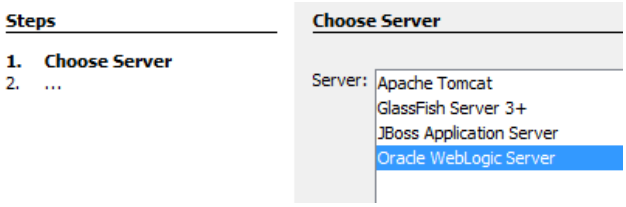
Practice Tasks

1. Open NetBeans IDE.

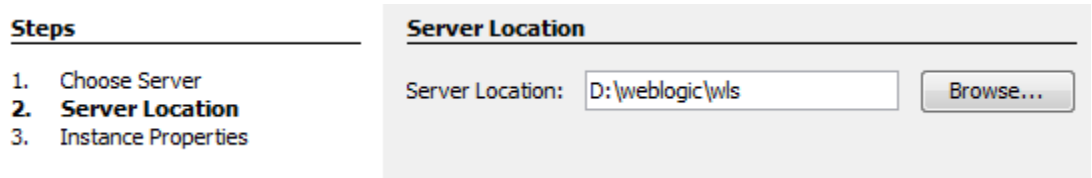
2. Click the Services tab, right-click the Servers node, and select Add Server.



3. From the Server list, select Oracle WebLogic Server and click Next.

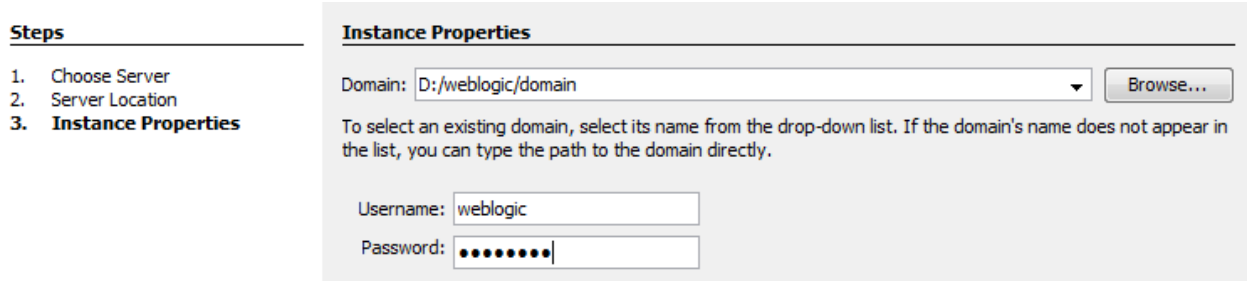


4. Set the server location to `D:\weblogic\wls` and click Next.

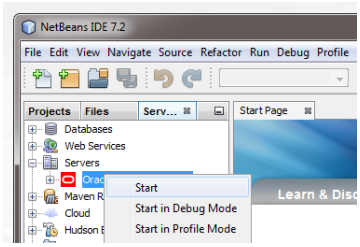


5. The domain should be set to `D:\weblogic\domain`. If it is not, set it.

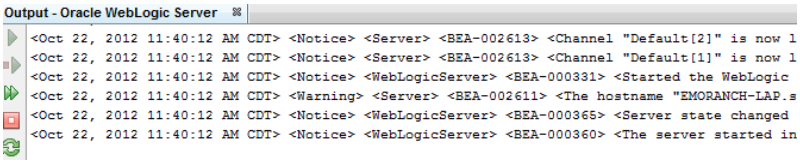
6. Set `weblogic` as the username and `welcome1` as the password.



7. Click finish.
8. Start the server by right-clicking the Oracle WebLogic Server node under Services and clicking Start.

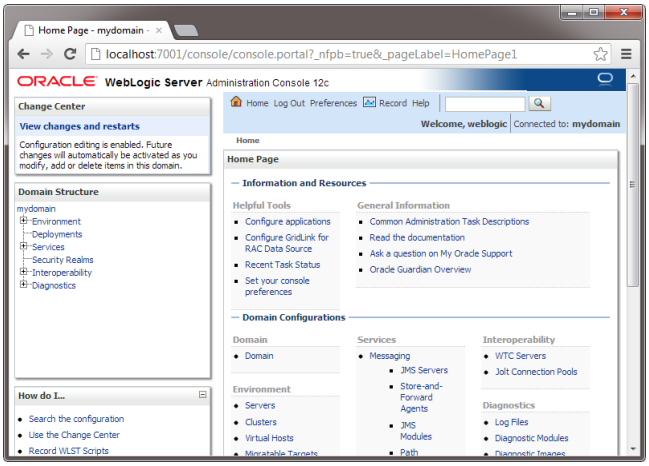


9. The server starts and its console is displayed in the Output view in NetBeans.



10. Open a browser and navigate to `http://localhost:7001/console`. After waiting for the console application to load, you should see something like this:

11. Log in using the username `weblogic` and the password `welcome1`.



You should see the WebLogic Administration Console Home.

Practice 1-2: Deploy the Example Project

Overview

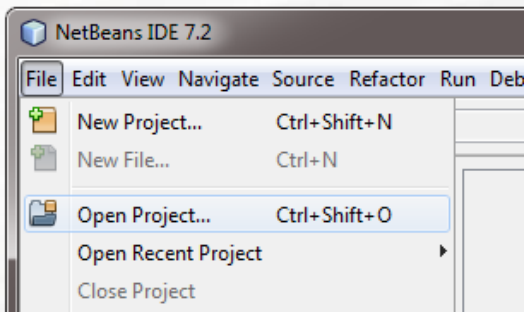
In this practice, you deploy an Example Project in WebLogic using NetBeans.

Assumptions

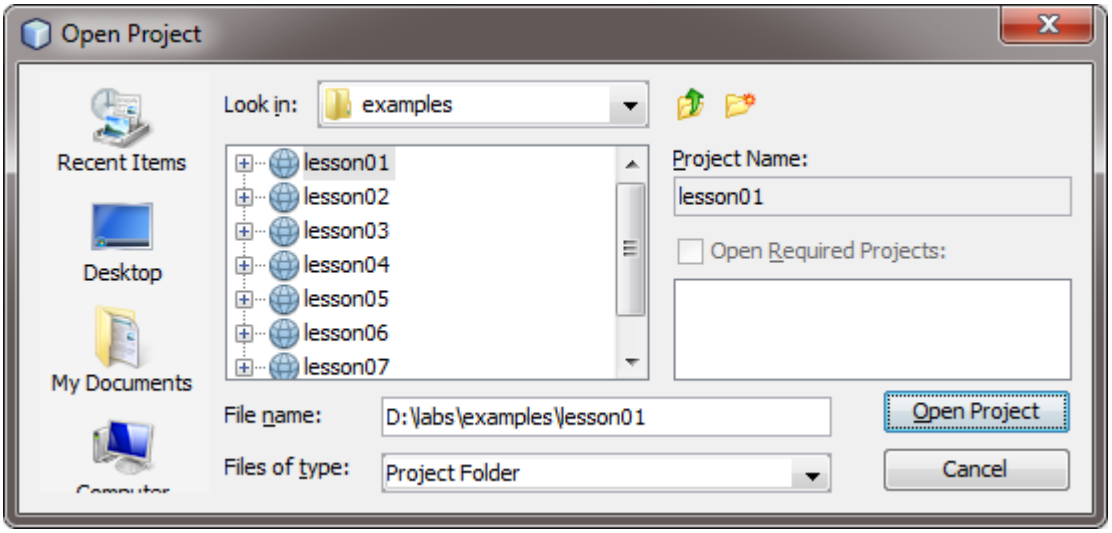
WebLogic is set to run using NetBeans

Practice Tasks

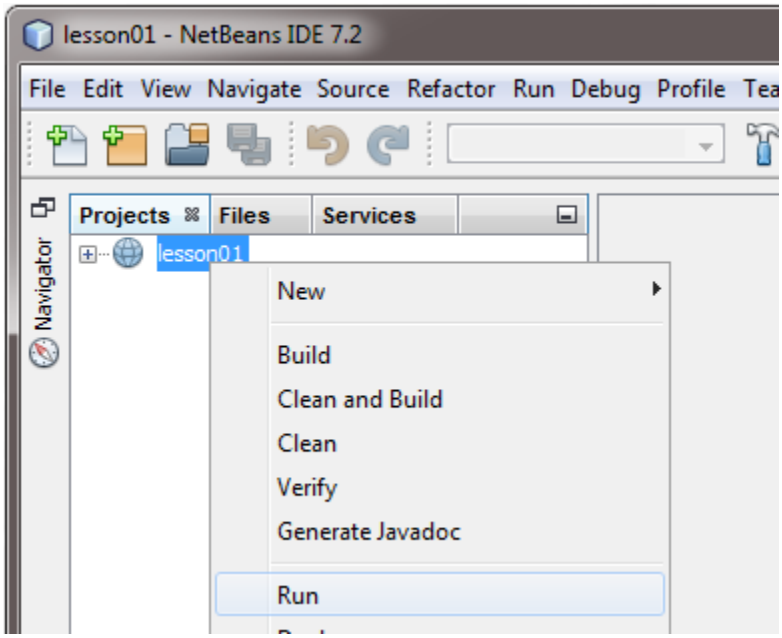
1. In NetBeans, select File > Open Project.



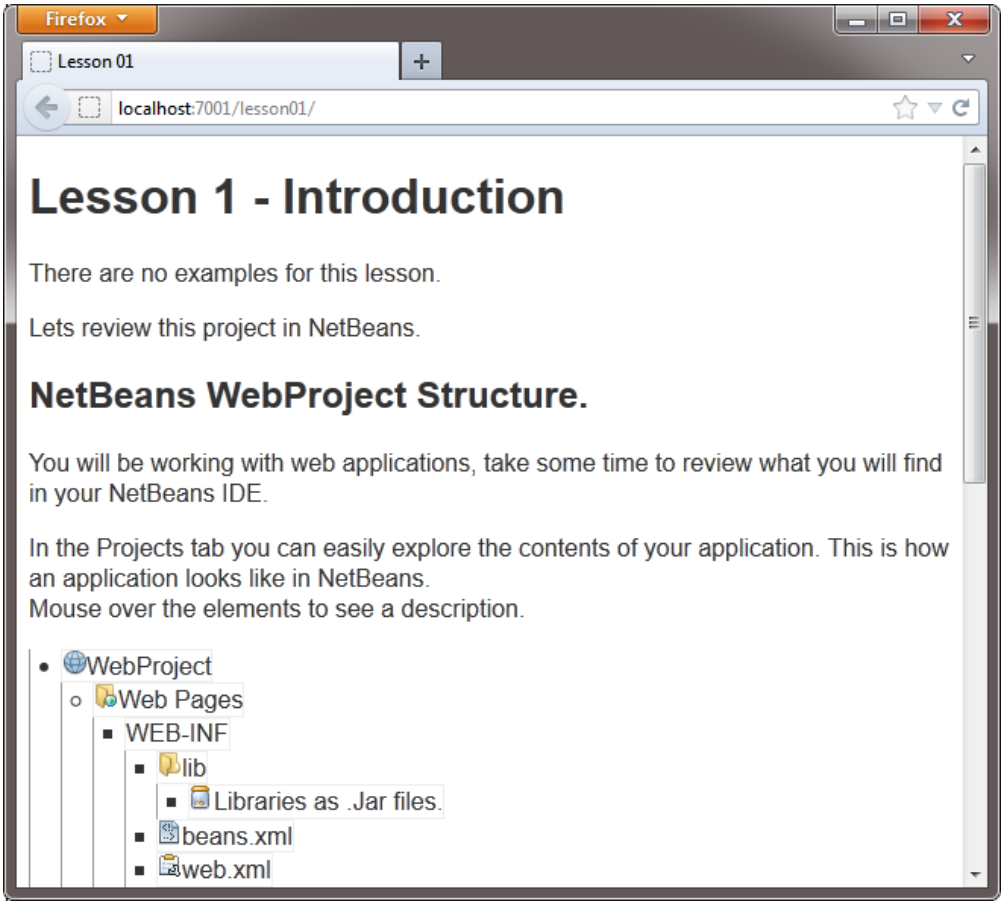
2. Go to `D:\labs\examples` and open the `lesson01` project.



3. Right-click the `lesson01` project and select Run from the menu.

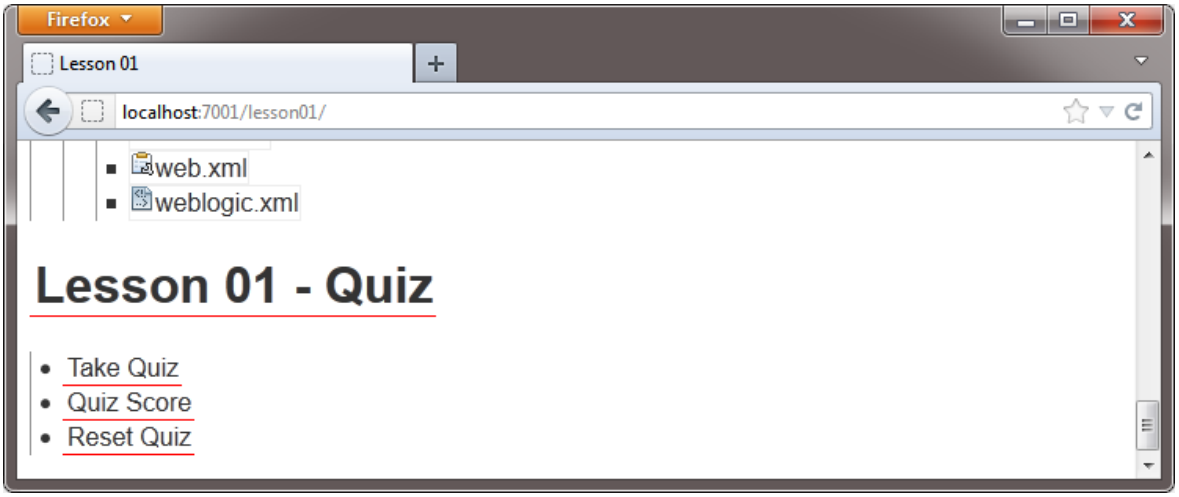


4. A browser opens at the URL `http://localhost:7001/lesson01/`.



Note: If you encounter an intranet warning asking you for confirmation to proceed, accept it. This warning is displayed because you are accessing a local URL (localhost or your intranet's IP).

5. In your NetBeans IDE and in your browser, explore the example contents, review the structure of the NetBeans web projects, and then take the quiz for this lesson (click the Take Quiz link at the bottom of the webpage to access the quiz).



Additional Information

All lessons include examples that can be deployed in the same way that you deployed the application in practice 1-2. If you want to review or test your knowledge of each lesson, deploy its examples, review them, and take the quiz at the end.

The `d:\labs\examples\` folder contains example projects that can be deployed in the same way that you deployed this example project. Open the project in NetBeans, right-click it, and select Run from the menu to deploy and run each example.

Example applications run the demo source code from within themselves. They contain reference material and a quiz section for the lesson where you can test your knowledge.

Deploy these examples after you complete your practices for the lesson to enhance your learning experience.

Practices for Lesson 2: Deployment

Chapter 2

Practices for Lesson 2

Practices Overview

In the practice for this lesson, you will learn how to use the WebLogic console to deploy WAR files.

Practice 2-1: Deploying the Examples from a WAR File

Overview

In this practice, you deploy a WAR file using Oracle WebLogic Server.

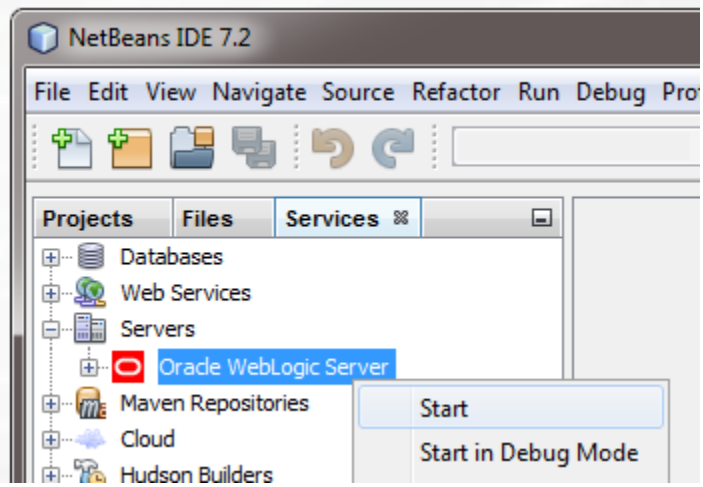
Assumptions

WebLogic has been configured per Practice 1-1.

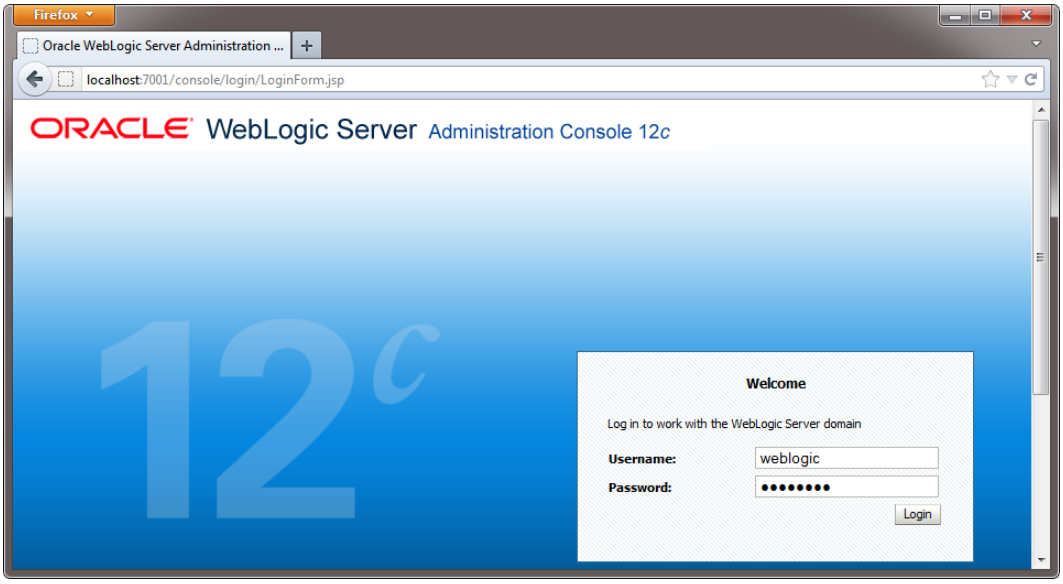
Tasks

1. Start Oracle WebLogic Server if it is not already running.

   To do so, open the Services tab in NetBeans, and then right-click the Oracle WebLogic Server node and select Start.

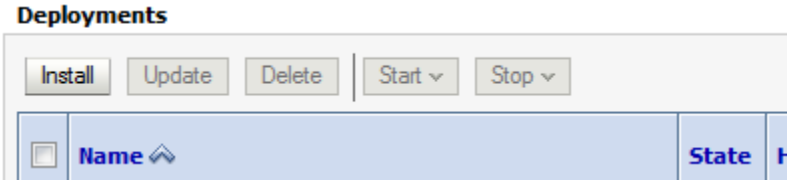


2. Open a web browser and go to `http://localhost:7001/console`. You should see the WebLogic Server login screen.
3. Log in with the username `weblogic` and password `welcome1`.

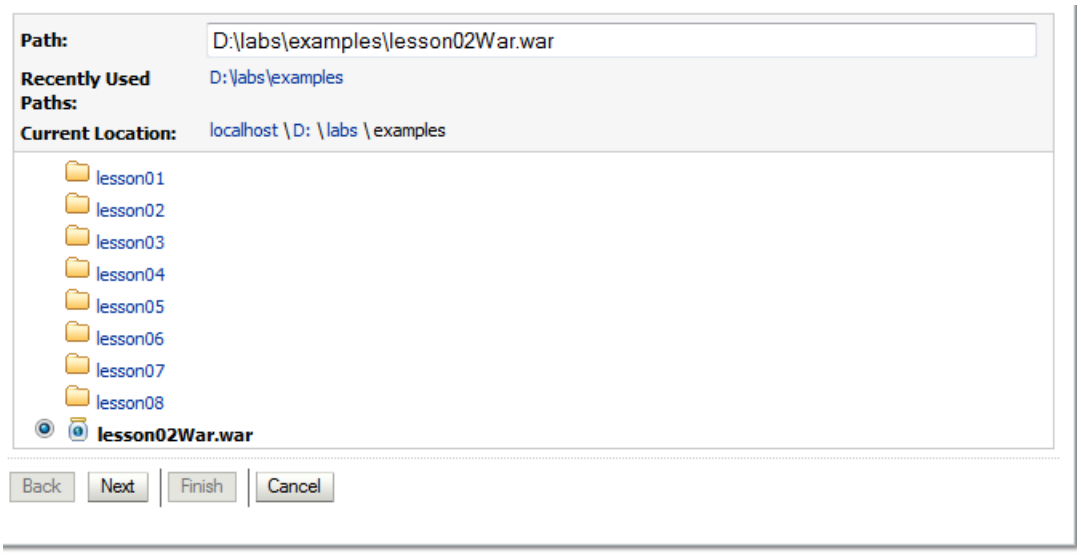


4. Click the Deployments link.

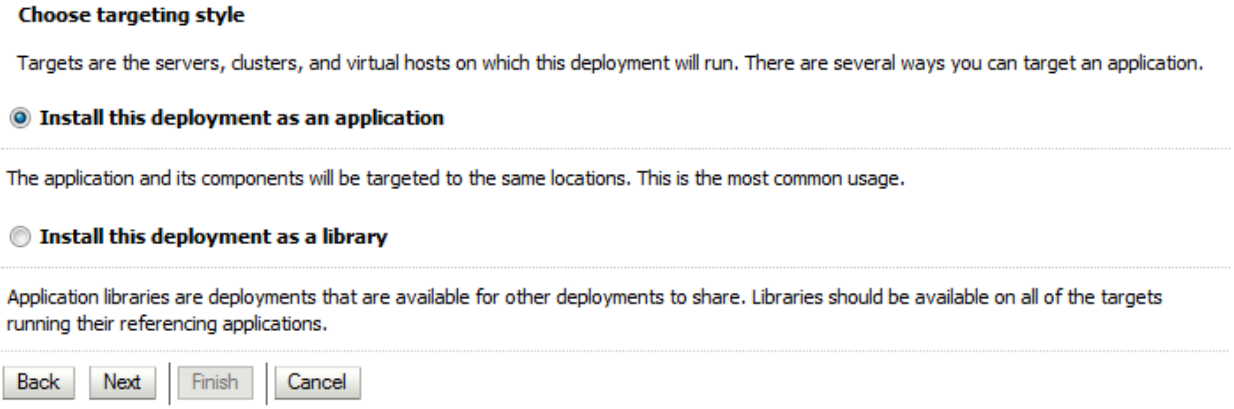
5. On the next page, click Install.



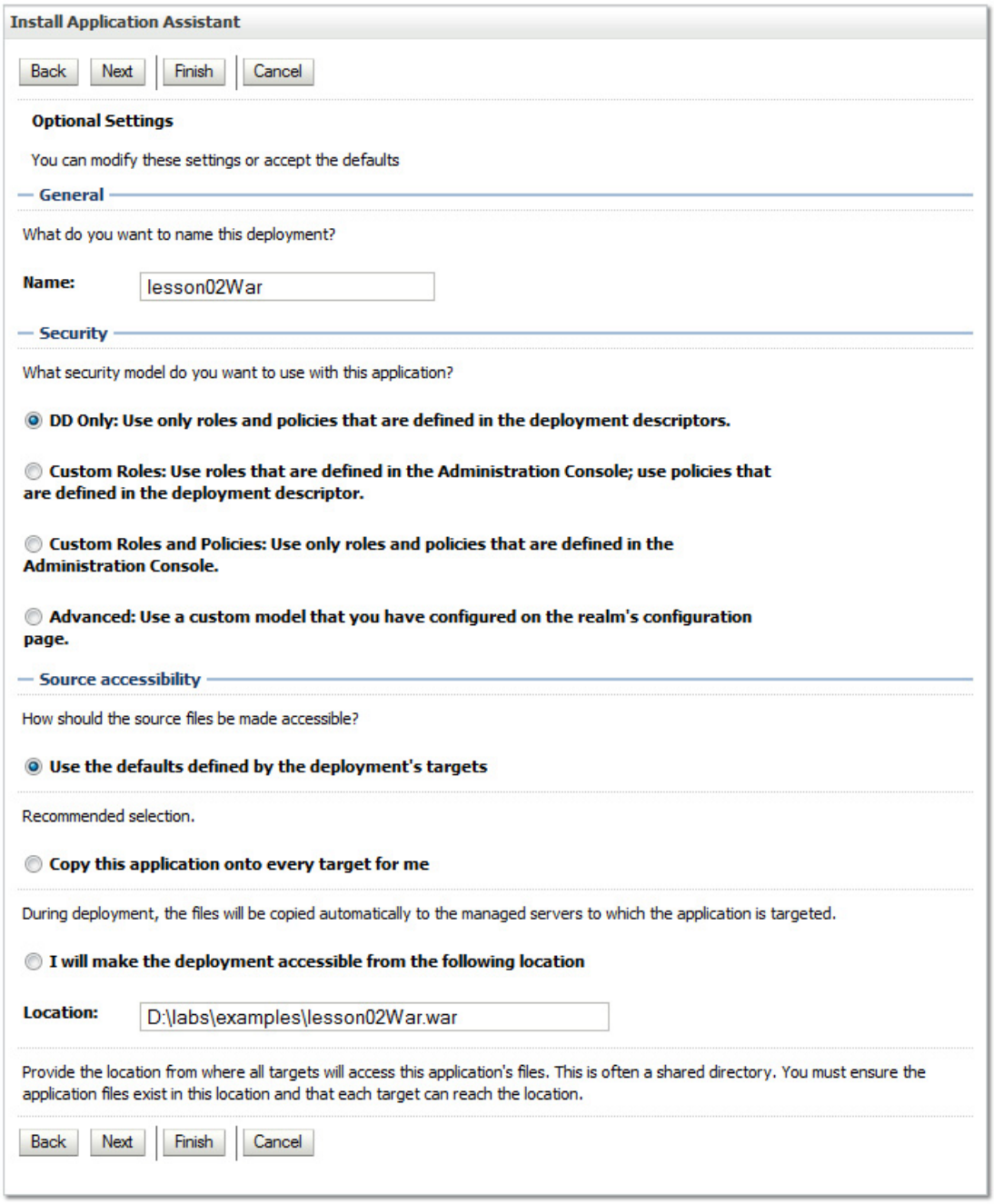
6. Browse to `D:\labs\examples\`, select the `lesson02War.war` file, and click Next.



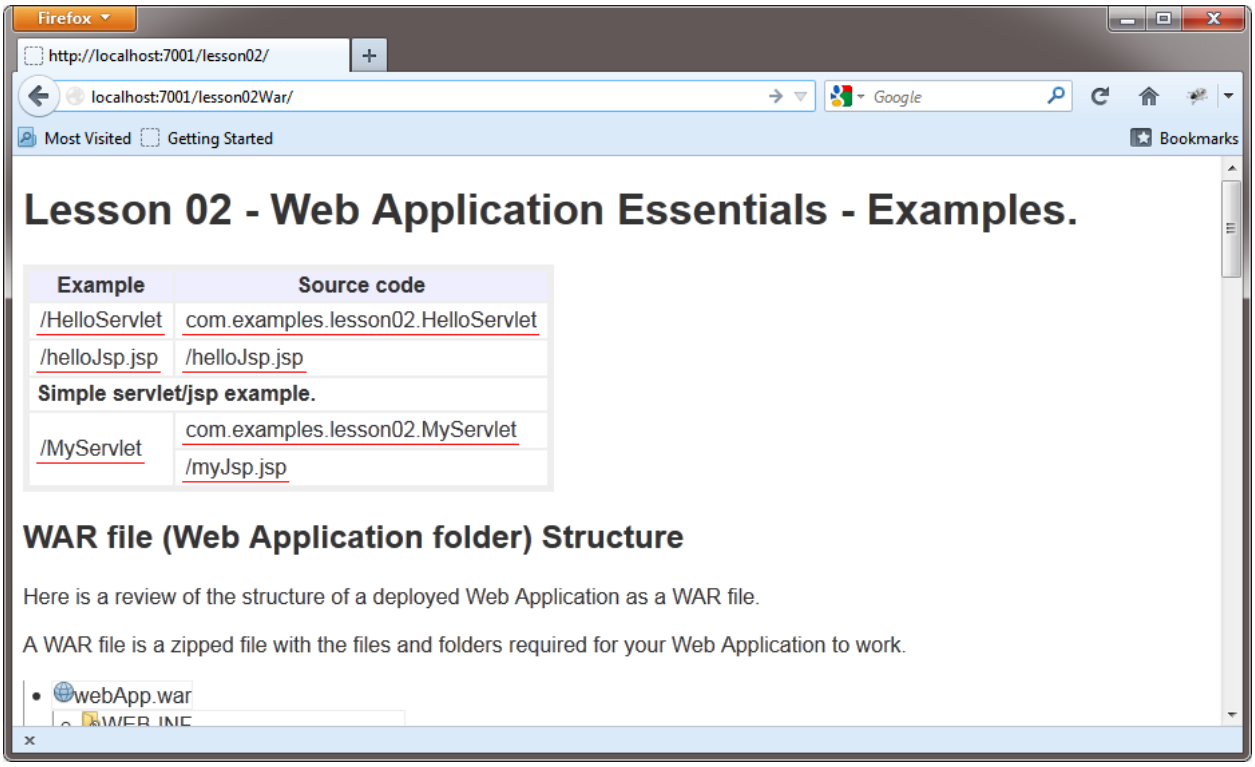
7. Select “Install this deployment as an application” and click Next.



8. Click `Finish`. The Quiz application is deployed on the server.

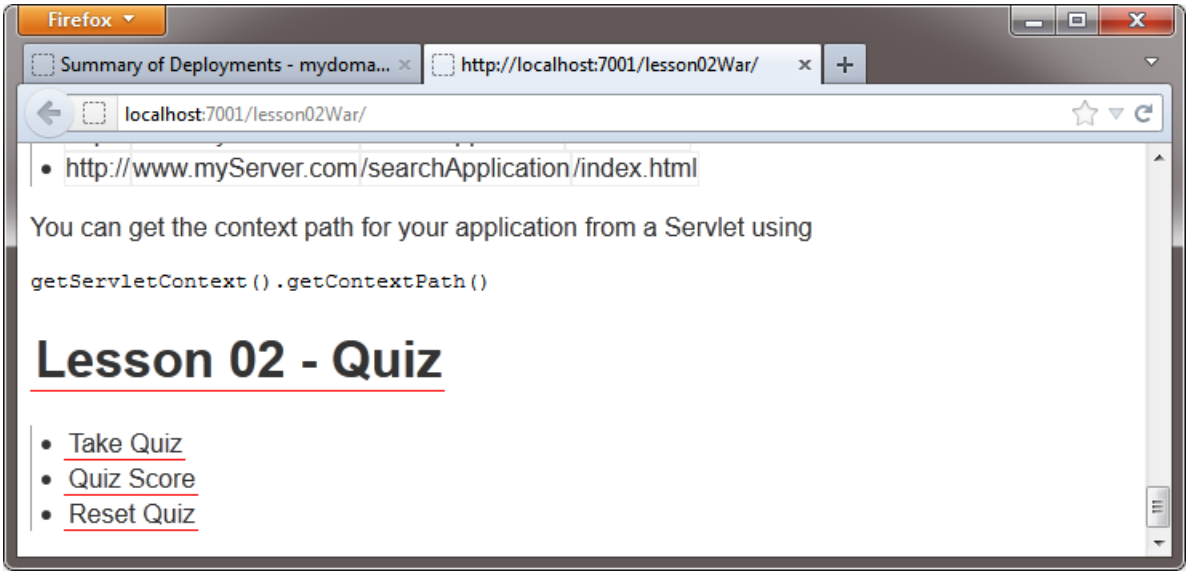


9. Open a web browser and go to http://localhost:7001/lesson02War/.
   You should see the homepage for the example project.



   In this application, you can review the internal structure of a WAR file.

10. Scroll to the bottom of the webpage, and use the links to take the quiz for the Lesson 2.



Additional Information

In this practice, you deployed a WAR file in WebLogic.

WAR deployment is frequently used in production environments and it is the preferred way of distributing web applications.

The application that you deployed contains a review of the internal structure of a WAR file and the quiz for Lesson 2.

You may also open the `d:\labs\examples\lesson02` project in NetBeans to see the project that generated the WAR file that you deployed in this practice.

Practices for Lesson 3: Creating Servlets

Chapter 3

Practices for Lesson 3

Practices Overview

In these practices, you will create simple web applications that use servlets to display dynamic information in a web browser.

Practice 3-1: Creating a Simple Visitor Count Servlet

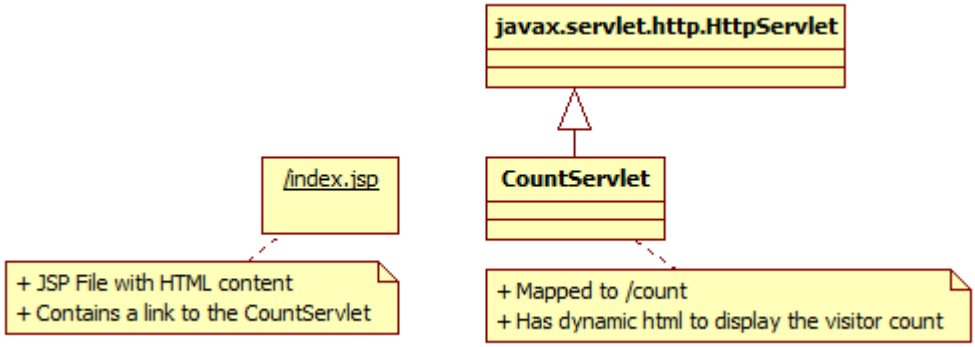
Overview

In this practice, you create a simple visitor count service that displays the number of times your page has been accessed.

Your application starts in `index.jsp` and has a link to the servlet that you will create.

The servlet counts the times that it has been accessed. Remember that there is only one instance per servlet so instance variables are the same for each call to the Servlet service methods.

The following diagram outlines your web application content, mappings, and navigation.



Note: The `index.jsp` in this practice contains only HTML code and it is treated as a simple HTML file.

Practice Overview

To create a web application with a counter servlet, you do the following:

1. Open NetBeans and create a new web application project named `lab_03_01`.
2. Create a new servlet named `CounterServlet` in the `com.lab0301.servlets` package.
3. Map `CounterServlet` to the `/counter` URL.
4. Modify the `index.jsp` file and add a link to `/counter` to go to `CounterServlet`. Add the following HTML code:

```
<a href="count">Go to count</a>
```

5. Add a counter `int` instance variable to `CounterServlet`.

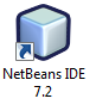
```
private int counter = 0;
```

6. In the `processRequest` method inside `CounterServlet`, increment the counter instance variable.
7. Add HTML to display the value of the counter variable:

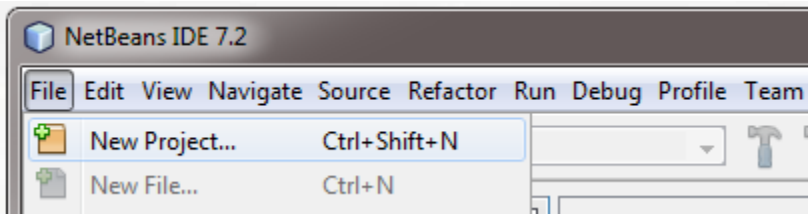
```
out.println("This servlet has been accessed " + count + "
times.");
```

8. Test the application by opening a web browser and opening the following URL: http://localhost:7001/lab_03_01/index.html

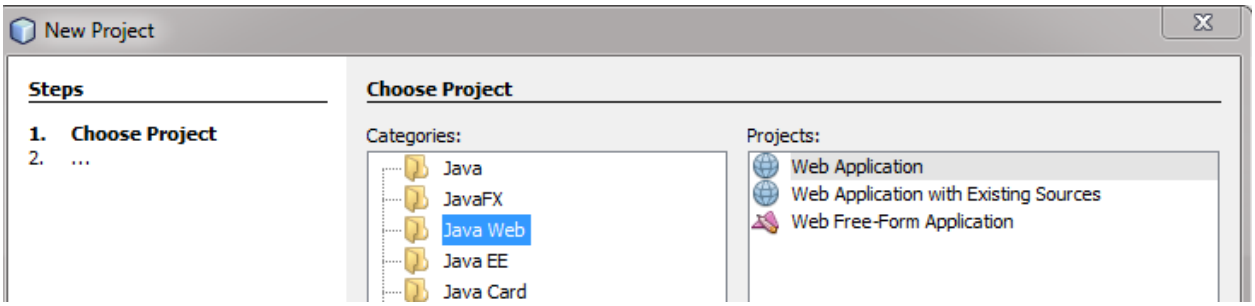
Step-by-Step Instructions



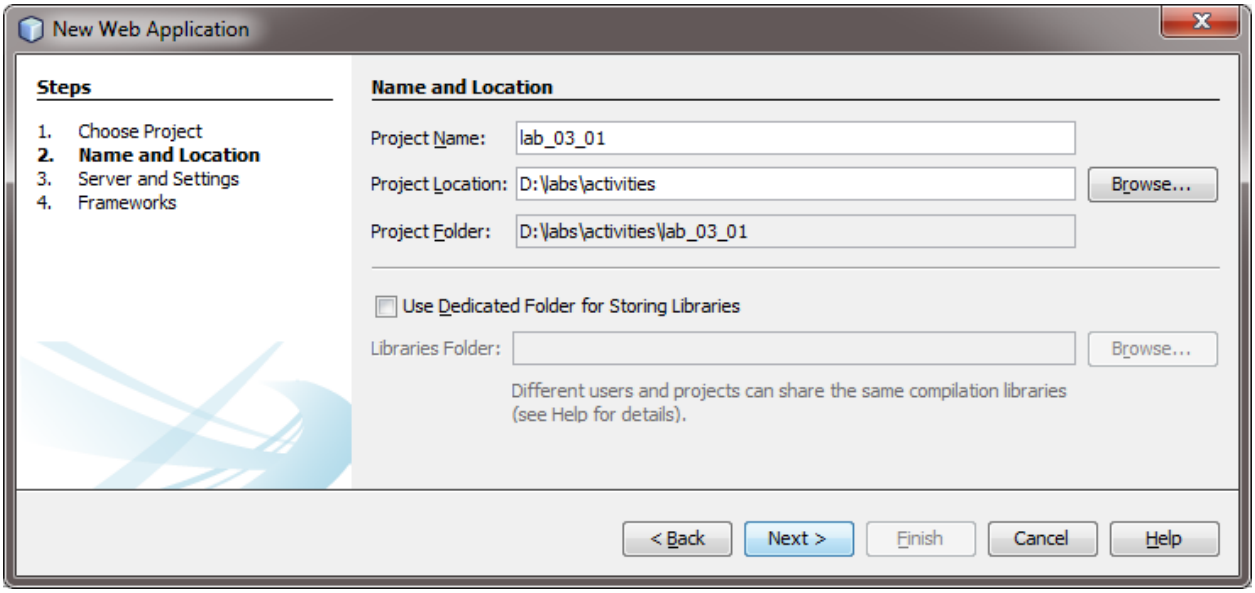
1. Open NetBeans IDE.
2. From the File menu, select New Project.



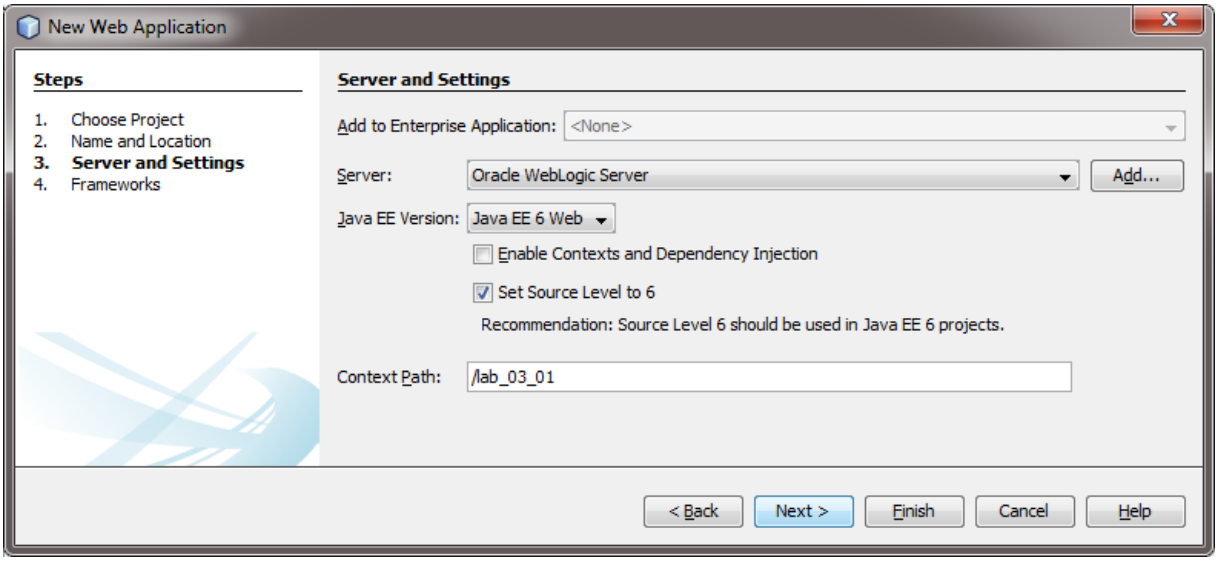
3. From the Choose Project lists, select the Java Web category and Web Application project.



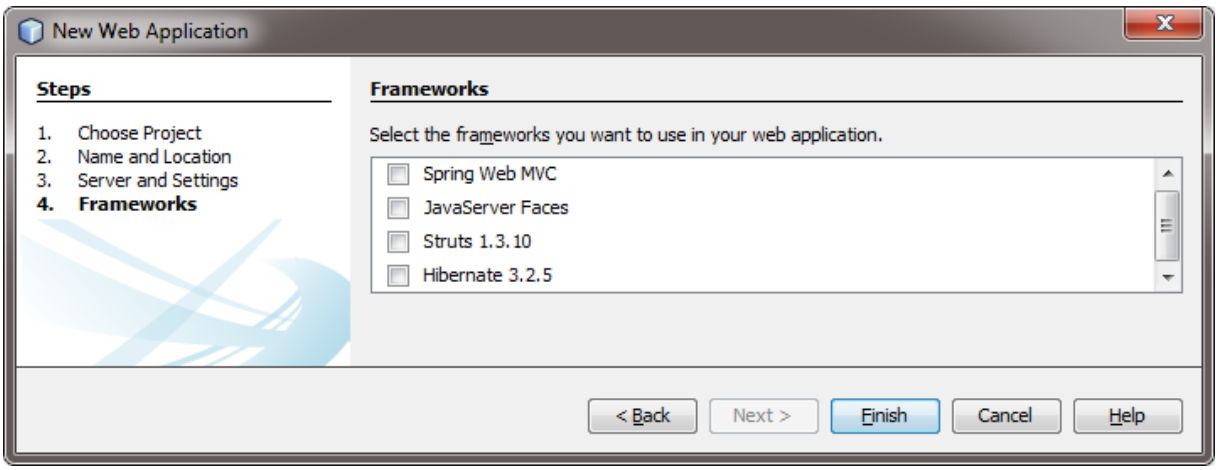
4. Click Next
5. Name the project `lab_03_01` and browse to select `D:\labs\activites` as the location. Click Next.



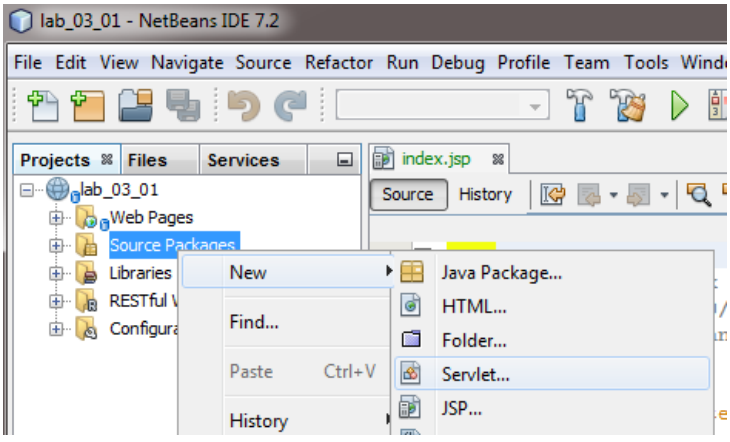
6. For “Server and Settings,” select Oracle WebLogic Server and verify that the Java EE version is set to Java EE 6 Web. Click Next.



7. For Frameworks, leave all options unchecked. Click Finish.

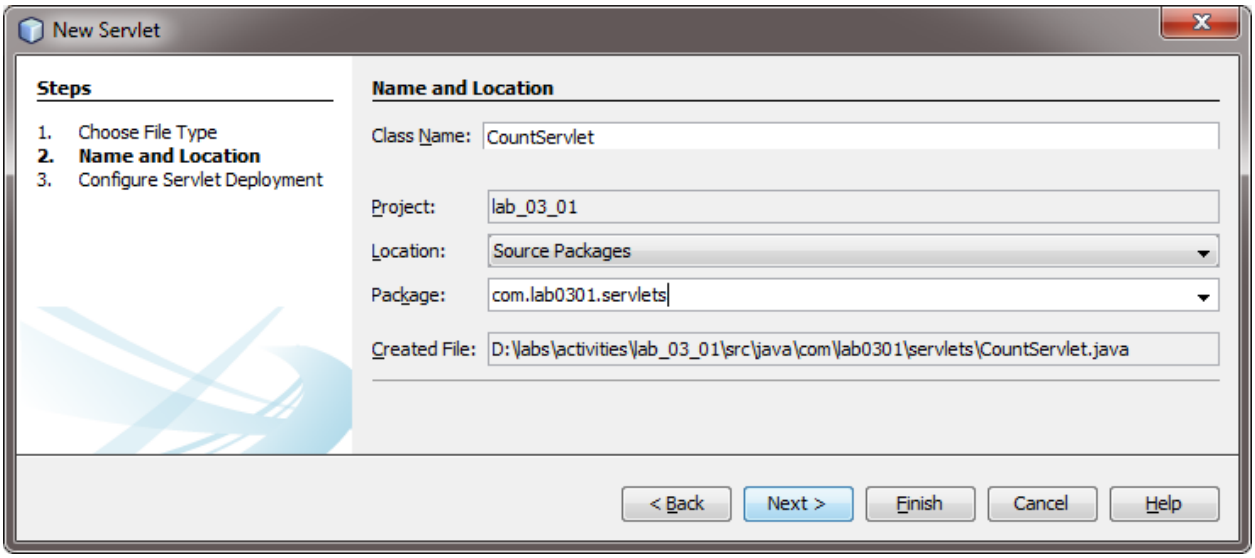


8. Open the Projects tab. Expand the project, right-click Source Packages, and select New > Servlet.

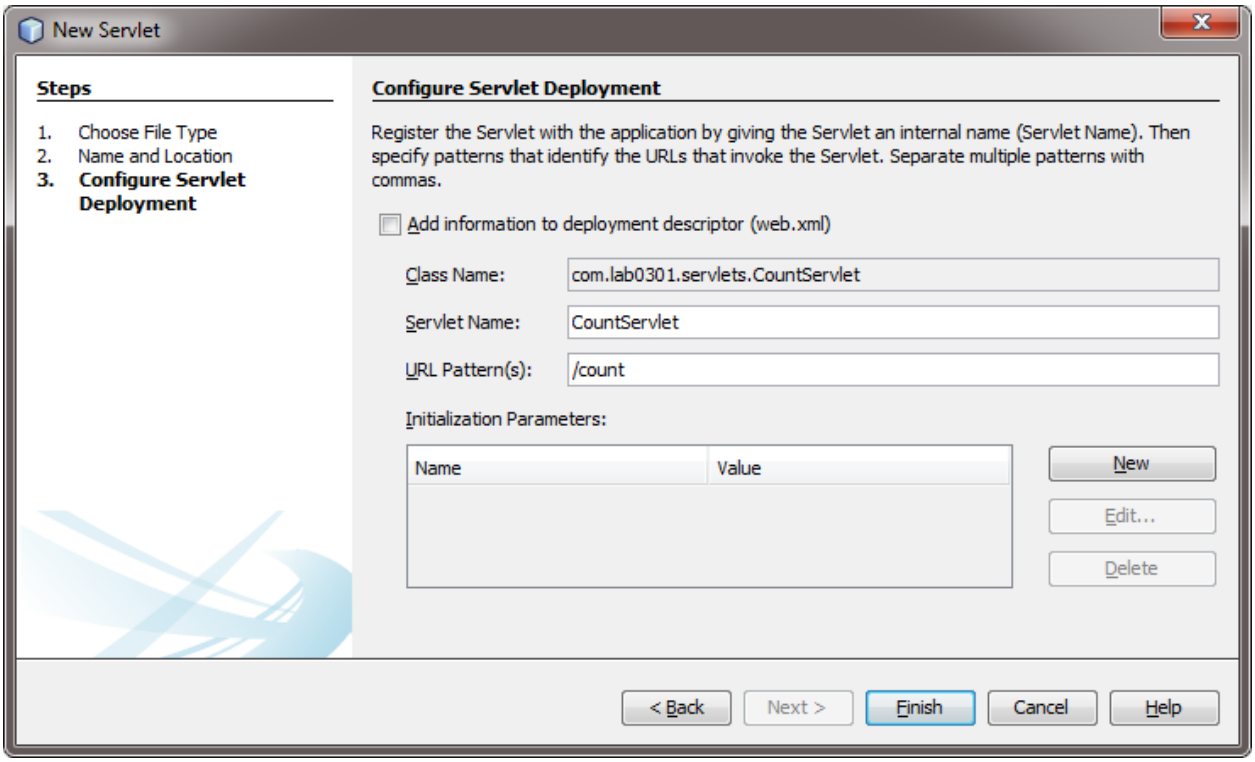


- If the Servlet Option is not available in the menu, select Other.
- In the dialog box that opens, select Web in the Category list and then Servlet in the File Types list.

9. Set the name of the servlet to `CountServlet` and the package to `com.lab0301.servlets`.
10. Click Next.



11. For Configure Servlet Deployment, change the URL pattern to `/count`. Click Finish.



The Servlet class is created and opens in the editor.

12. Open the `index.jsp` file. Add the highlighted code to link the count servlet. Note that you use the URL that you mapped in the servlet configuration.

```
<%@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>
    <a href="count">Go to count</a>
  </body>
</html>
```

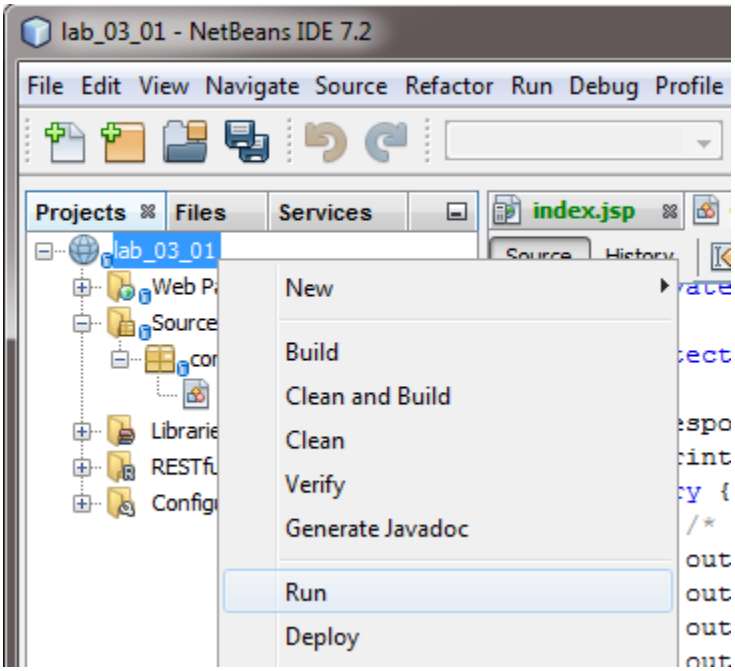
13. Open the `CountServlet` Java class. Locate the `processRequest` method. It has some HTML output. Add the highlighted code to display the hit count.

```
@WebServlet(name = "CountServlet", urlPatterns = {"/count"})
public class CountServlet extends HttpServlet {

  private int count = 0;

  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    count++;
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      /* TODO output your page here. You may use following sample code. */
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Servlet CountServlet</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>Servlet CountServlet at " + request.getContextPath() + "</h1>");
      out.println("This servlet has been accessed " + count + " times.");
      out.println("</body>");
      out.println("</html>");
    } finally {
      out.close();
    }
  }
```

Remember that only one instance per servlet is created by the web server.

14. The count instance variable is incremented with each call, because you are using the same instance for all the requests to the servlet.

15. Deploy your web application, right-click the project, and select Run.



16. In your web browser, click the "Go to count" link. If your web browser did not start automatically, open it and go to http://localhost:7001/lab_03_01/index.jsp.

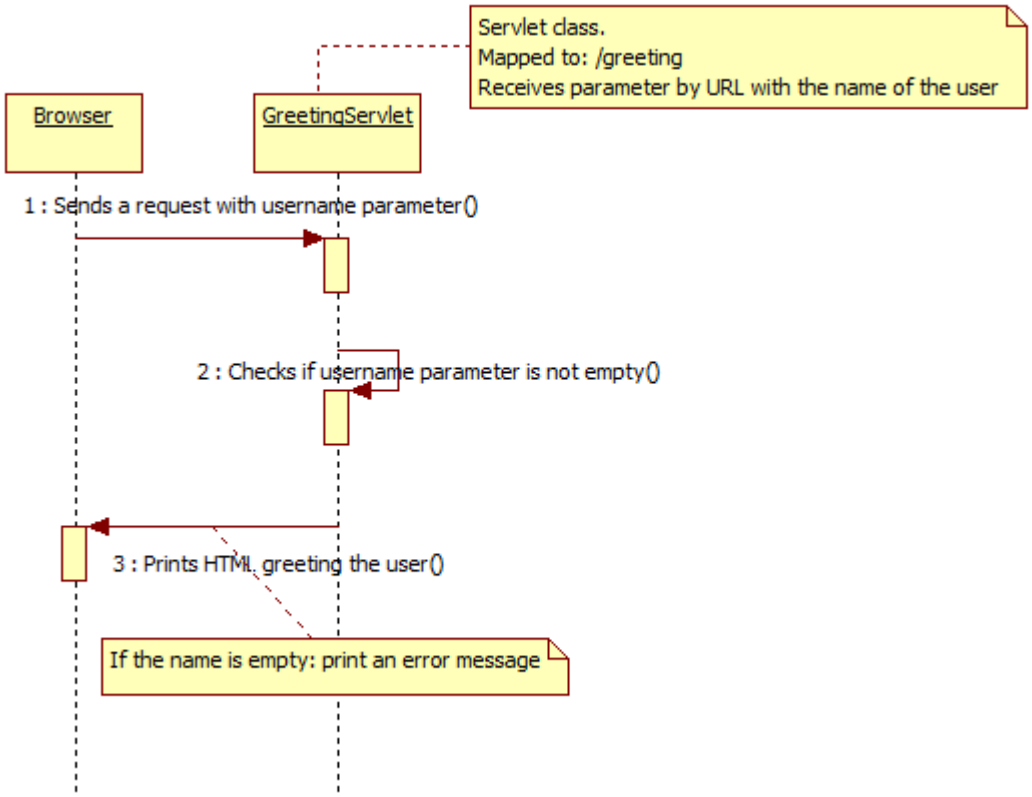
17. Refresh the page and notice how the count increases each time.

Practice 3-2: Creating a Greeting Servlet

Overview

In this practice, you create a simple servlet that processes a URL parameter.

Your application consists of one servlet that will receive a URL parameter, will parse it, and if present, will use it as a name to greet you.



Practice Overview

To create a Greeting Servlet, do the following:

1. Create a new web application project named `lab_03_02` in NetBeans.
2. Create a new servlet named `GreetingServlet` in the `com.lab0302.servlets` package.
3. Map `GreetingServlet` to the URL `/greeting`.
4. In the `GreetingServlet` `processRequest` method, get the username parameter from the request by using the following:

```
String username = request.getParameter("username")
```

5. Check whether the parameter is null or an empty string. If it is empty, print an error message as HTML using the out PrintWriter.
6. If the parameter is not empty, print "HELLO " + USERNAME using the output PrintWriter.

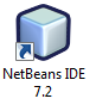
```
if (username == null || "".equals(username.trim())) {
  out.println("<h1>No username given. Who are you?</h1>");
} else {
  out.println("<h1>HELLO " + username + "</h1>");
}
```

7. Delete the `index.jsp` file or modify it to add a link to the servlet.
8. Run the application and navigate in your browser to:

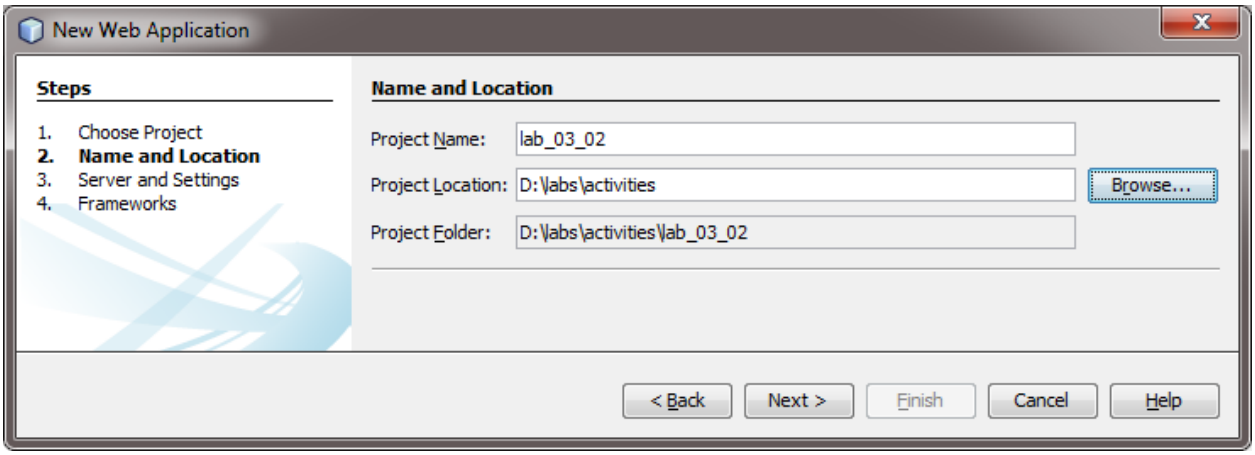
```
http://localhost:7001/lab_03_02/greeting
http://localhost:7001/lab_03_02/greeting?username=Mike
http://localhost:7001/lab_03_02/greeting?username=
http://localhost:7001/lab_03_02/greeting?username=none
```

9. Note what happens with each invocation.
10. Change the parameter and see what happens.

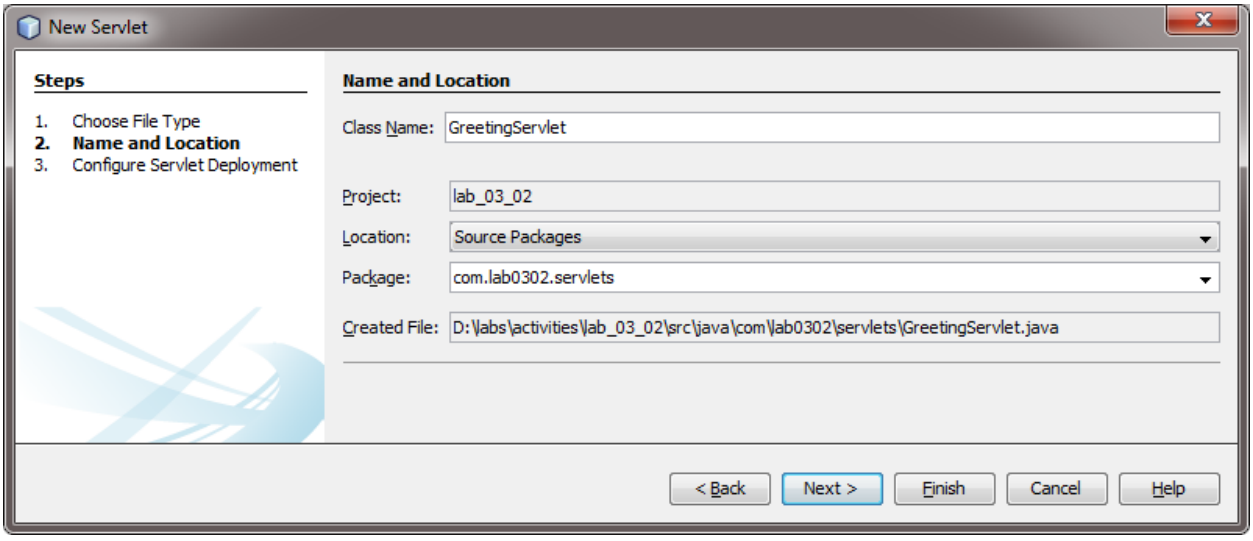
Step-by-Step Instructions



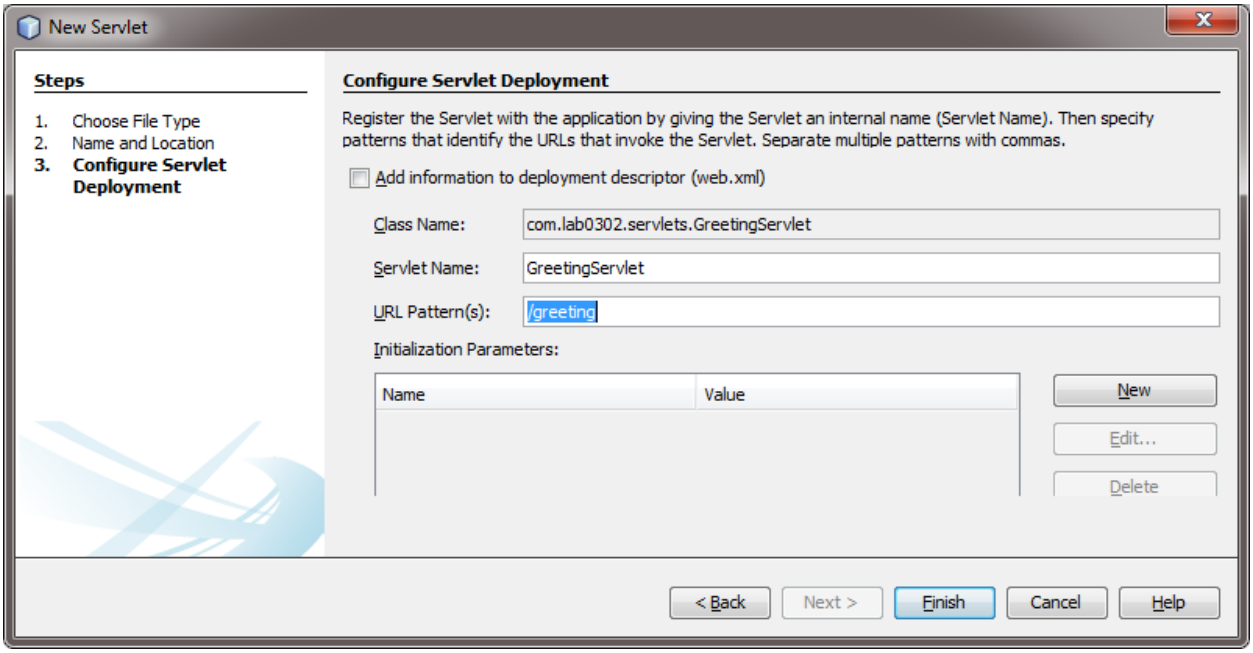
1. Open NetBeans IDE.
2. From the File menu, select New Project, and create a web application project. (For detailed instructions on how to create a web project, see Practice 3-1.)
3. Name the application `lab_03_02` and browse to select `d:\labs\activities` as the project location.



4. Verify you target Oracle's WebLogic server and use Java EE 6 Web. For Frameworks, leave all options unchecked and click Finish.
5. Open the project, right-click Source Packages, and select New > Servlet.
6. Name the servlet `GreetingServlet` and set the package to `com.lab0302.servlets`. Click Next.



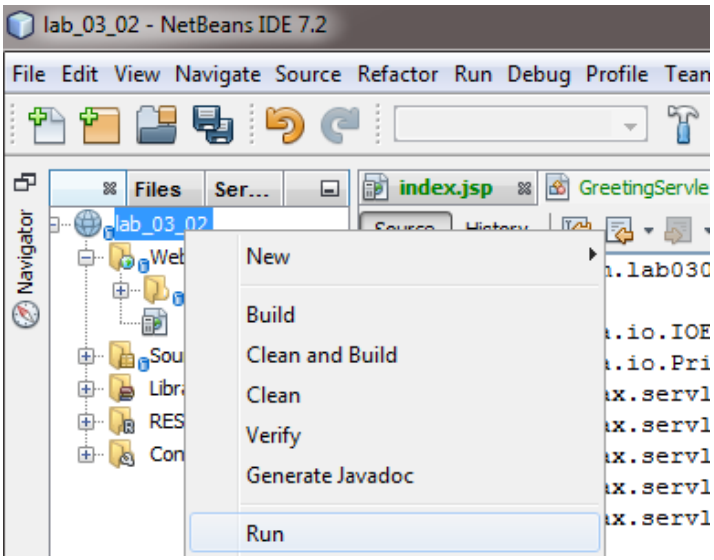
7. Set the URL pattern to `/greeting` and click Finish.



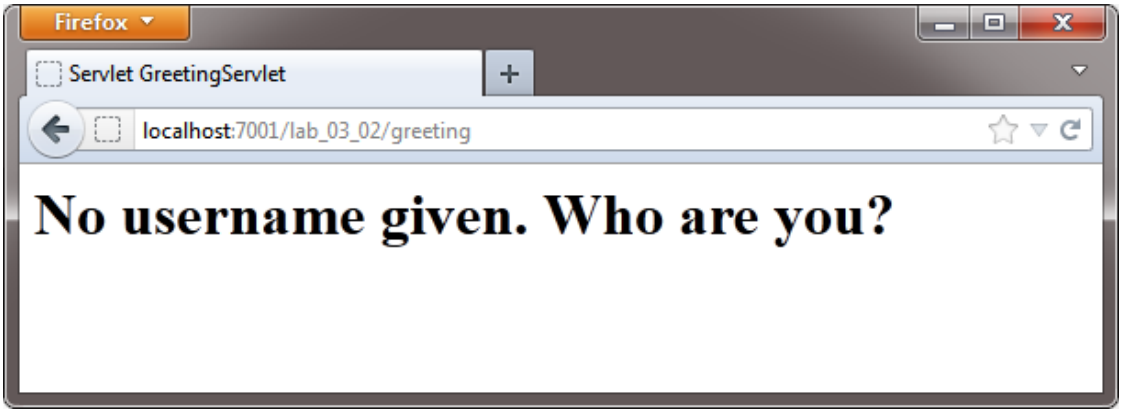
8. Add the highlighted text to the Servlet class to get the username parameter and verify that it is not empty.

```
@WebServlet(name = "GreetingServlet", urlPatterns = {"/greeting"})
public class GreetingServlet extends HttpServlet {

  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      String inputUsername = request.getParameter("username");
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Servlet GreetingServlet</title>");
      out.println("</head>");
      out.println("<body>");
      if (inputUsername == null || "".equals(inputUsername.trim())) {
        out.println("<h1>No username given. Who are you?</h1>");
      } else {
        out.println("<h1>HELLO " + inputUsername + "</h1>");
      }
      out.println("</body>");
      out.println("</html>");
    } finally {
      out.close();
    }
  }
```

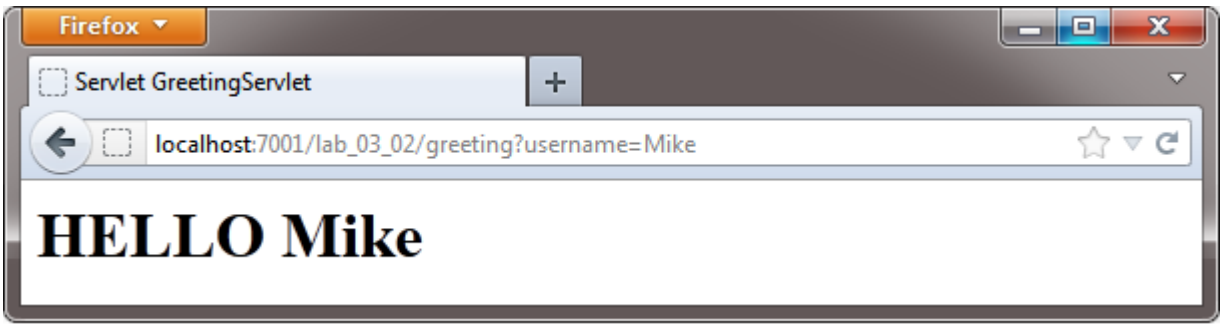
9. Run the project by right-clicking the project and selecting Run.



10. In your web browser, navigate to http://localhost:7001/lab_03_02/greeting.



11. Now navigate to http://localhost:7001/lab_03_02/greeting?username=Mike.



12. Try the following URLs and note the results.

```
http://localhost:7001/lab_03_02/greeting
http://localhost:7001/lab_03_02/greeting?username=
http://localhost:7001/lab_03_02/greeting?username=none
```

Additional Information

You may modify the code for your servlets to add validations of data or do more complex calculations to display data.

Servlets are good for complex calculations, invoking external services, and validating input. As you will see, having HTML output in servlets is complicated and can be difficult to maintain.

Now that you have completed your practices for the lesson, take some time to deploy this lesson's example. In NetBeans open and run the project that is in `D:\labs\examples\lesson03`.

You will see some servlet examples as well as their source code. Explore the examples and take the quiz at the bottom of the web page.

Feel free to explore the example project and pay special attention to the servlets inside the project.

Practices for Lesson 4: Form Processing

Chapter 4

Practices for Lesson 4

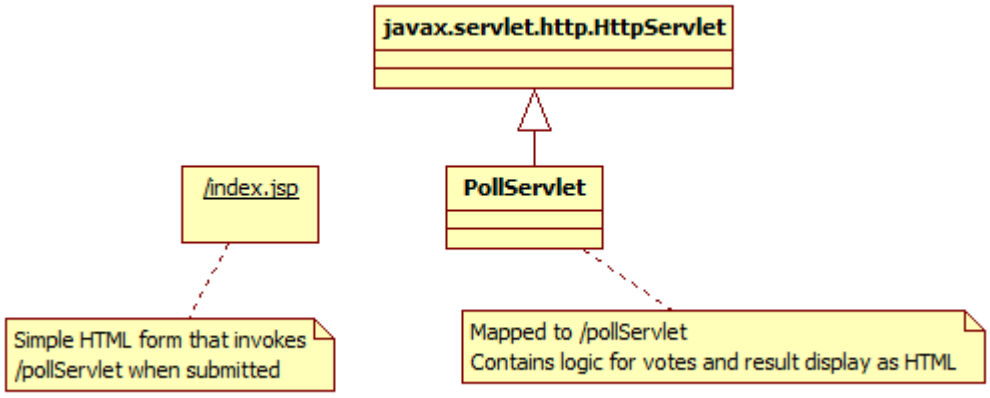
Practices Overview

In these practices, you will use servlets to process form data and display information in a web page. Sessions let you persist information for a user, and this information is used to track the user state and interactions with your application.

Practice 4-1: Creating a Poll Web Application Using Servlets

Overview

In this practice, you create a poll web application using servlets to keep track of the votes on your page. The following diagram describes the navigation in this application:



In this practice, you create a simple HTML form and handle the input parameters in the servlet by using the `request.getParameter` method.

Practice Overview

Tasks

1. Create a new web application in `D:\labs\activities` and name it `lab_04_01`.
2. Add the following HTML form to the `index.jsp` file.

```
<form action="pollServlet" method="POST">
  <h3>What is your favorite color?</h3>
  <input type="radio" name="color" value="red">red<br>
  <input type="radio" name="color" value="blue">blue<br>
  <input type="radio" name="color" value="green">green<br>
  <input type="submit" value="submit">
</form>
```

3. Create a new servlet named `PollServlet` in the `com.lab0401.servlets` package.
4. Map the `PollServlet` to the `/pollServlet` URL.
5. Create a `Map<String, Integer>` in the `PollServlet` to store the votes. The key of the map is the color and the value is the number of votes.

```
private Map<String, Integer> votes = new HashMap<String,
Integer>();
```

6. Get the form parameter using:

```
String votedColor = request.getParameter("color");
```

7. Get the vote count from the votes map and increment it.

```
Integer voteCount = votes.get(votedColor);
if(voteCount==null){
  voteCount=0;
}
voteCount++;
votes.put(votedColor, voteCount);
```

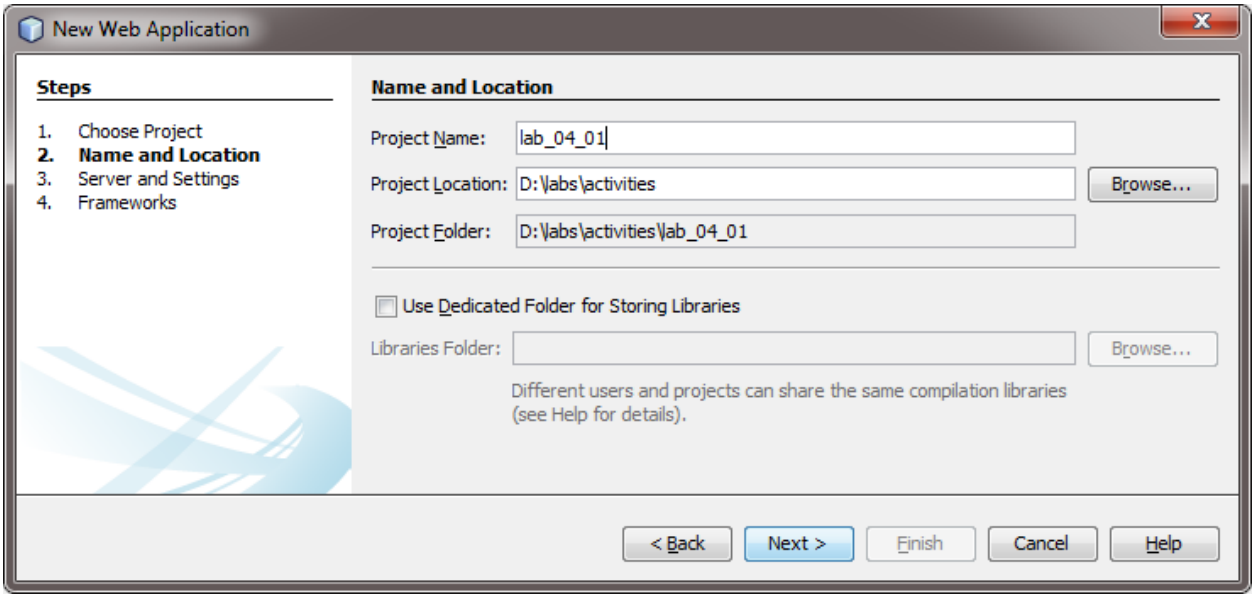
8. Print out all the votes as HTML:

```
out.println("<ul>");
for(Map.Entry<String, Integer> vote : votes.entrySet()){
  out.println( "<li>" + vote.getKey() + ":" + vote.getValue() +
"</li>");
}
out.println("</ul>");
```

9. Run the application and vote on the page for a color.
10. Browse back to `index.jsp` and submit the form again.

Step-by-Step Instructions

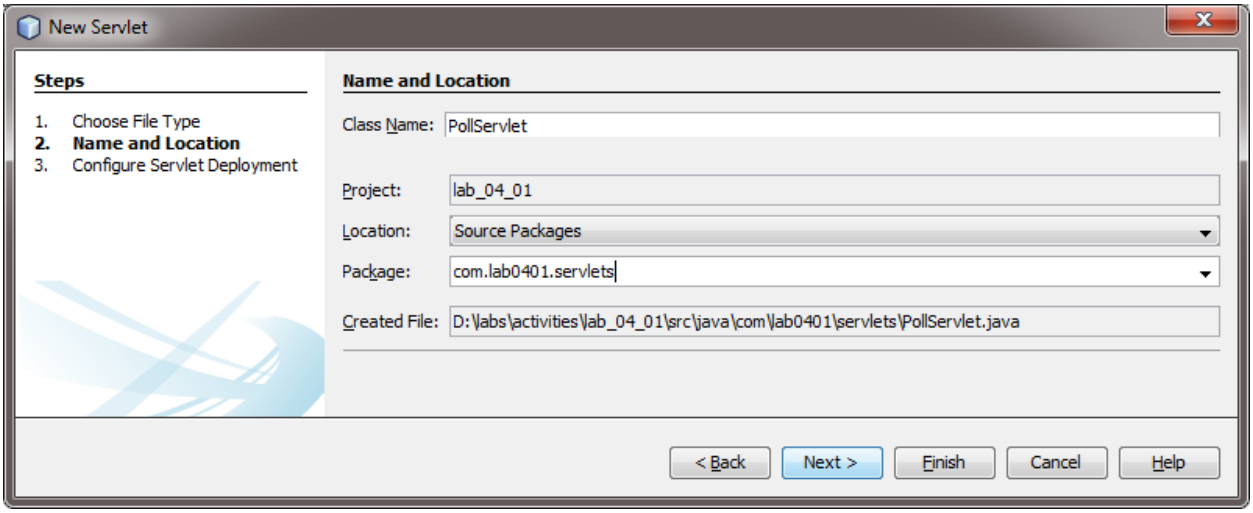
1. Create a new web application project in NetBeans called `lab_04_01` in `D:\labs\activities`.

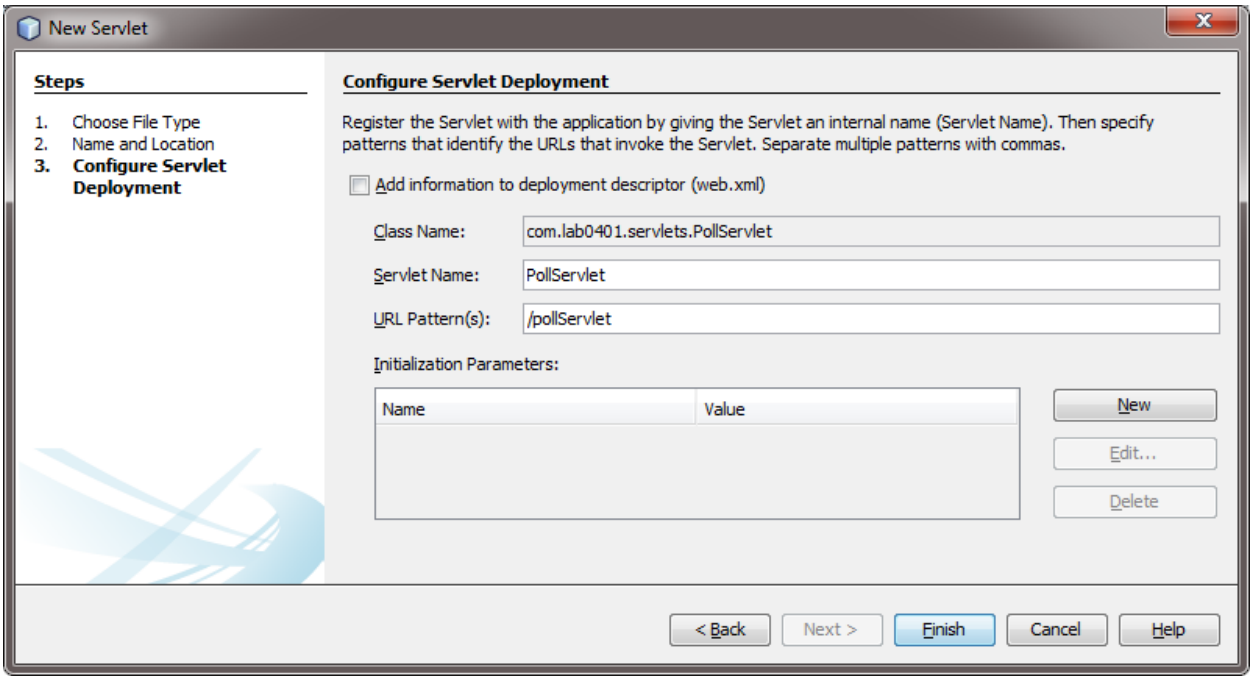


2. Open the `index.jsp` file if it is not already open, add the highlighted HTML code for the form, and modify the title and h1 header:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Poll</title>
  </head>
  <body>
    <h1>Welcome!</h1>
    <form action="pollServlet" method="POST">
      <h3>What is your favorite color?</h3>
      <input type="radio" name="color" value="red">red<br>
      <input type="radio" name="color" value="blue">blue<br>
      <input type="radio" name="color" value="green">green<br>
      <input type="submit" value="submit">
    </form>
  </body>
</html>
```

3. Create a new servlet named `PollServlet` in the `com.lab0401.servlets` package and map it to the `/pollServlet` URL.





Note that the value in the URL Pattern(s) field is **/pollServlet** with a lowercase p. URL mappings are case sensitive.

4. Add the highlighted code to PollServlet to handle the poll voting and display.

```
@WebServlet(name = "PollServlet", urlPatterns = {"/pollServlet"})
public class PollServlet extends HttpServlet {

  private Map<String, Integer> votes = new HashMap<String, Integer>();

  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {

    String votedColor = request.getParameter("color");
    Integer voteCount = votes.get(votedColor);
    if (voteCount == null) {
      voteCount = 0;
    }
    voteCount++;
    votes.put(votedColor, voteCount);

    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Votes</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>Vote results:</h1>");

      out.println("<ul>");
      for (Map.Entry<String, Integer> vote : votes.entrySet()) {
        out.println("<li>" + vote.getKey() + ":" + vote.getValue() + "</li>");
      }
      out.println("</ul>");

      out.println("</body>");
      out.println("</html>");
    } finally {
      out.close();
    }
  }
```

5. Fix any missing imports—to do so, open the Source menu and select Fix Imports (Ctrl + Shift + I).

   You may add the imports manually for classes: `java.util.HashMap` and `java.util.Map`.

6. Right-click the project name in the Projects tab and select Run from the menu to deploy the application in WebLogic.
7. Open a browser and navigate to http://localhost:7001/lab_04_01/index.jsp.
8. Vote for a color by selecting its radio button and clicking on the Submit button.
9. Browse back and vote for another color.

   Note how the form parameter “color” is sent to the servlet and the servlet handles some logic to present the results of the poll.

Practice 4-2: Using Sessions to Track Votes

Overview

In this practice, you will restrict votes to polls based on the session state. You also parse HTML Form parameters to define which poll the user voted on.

In this practice, you begin with an existing project that contains an HTML form and a partially implemented servlet that you must complete to make the voting functionality work and then restrict voting to only non-voted polls.

Practice Overview

For this practice, you will begin with an existing project and you will modify the files in it.

1. Open the `lab_04_02` in `D:\labs\activities` project.
2. Open the `com.lab0402.servlets.PollServlet` Java class.
3. In this class, you already have several methods that you will use to vote and control polls.

Method	Description	Parameters
`vote(pollName, option)`	Adds a vote to the option of a given poll	`pollName`: The name of the poll `option`: The selected option in the poll
`printSuccess(response, pollName)`	Prints a vote success message and the poll results in the response	`response`: The `HttpServletResponse` object `pollName`: The name of the poll
`printFail(response, pollName)`	Prints a vote failed message and the poll results in the response	`response`: The `HttpServletResponse` object `pollName`: The name of the poll

4. In the `doPost` method, get the HTTP form parameters from the response:

```
String pollName = request.getParameter("pollName");
String option = request.getParameter("option");
```

5. Get the session from the request.

```
HttpSession session = request.getSession();
```

6. Get the `voted` attribute from the session:

```
Boolean voted = (Boolean)session.getAttribute(pollName);
```

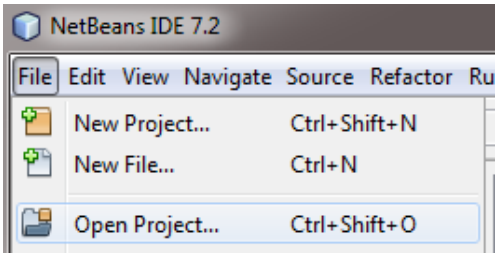
7. If `voted` is `true`, call the `printFail` method.
8. If `voted` is equal to `null` or `false`, call the `vote` method immediately after the `printSuccess` method.
9. Set the `voted` attribute to `true` in the session:

```
session.setAttribute(pollName, true);
```

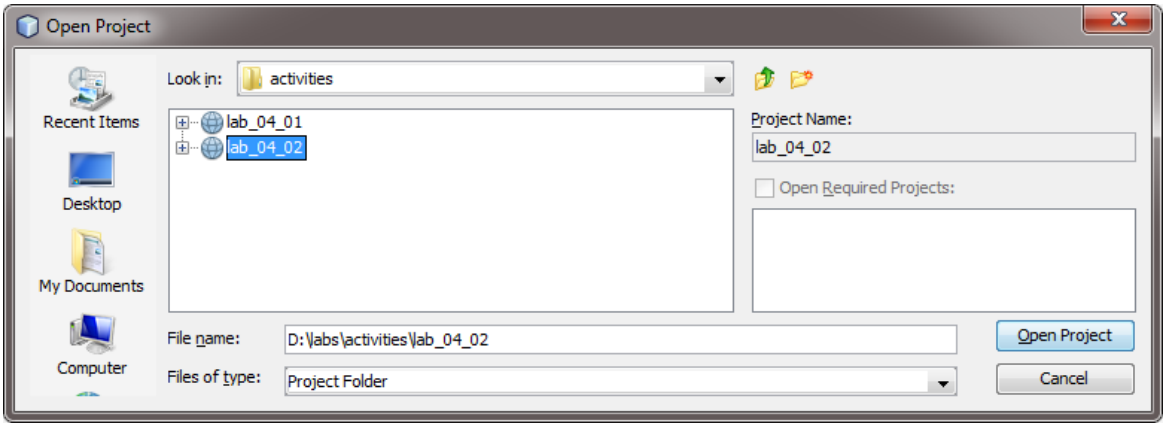
10. Test your application by deploying it in WebLogic and voting in the polls.
11. Validate the poll restriction by voting twice in a poll, you should get an error message the second time.
12. Validate the session restriction by voting in a poll, closing your browser, opening it again, and voting in the same poll.

Step-by-Step Instructions

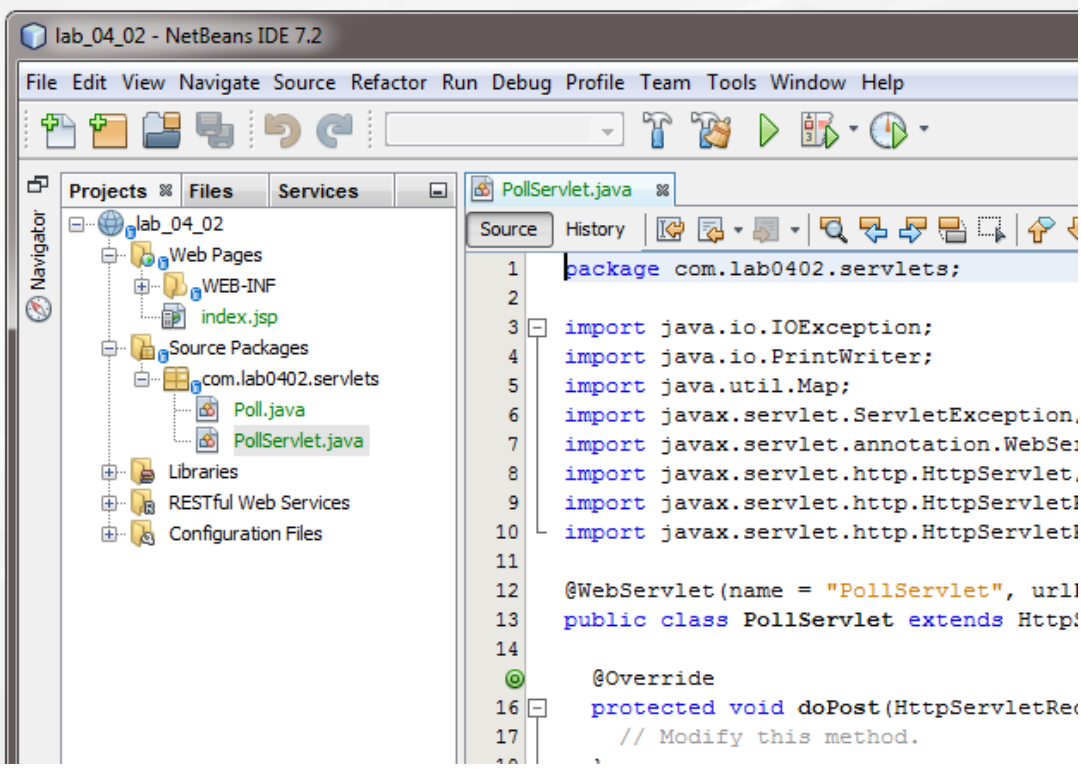
1. In NetBeans, select File > Open Project.



2. Browse to `d:\labs\activities\lab_04_02` and click Open Project.



3. Open the `com.lab0402.servlets.PollServlet` class.



4. Parse the requests parameters. Go to the `doPost` method and add the following code:

```
  String pollName = request.getParameter("pollName");
  String option = request.getParameter("option");
```

5. Get the session from the request.

```
HttpSession session = request.getSession();
```

6. Add the session validation to verify that the user has not already voted in the poll.

```
Boolean voted = (Boolean)session.getAttribute(pollName);
if(voted != null && voted == true){
  //User already voted.
}else{
  //User has not voted.
}
```

7. Add the code for the fail message for when the user has already voted.

```
//User already voted.
printFail(response, pollName);
```

8. Add the code to `vote`, store the `voted` status in the session, and finally print the success message.

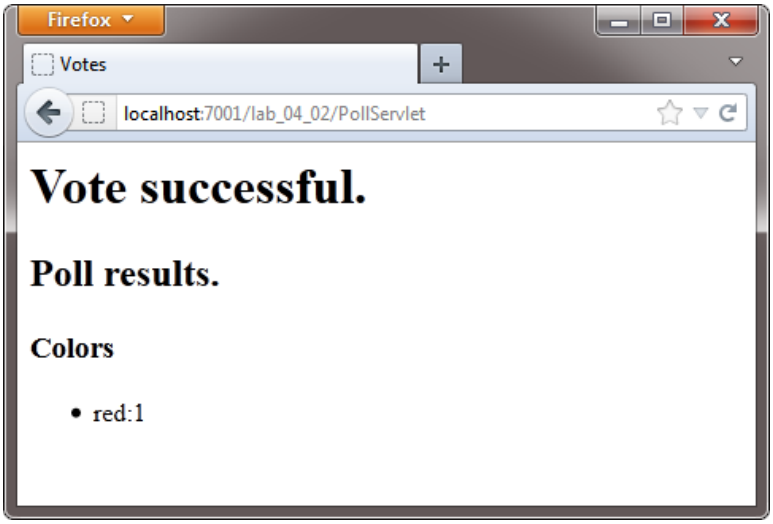
```
//User has not voted.
vote(pollName, option);
session.setAttribute(pollName, true);
printSuccess(response, pollName);
```

Use the following code for reference:

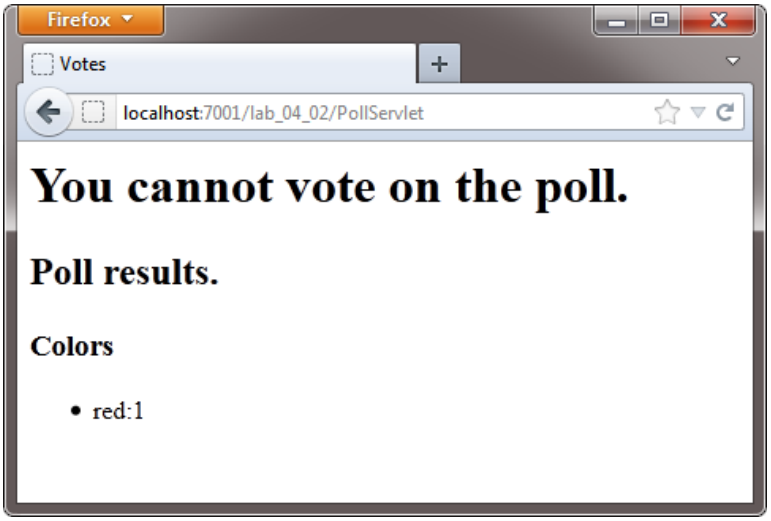
```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException
  String pollName = request.getParameter("pollName");
  String option = request.getParameter("option");

  HttpSession session = request.getSession();

  Boolean voted = (Boolean)session.getAttribute(pollName);
  if(voted != null && voted == true){
    //User already voted.
    printFail(response, pollName);
  }else{
    //User has not voted.
    vote(pollName, option);
    session.setAttribute(pollName, true);
    printSuccess(response, pollName);
  }

}
```

9. Right-click the project name and select Run. Your browser opens at http://localhost:7001/lab_04_02/ displaying a page with the current polls.

10. Vote in a poll. Note that you get the poll results and your vote has been cast.



11. Click the browser's Back button and try to vote again in the same poll. You see an error page and the vote count is not altered this time.



12. Because the voting is restricted per session, you can start a new session by closing the browser and opening it again.
13. Close the browser and open it again and go to the URL http://localhost:7001/lab_04_02/ to test this restriction.

Additional Information

In these practices, you implemented a servlet that handles input from a form and handled stored values in a session to validate user votes.

Take some time to review the code for Practice 4-2 and focus on how the vote validation uses a session.

- What other ways to restrict user voting do you think there are?
- How would you implement it?

Remember that you can open this lesson's example project by using NetBeans. In the example project, you will find additional source code and the quiz for this lesson. The project is located in `D:\labs\examples\lesson04`.

Practices for Lesson 5: Web Application Configuration

Chapter 5

Practices for Lesson 5

Practices Overview

In these practices, you will configure a web application using the `web.xml` deployment descriptor and Java EE annotations.

Practice 5-1: Configuring a Web Application

Overview

In this practice, you will configure an existing application. The application you configure is the poll application that you have been working on in previous lessons. You must configure it correctly in the `web.xml` file and using Java EE annotations. The application has the following features and configurations.

Features and Configurations

- Welcome page: `home.html`
- Session timeout: `30 minutes`
- Context parameter: `appName=Polls`
- Servlets:
  - Servlet name: `VoteServlet`
    - Servlet class: `com.lab0501.servlets.VoteServlet`
    - URL mapping: `/Vote`
  - Servlet name: `ResultServlet`
    - Servlet class: `com.lab0501.servlets.ResultServlet`
    - URL mapping: `/Result`
    - Parameter: `title=Poll Results`
  - Servlet name: `PollServlet`
    - Servlet class: `com.lab0501.servlets.PollServlet`
    - URL mapping: `/Poll`

Practice Overview

1. Open the project in `D:\labs\activities\lab_05_01`.
2. Configure the following basic `web.xml` features:
   - Session timeout:

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

   - Welcome file:

```
<welcome-file-list>
  <welcome-file>home.html</welcome-file>
</welcome-file-list>
```

   - Context initialization parameter:

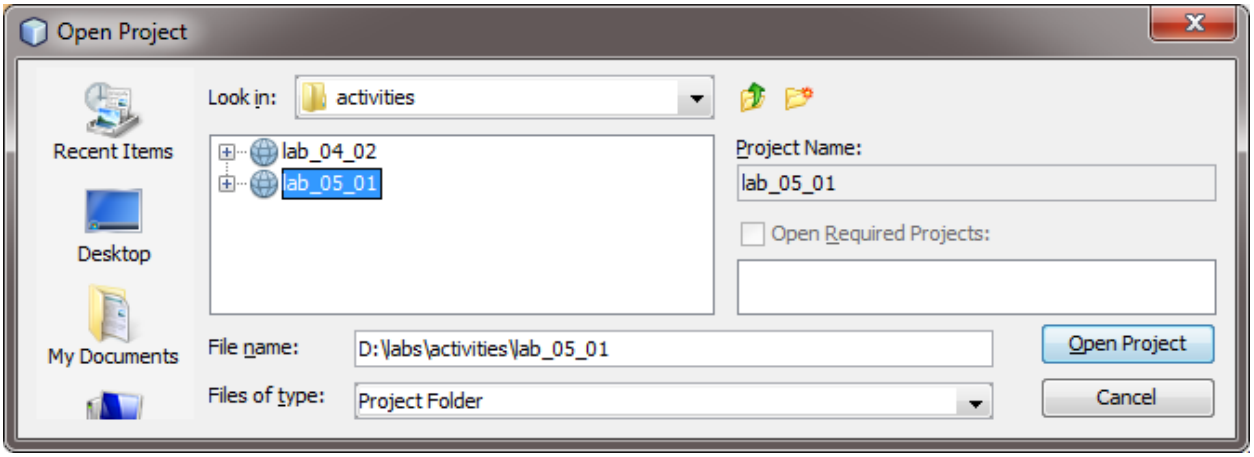
```
<context-param>
  <param-name>appName</param-name>
  <param-value>Polls</param-value>
</context-param>
```

3. From here on you can either configure the rest in the `web.xml` file or use annotations. This practice will use annotations.

   To see the configuration that uses the `web.xml`, go to your solutions folder at `d:\labs\solutions\lab_05_01\xml`, where you will find the `web.xml` file with all the required configurations.
4. Add the `@WebListener` annotation to `com.lab0501.listener.PollInitListener`.
5. Add the `@WebServlet` annotation to the `com.lab0501.servlets.VoteServlet` and `com.lab0501.servlets.PollServlet` classes. Use the "Features and Configurations" list on the previous page for configuration details.
6. Add the `@WebServlet` annotation to the `com.lab0501.servlets.ResultServlet` class. Then add, in the `initParams` attribute of the annotation, a `@WebInitParam` element to the class (use an array of one element), using the configuration outlined on the previous page,

Step-by-Step Instructions

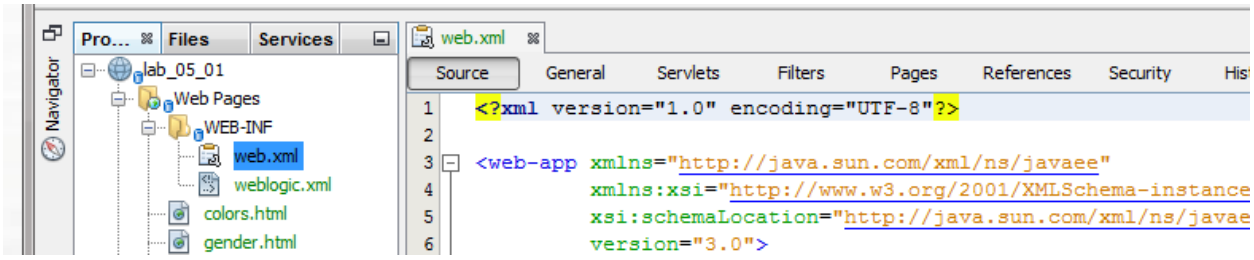
1. Open the project.
2. In NetBeans, select File > Open Project.
3. Browse to `D:\labs\activities\` and open the `lab_05_01` project.



4. Run the application. Note that the application lacks any functionality.

   The application already contains all the code necessary to work, but because none of the servlets or listeners is configured, the application fails to do anything. As you add elements to the `web.xml` file and annotations to the Java classes, run the application to note how functionality is gradually restored. You can tweak the Context and Initialization parameters.

5. From the projects tab, expand Web Pages, expand WEB-INF, and open the `web.xml` file.



6. Configure session timeouts by adding the following highlighted code to the `web.xml` file. Remember to add it between the `web-app` tags.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">

  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>

</web-app>
```

7. To configure the web application's welcome file(s) that will be served when a user access a folder, add the following highlighted code below the `session-config` element. Remember that multiple welcome-file elements are allowed.

```
  </session-config>

  <welcome-file-list>
    <welcome-file>home.html</welcome-file>
  </welcome-file-list>

</web-app>
```

8. Add the following highlighted code below the `welcome-file-list` element to add the required Context parameters.

```
  </welcome-file-list>

  <context-param>
    <param-name>appName</param-name>
    <param-value>Polls</param-value>
  </context-param>

</web-app>
```

The complete `web.xml` file should look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">

  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>

  <welcome-file-list>
    <welcome-file>home.html</welcome-file>
  </welcome-file-list>

  <context-param>
    <param-name>appName</param-name>
    <param-value>Polls</param-value>
  </context-param>

</web-app>
```

9. Save the file.
10. This application contains one servlet context listener that you have to configure. To do so, open the file: Source Packages > `com.lab0501.listener` > `PollInitListener`.

11. The file is missing the `@WebListener` annotation. Add the highlighted code.

```
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

@javax.servlet.annotation.WebListener
public class PollInitListener implements ServletContextListener {

  @Override
  public void contextInitialized(ServletContextEvent sce) {
```

Alternatively, you can import `javax.servlet.annotation.WebListener` and use just the `@WebListener` to annotate the class:

```
import javax.servlet.annotation.WebListener;

@WebListener
```

The application contains three servlets. You will configure all of them using annotations.

12. Open the `com.lab0501.servlets.PollServlet` class and add the `WebServlet` annotation.

```
@javax.servlet.annotation.WebServlet(name="PollServlet", urlPatterns={"/Poll"})
public class PollServlet extends HttpServlet {
```

13. Open the `com.lab0501.servlets.VoteServlet` class and add the `WebServlet` annotation.

```
@javax.servlet.annotation.WebServlet(name="VoteServlet", urlPatterns={"/Vote"})
public class VoteServlet extends HttpServlet {
```

14. Finally open the `com.lab0501.servlets.ResultServlet` class. This class requires a `WebServlet` annotation with an initialization parameter. Configure it by adding the annotation like this:

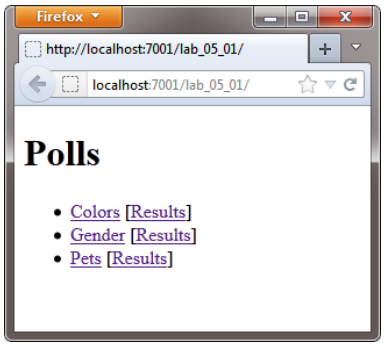
```
@javax.servlet.annotation.WebServlet(
  name="ResultServlet",
  urlPatterns={"/Result"},
  initParams={
      @javax.servlet.annotation.WebInitParam(name="title", value="Poll Results")
    }
  )
public class ResultServlet extends HttpServlet {
```

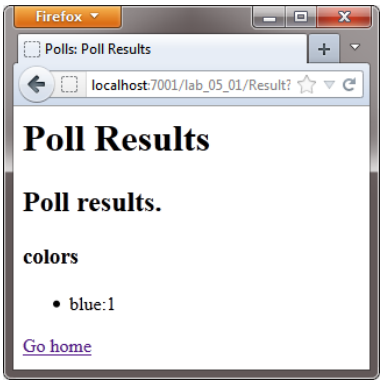
The preceding code can be in a single line; however, if the line is too long, it is always a good idea to split it so that the code is more readable.

15. Save all the files and run your application.
16. Right-click the `lab_05_01` project and select Run.

Your browser will open and your application should be configured and working.

17. To test the application's configuration, vote in a poll and see the results in the browser window.





Practice 5-2: Challenge Practice: `web.xml`-Only Configuration

Overview

In the previous practice, you reviewed how to configure a web application by using annotations. This application can be configured without annotations, by using only the `web.xml` file.

Assumptions

The Practice 5-1 is completed.

Tasks

1. Remove the `@WebServlet` and `@WebListener` annotations from the servlets and listener:
   - `com.lab0501.servlets.PollServlet`
   - `com.lab0501.servlets.VoteServlet`
   - `com.lab0501.servlets.ResultServlet`
   - `com.lab0501.listener.PollInitListener`
2. Run the application. You should not have any functionality available from the servlets in the application.
3. Configure the listener by adding the following to the `web.xml` file:

```
<listener>
  <listener-class>com.lab0501.listener.PollInitListener</listener-
class>
</listener>
```

4. Configure Poll servlet by adding the following to the web.xml file:

```
<servlet>
  <servlet-name>PollServlet</servlet-name>
  <servlet-class>com.lab0501.servlets.PollServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>PollServlet</servlet-name>
  <url-pattern>/Poll</url-pattern>
</servlet-mapping>
```

5. The Vote servlet uses a similar configuration. Using the Poll servlet configuration, try to figure out the configuration for the Vote servlet and add it to the file. If you require further reference, use the `D:\labs\solutions\lab_05_02\web.xml` file.
6. For the Result servlet, configure it in the same way you did with the Poll and Vote servlets and add the following to the servlet element to add an initialization parameter to it:

```
    <init-param>
      <param-name>title</param-name>
      <param-value>Poll Results</param-value>
    </init-param>
```

7. Run the application. All functionality is restored and configured in the `web.xml` file.

Additional Information

Remember that you can open this lesson's example project by using NetBeans. In the example project, you will find additional source code and the quiz for this lesson. The project is located in `D:\labs\examples\lesson05`.

Practices for Lesson 6: Implementing Model-View-Controller

Chapter 6

Practices for Lesson 6

Practices Overview

In the practice for this lesson, you will implement a servlet as a controller. Servlets are good for processing parameters and delegating work to services. In this practice, a JSP will handle the print logic. You will use the servlet controller to send data to the view and let the JSP print out the page to the user.

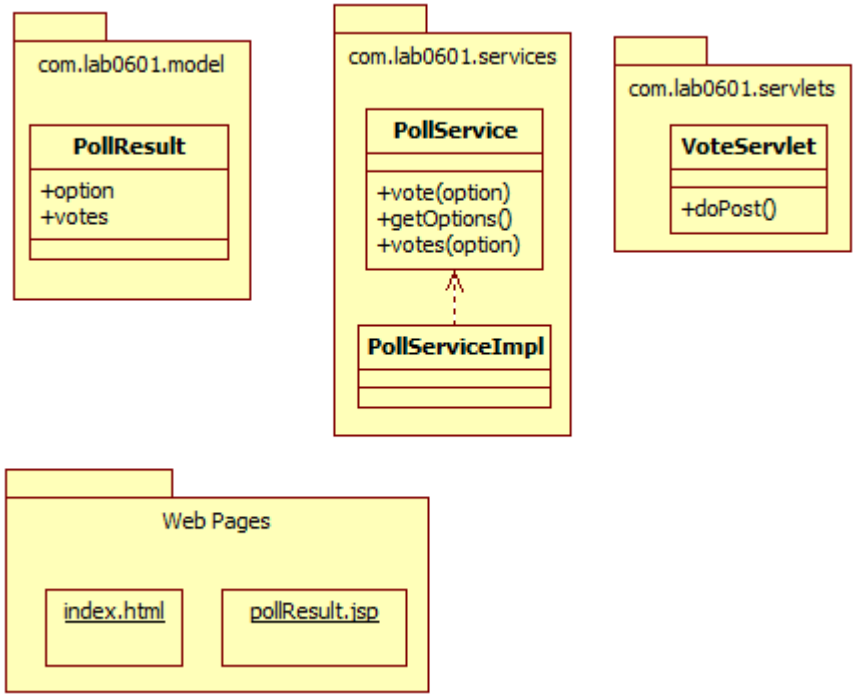
Practice 6-1: Implementing a Servlet Controller

Overview

In this practice, you implement a Servlet Controller to complete an MVC application. You begin with an existing project where the model is the PollResult class, the view is a JSP page, and the service is the PollService interface. You implement the VoteServlet class to use the service to get the model and then dispatch the view.

You start with an existing web project, and the application already has some classes and pages. The following diagram is an overview of the existing classes and pages in the web project and application.

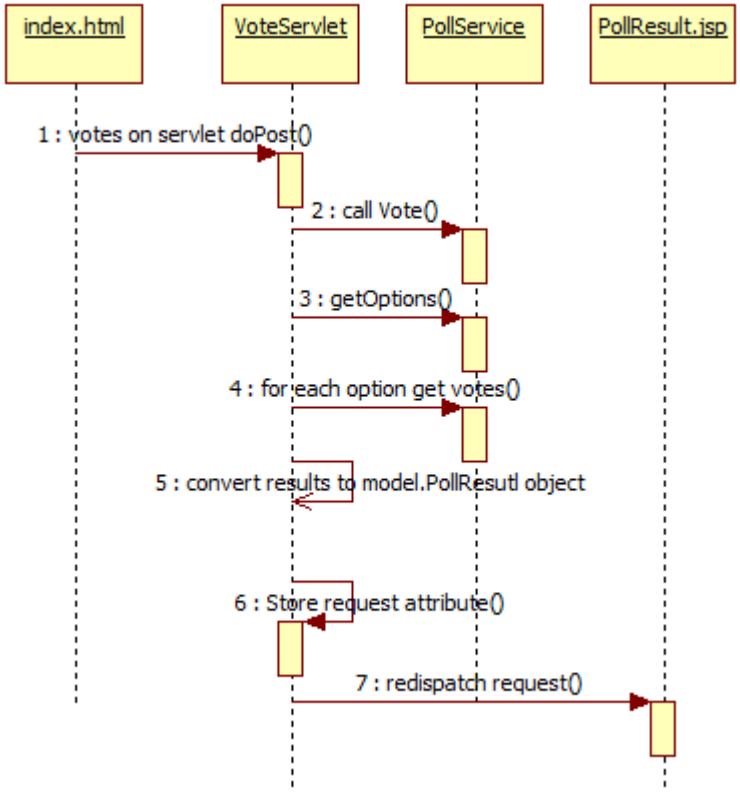
The Poll Project for this Practice (lab_06_01)



The PollResult object is a simple Java Bean that you will create to store the results of the poll and send a collection of the PollResult objects to the JSP view inside a request attribute

You will need to implement the VoteServlet to cast a vote using the service, transform the options and votes into a list of PollResults, store the list in a request attribute, and finally redispatch the request to the JSP view.

The following sequence diagram outlines this process.



Practice Overview

1. Open the `lab_06_01` project in `d:\labs\activities`.
2. Open the `com.lab0601.servlets.VoteServlet` class.
3. You need to implement this servlet as a controller.
4. Inject the `PollService` class.
5. To inject the `PollService` class add the following code as an instance variable:

```
@Inject
private PollService pollService;
```

   * You need to add the `javax.inject.Inject` and `com.lab0601.services.PollService` imports.

6. Modify the `doPost` method.
7. Get the user vote from the request parameters.

```
String selectedOption = request.getParameter("option");
```

8. Call the `service.vote` method to cast a vote.

```
pollService.vote(selectedOption);
```

9. Convert the results of the polls to a `List`.

```
List<PollResult> results = new ArrayList<PollResult>();
for (String option : pollService.getOptions()) {
  int votes = pollService.getVotes(option);
  PollResult result = new PollResult(option, votes);
  results.add(result);
}
```

   *You need to add the `java.util.List`, `java.util.ArrayList`, and `com.lab0601.model.PollResult` imports.

10. Set the results list inside the `pollResults` request attribute.

```
request.setAttribute("pollResults", results);
```

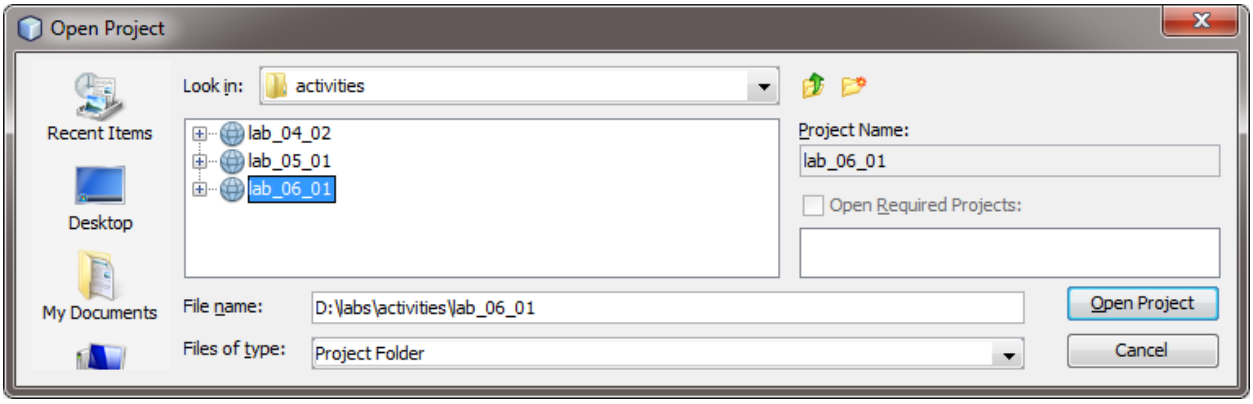
11. Redispatch the request to the `pollResult.jsp` page.

```
request.getRequestDispatcher("pollResult.jsp").forward(request,
response);
```

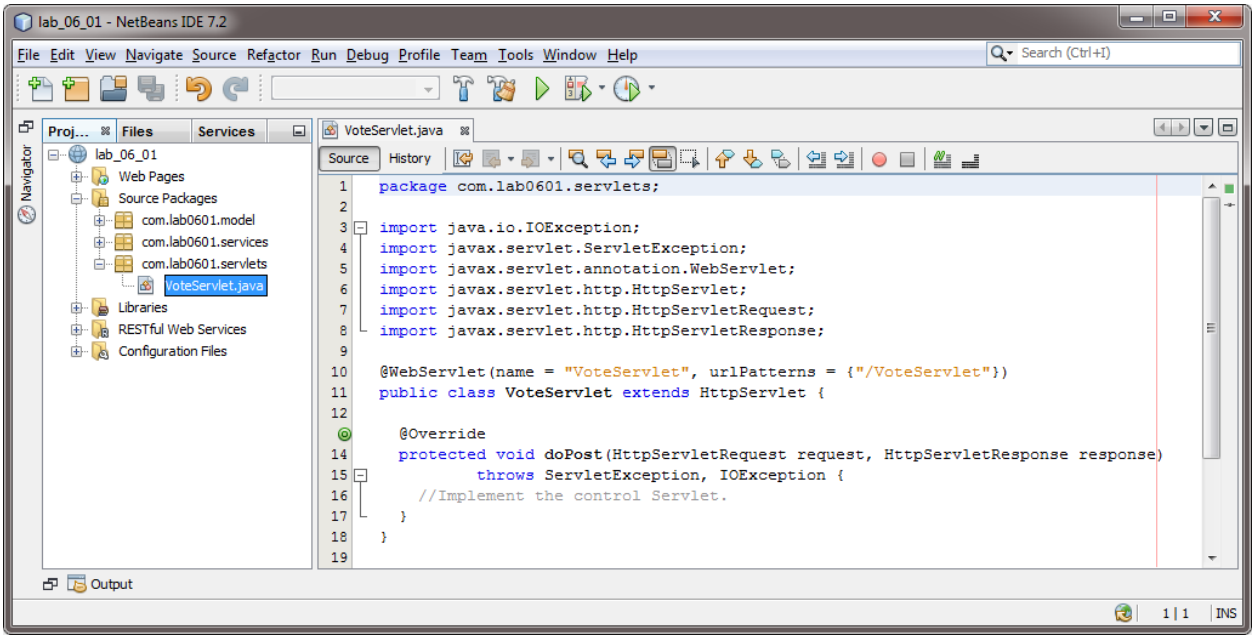
12. Run your application and test the poll voting.

Step-by-Step Instructions

1. Open NetBeans IDE.
2. Select File > Open Project.
3. Browse to `d:\labs\activities` and open the `lab_06_01` project.



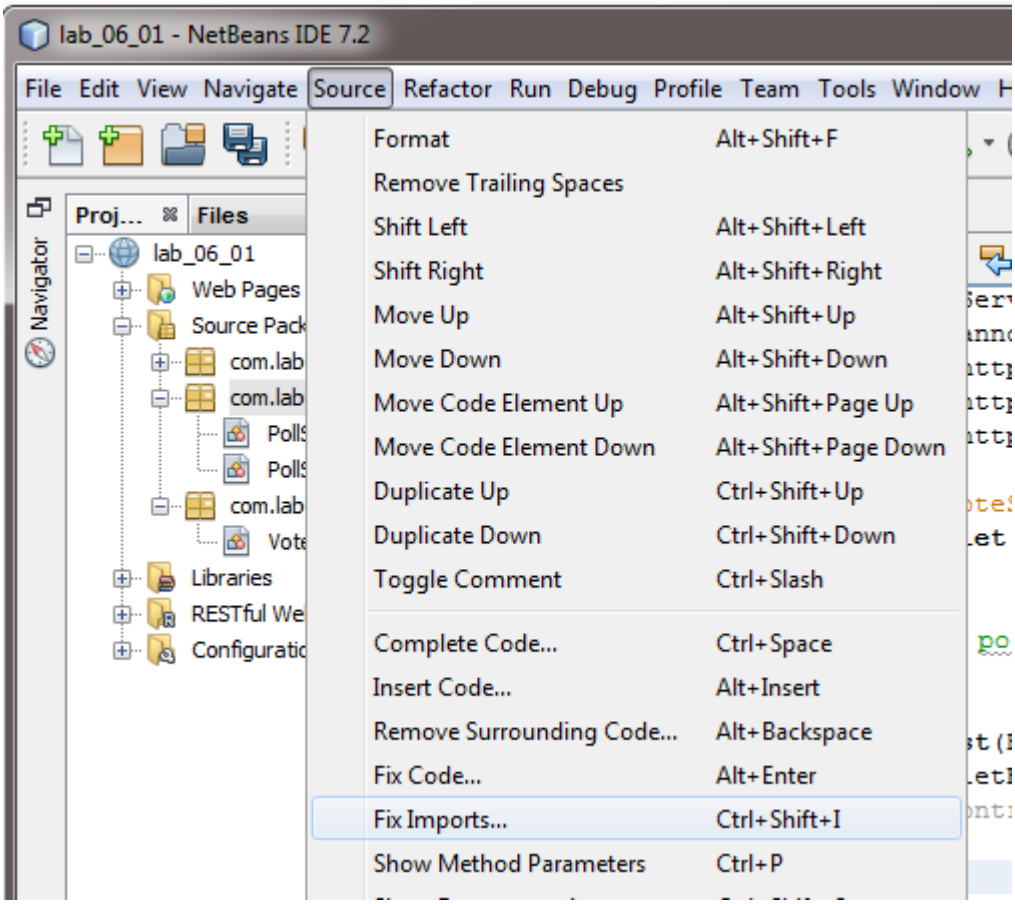
4. Open `com.lab0601.servlets.VoteServlet`.



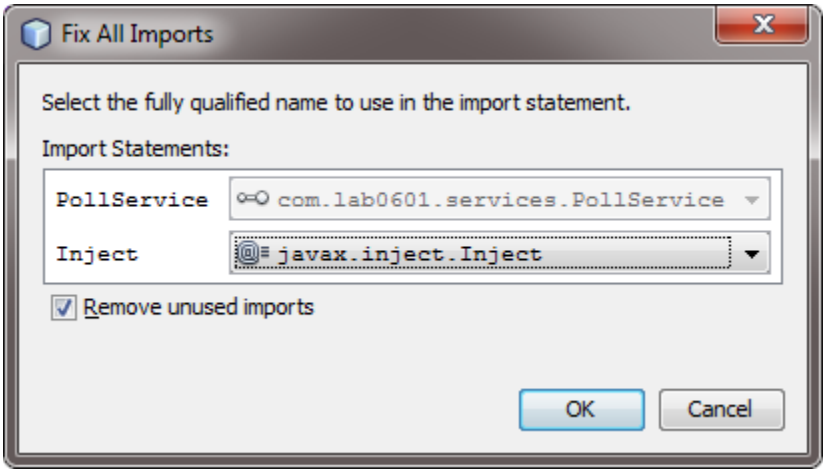
5. Add the highlighted code to enable injection for the `PollService` class.

```
@WebServlet(name = "VoteServlet", urlPatterns = {"/VoteServlet"})
public class VoteServlet extends HttpServlet {

  @Inject
  private PollService pollService;

  @Override
  protected void doPost(HttpServletRequest request, HttpServletResponse response
          throws ServletException, IOException {
    //Implement the control Servlet.
```

6. You will likely get compile errors, because imports are missing. To fix the missing imports, select Fix Imports from the Source menu (or press Ctrl + Shift + I).



7. Select the following classes to resolve the proper imports.



8. Open the `com.lab0601.services.PollService`. Note the methods available for poll manipulation:

Method	Description
`void vote(String option)`	Vote for an option.
`String[] getOptions()`	Get all the available options for the poll.
`int getVotes(String option)`	Get the number of votes for the option.

9. Go back to the `com.lab0601.servlets.VoteServlet` class and, to get the user's vote from the request parameter, add the highlighted code:

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

  String selectedOption = request.getParameter("option");

}
```

10. Call the vote method on the service:

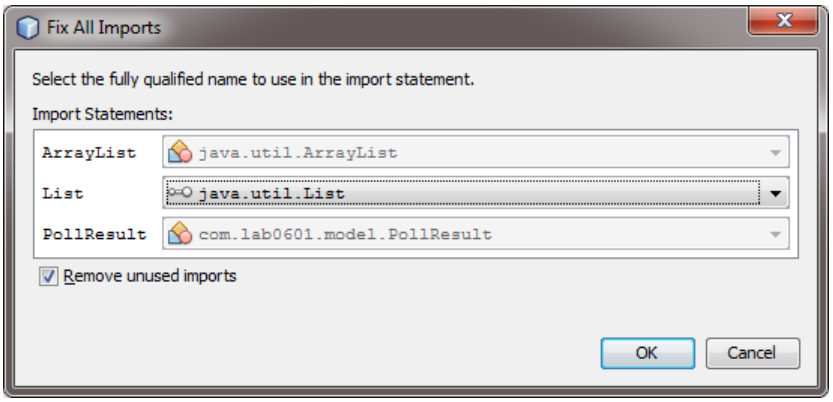
```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

  String selectedOption = request.getParameter("option");

  pollService.vote(selectedOption);

}
```

If you run the application at this point, you will be able to cast votes but you cannot see the results of the poll. The results are rendered by the `pollResult.jsp`, this page requires the results to be in a `List` of `PollResult` objects stored in the `pollResults` request attribute.

11. Convert the Poll result data from the service to a `List` of `PollResult` model objects.

```
  pollService.vote(selectedOption);

  List<PollResult> results = new ArrayList<PollResult>();
  for (String option : pollService.getOptions()) {
    int votes = pollService.getVotes(option);
    PollResult result = new PollResult(option, votes);
    results.add(result);
  }

}
```

12. Fix the missing imports by using Source > Fix Imports.



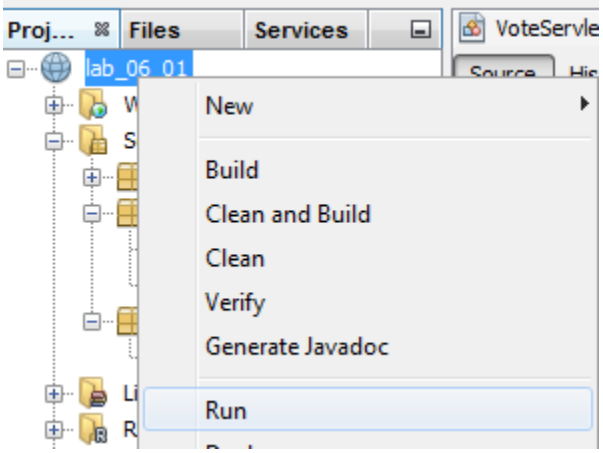
13. The `pollResult.jsp` page requires getting the `pollResults` request attribute and then dispatching the request to the page. To accomplish this, add the highlighted code:

```
    List<PollResult> results = new ArrayList<PollResult>();
    for (String option : pollService.getOptions()) {
      int votes = pollService.getVotes(option);
      PollResult result = new PollResult(option, votes);
      results.add(result);
    }

    request.setAttribute("pollResults", results);

    request.getRequestDispatcher("pollResult.jsp").forward(request, response);
}
```

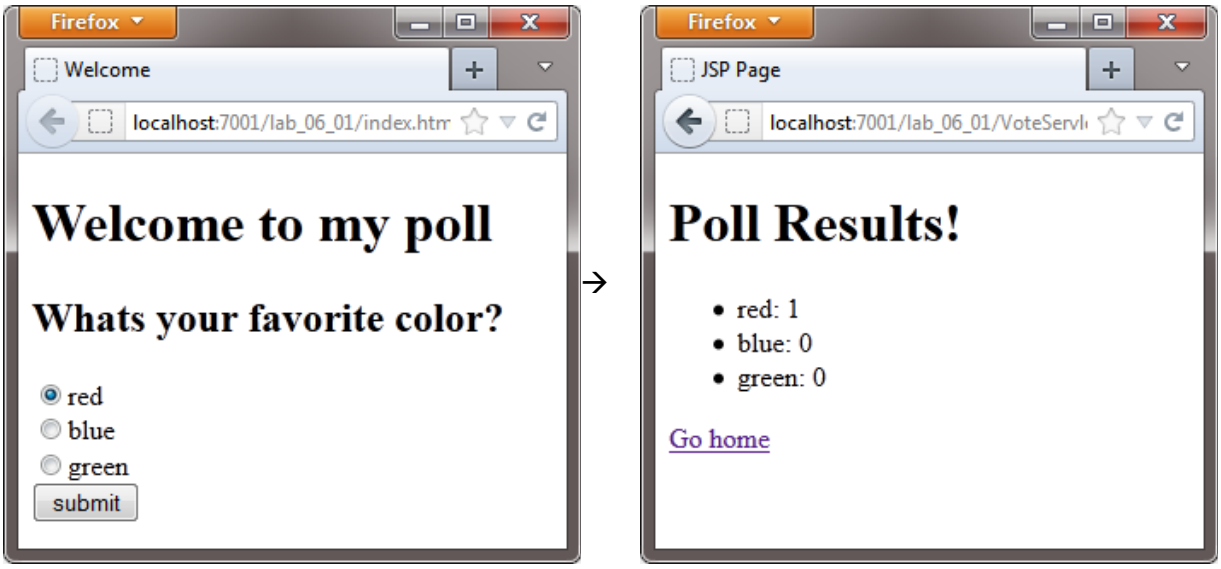
The final code for the servlet (without imports):

```
@WebServlet(name = "VoteServlet", urlPatterns = {"/VoteServlet"})
public class VoteServlet extends HttpServlet {

  @Inject
  private PollService pollService;

  @Override
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {

    String selectedOption = request.getParameter("option");

    pollService.vote(selectedOption);

    List<PollResult> results = new ArrayList<PollResult>();
    for (String option : pollService.getOptions()) {
      int votes = pollService.getVotes(option);
      PollResult result = new PollResult(option, votes);
      results.add(result);
    }

    request.setAttribute("pollResults", results);

    request.getRequestDispatcher("pollResult.jsp").forward(request, response);

  }
}
```

14. Run the application. Right-click the project name and select Run.



15. Test your application by casting a vote. You should see your vote in the result page.



Additional Information

In this practice, you implemented a servlet controller that creates model objects for the view by using CDI injected Services and that leaves the HTML output to a JSP page.

You may be a little confused in this lesson because you used a JSP for the view; however, the important part of this practice is that you understand the MVC pattern and how it allows you to separate the Java code from the HTML code and use CDI to further separate the Java Controller code and the Java service code.

In the following lessons and practices, you will be learning about using JSP as your view technology.

What advantages do you think CDI has over implementing everything in the servlet?

What are the differences between `request→redispatch` and `response→sendRedirect`?

Remember that you can open more examples related to this practice and lesson by using NetBeans. In the example project, you will find additional source code and quiz questions. The project is located in `D:\labs\examples\lesson06`.

Practices for Lesson 7: Implementing JSP

Chapter 7

Practices for Lesson 7

Practices Overview

In these practices, you will implement a JSP to render HTML pages with data set up by servlets.

Practice 7-1: Implementing a JSP view

Overview

In this practice, you create a JSP view using Java scriptlets.

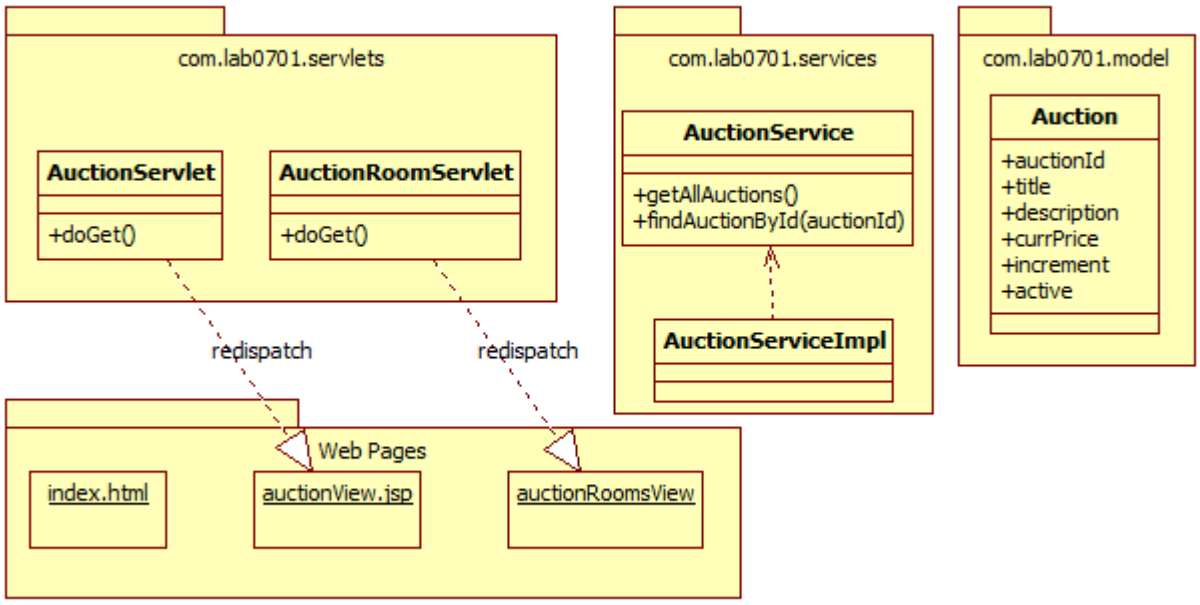
You begin this practice with an existing project and build the required JSP files to display dynamic data in your web browser. You start with an existing project that outlines an auctioning application. The next practices use the same application with incrementally more features.

The Auction Project for This Practice (lab_07_01)

The auction project for this lesson displays only some details on the available auctions.

The architecture for the project follows the same four tiers that the previous lesson showcased (Model, View, Controller, and Service).

The classes and web pages are outlined in the following diagram:



The service layer allows you to get a list of the auctions and find auctions by ID.

Your objective for this lesson is to use the `AuctionServlet` to get the selected auction and redispatching to the `auctionView.jsp` view. Once you do that, you will modify the JSP view to include the data from the Servlet.

The auction list in `auctionRoomsView.jsp` is *not* dynamic and it is hardcoded in the JSP file.

This lesson includes a challenge at the end where you have to make the `auctionRoomsView.jsp` dynamic by using the `AuctionRoomServlet`. You are given only overview instructions on how to complete the challenge. Be sure to exchange your experiences in this challenge with the instructor and other attendees.

Practice Overview

1. Open the `lab_07_01` project at `d:\labs\activities`.
2. Open the `com.lab0701.servlets.AuctionServlet` class and add the `AuctionService` object by using CDI `Inject` annotation.
3. Add the code to get the selected auction ID from the request parameter `auction` and use that ID to get the auction object from the `AuctionService` object that you injected into the servlet.
4. Set the auction object in the `auction` request attribute. Use this code for reference.

```
int auctionId =
Integer.parseInt(request.getParameter("auction"));
Auction auction = auctionService.findAuctionById(auctionId);
request.setAttribute("auction", auction);
```

5. Open the `auctionView.jsp` file located in Web Pages.
6. Add the import `com.lab0701.model.Auction` page directive.
7. Get the selected auction in the `auction` request attribute by using the following JSP scriptlet:

```
<% Auction auction = (Auction)request.getAttribute("auction");%>
```

8. Using the <%= %> JSP expression scriptlet, display the auction properties instead of static text per the following table:

Placeholder	JSP Expression Code
##	<%=auction.getAuctionId()%>
AUCTION_NAME	<%=auction.getTitle()%>
AUCTION_DESCRIPTION	<%=auction.getDescription()%>
AUCTION_CURRENTBID	<%=auction.getCurrPrice()%>
AUCTION_INCREMENT	<%=auction.getIncrement()%>

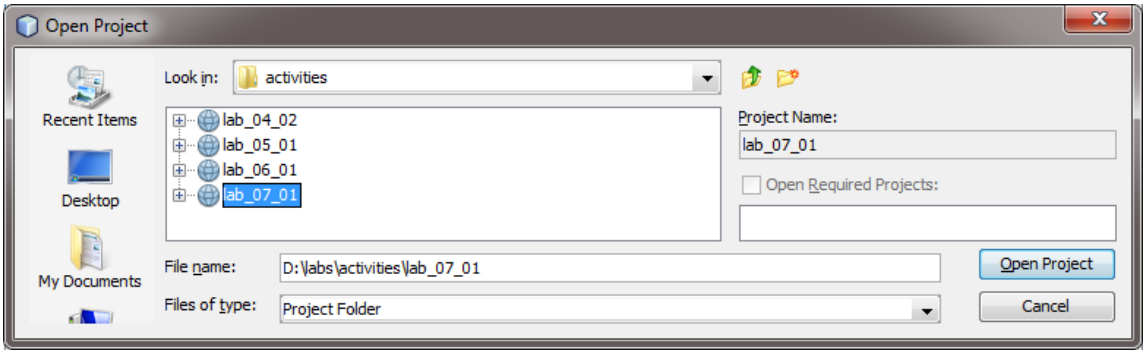
9. Add a scriptlet in place of the `IF_AUCTION_IS_ACTIVE_PRINT` and then test for `auction.isActive()` in the `if` statement.
10. In the body of the condition, use the out object to print the form.
    Use the following code for reference.

```
if(auction.isActive()){
  out.println("<form>");
  out.println("<input type=\"submit\" value=\"BID!\">");
  out.println("</form>");
}
```

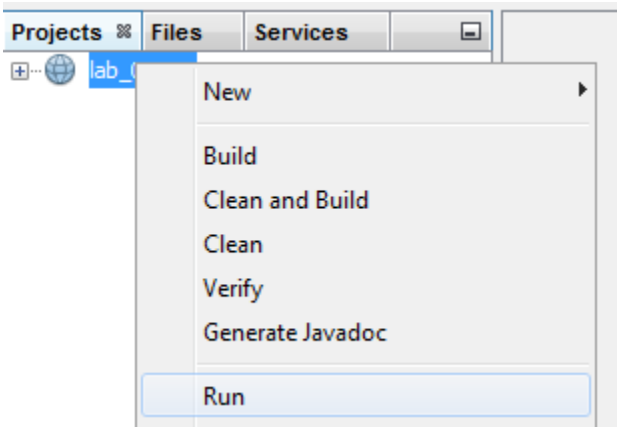
11. Run the application and verify that the details for the auctions are displayed on your browser.
    Note: Clicking the Bid button will result on an error, because that functionality is not available in this practice.

Step-by-Step Instructions

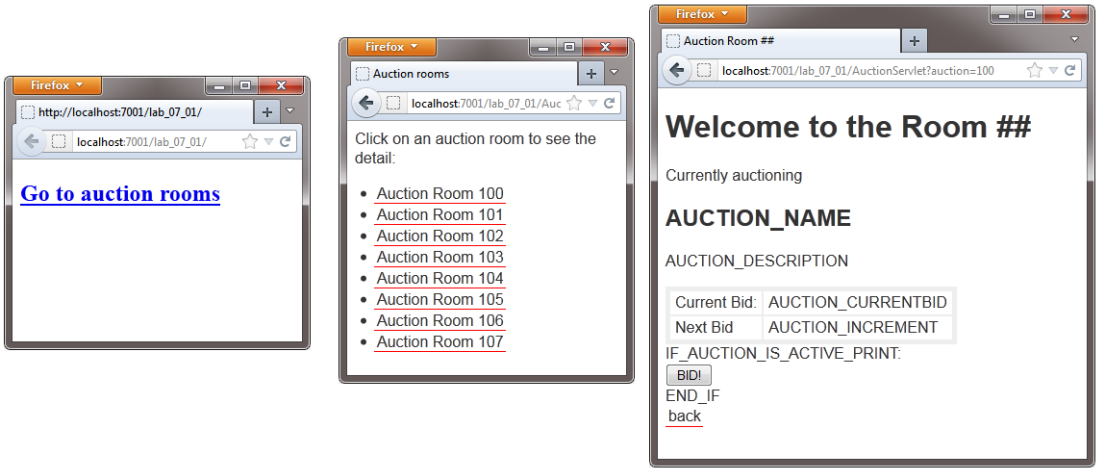
1. Open NetBeans IDE and open the `lab_07_01` project located at:
   `D:\labs\activities\`



2. Right-click the project name and select Run.



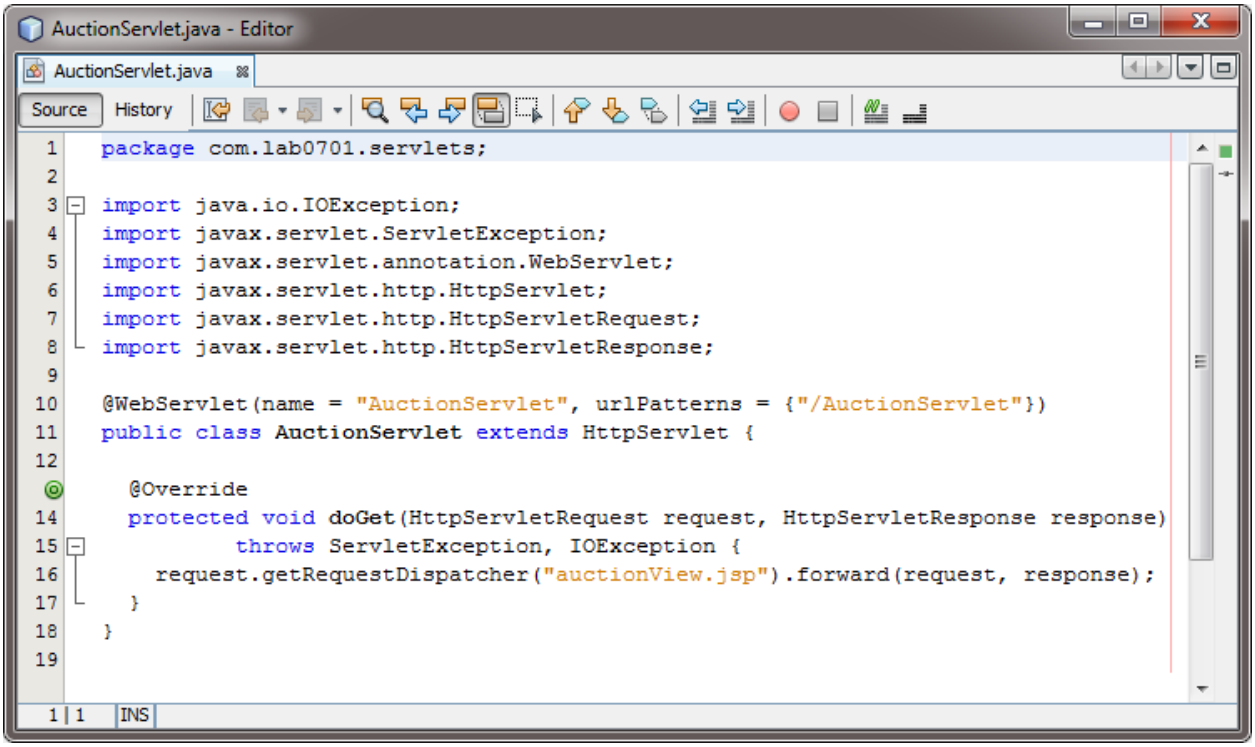
3. Browse to an auction, and note that the auction detail pages do not display the values of the selected auction.



The home page (index.html) contains a link to the Auction List page. The list page is a servlet that redispatches to a JSP file. For now, the JSP file displays the auction list statically, because the auction rooms are hardcoded HTML.

The detail page is a servlet that redispatches to a JSP file with the HTML code that outlines the design. You have to modify the Java servlet and the JSP file to display the auction data.

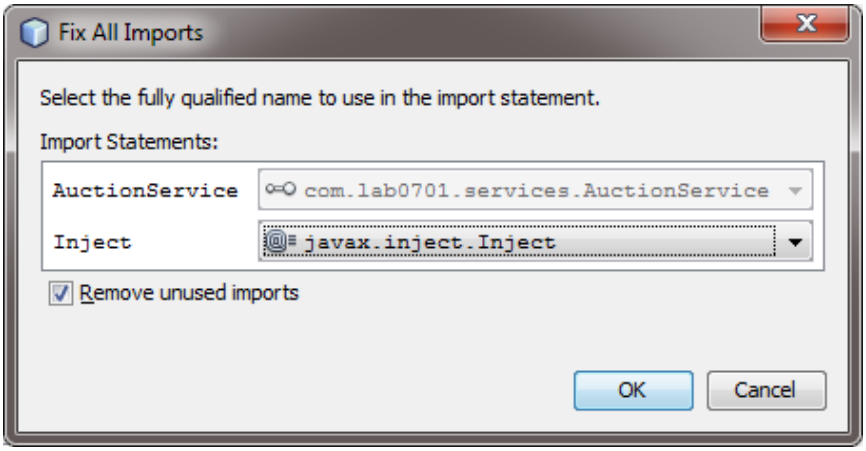
4. Open the `com.lab0701.servlets.AuctionServlet` class.



5. Add the dependency to the `AuctionService` by injecting it with the highlighted code:

```
@WebServlet(name = "AuctionServlet", urlPatterns = {"/AuctionServlet"})
public class AuctionServlet extends HttpServlet {

  @Inject
  private AuctionService auctionService;

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

6. Resolve missing imports by pressing Ctrl + Shift + I or by selecting Source > Fix Imports.



7. Get the selected auction ID from the request by adding the following code to the `doGet` method:

```
int auctionId =
Integer.parseInt(request.getParameter("auction"));
```

8. Get the selected auction by using `AuctionService` with the following code:

```
Auction auction = auctionService.findAuctionById(auctionId);
```

You may need to resolve imports again for the `com.lab0701.model.Auction` class.

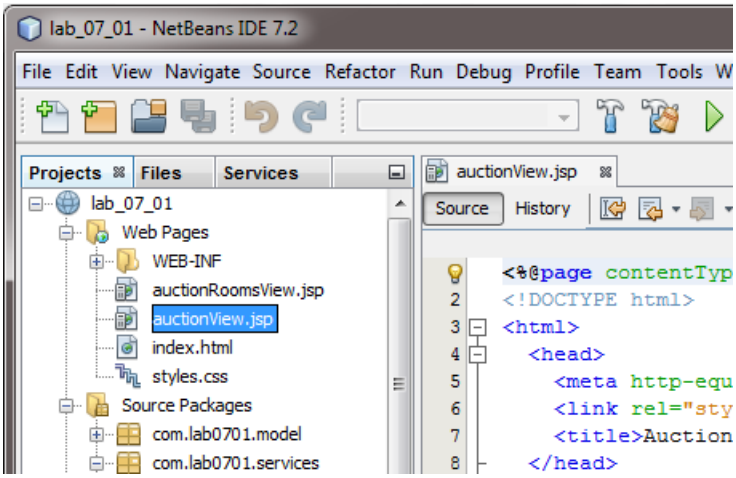
9. Add the auction object to the request attribute auction to retrieve it later on the JSP file by adding the following code:

```
request.setAttribute("auction", auction);
```

Your completed code for the servlet should look similar to this:

```
package com.lab0701.servlets;

import com.lab0701.model.Auction;
import com.lab0701.services.AuctionService;
import java.io.IOException;
import javax.inject.Inject;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "AuctionServlet", urlPatterns = {"/AuctionServlet"})
public class AuctionServlet extends HttpServlet {

  @Inject
  private AuctionService auctionService;

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    int auctionId = Integer.parseInt(request.getParameter("auction"));
    Auction auction = auctionService.findAuctionById(auctionId);
    request.setAttribute("auction", auction);
    request.getRequestDispatcher("auctionView.jsp").forward(request, response);
  }
}
```

10. Open Web Pages > auctionView.jsp.



The `auctionView.jsp` file contains only HTML code; you must add Java code to it by using JSP scriptlets to display dynamic data on the page.

11. Add the scriptlet that will get the auction from the request; do this by adding the highlighted code:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%
  com.lab0701.model.Auction auction = (com.lab0701.model.Auction)request.getAttribute("auction");
%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

If you prefer, you can add the import page directive by replacing the code above with:

```
<%@page import="com.lab0701.model.Auction"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%
  Auction auction = (Auction)request.getAttribute("auction");
%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=
```

You must change the placeholders inside the JSP file. For your convenience, they are in UPPERCASE and describe what value from the auction object you need to display.

12. In the auction room, display the ID of the auction.

    Replace the ## with

```
<%=auction.getAuctionId()%>
```

13. Replace all the remaining placeholders using the following table:

Placeholder	JSP Expression Code
AUCTION_NAME	<%=auction.getTitle()%>
AUCTION_DESCRIPTION	<%=auction.getDescription()%>
AUCTION_CURRENTBID	<%=auction.getCurrPrice()%>
AUCTION_INCREMENT	<%=auction.getIncrement()%>

After replacing the placeholders, your JSP file should look like the following picture (only HTML code is shown):

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" href="styles.css" type="text/css">
    <title>Auction Room <%=auction.getAuctionId()%></title>
  </head>
  <body>
    <h1>Welcome to the Room <%=auction.getAuctionId()%></h1>
    <p>Currently auctioning</p>
    <h2><%=auction.getTitle()%></h2>
    <p><%=auction.getDescription()%></p>
    <table>
      <tr>
        <td>Current Bid:</td>
        <td><%=auction.getCurrPrice()%></td>
      </tr>
      <tr>
        <td>Bid increment:</td>
        <td><%=auction.getIncrement()%></td>
      </tr>
    </table>
```

14. Find the `IF_AUCTION_IS_ACTIVE_PRINT` placeholder. Here you have to include an `if` condition.
15. You have to print the HTML output if the active flag inside the auction object is true. There are two ways to do this: The first way (shown on the left in the following picture) prints the HTML using the out writer object and everything will be Java code. The second way (on the right) splits the `if` braces to enclose the HTML code. Use either of these approaches to display the form.

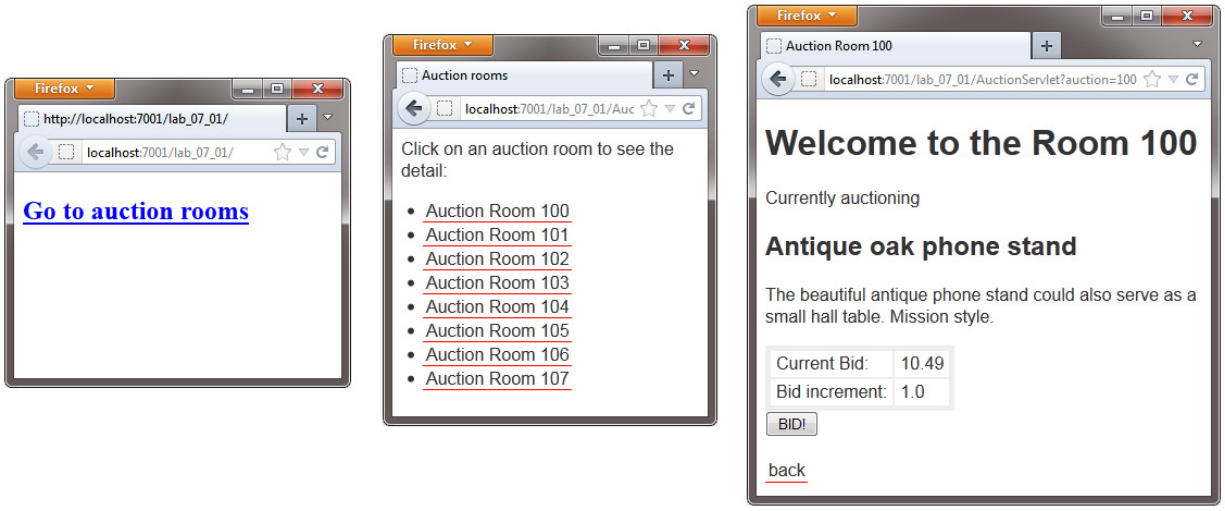
```
    </tr>
  </table>
  <% if (auction.isActive()) {
      out.println("<form>");
      out.println("<input type=\"submit\" value=\"BID!\">");
      out.println("</form>");
    }%>
  <br>
  <a href="AuctionRoomsServlet">back</a>
```

```
    </tr>
  </table>
  <% if (auction.isActive()) {%>
  <form>
    <input type="submit" value="BID!">"
  </form>
  <% }%>
  <br>
  <a href="AuctionRoomsServlet">back</a>
```

Use the code on the following page for reference:

```
<%@page import="com.lab0701.model.Auction"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%
  Auction auction = (Auction) request.getAttribute("auction");
%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" href="styles.css" type="text/css">
    <title>Auction Room <%=auction.getAuctionId()%></title>
  </head>
  <body>
    <h1>Welcome to the Room <%=auction.getAuctionId()%></h1>
    <p>Currently auctioning</p>
    <h2><%=auction.getTitle()%></h2>
    <p><%=auction.getDescription()%></p>
    <table>
      <tr>
        <td>Current Bid:</td>
        <td><%=auction.getCurrPrice()%></td>
      </tr>
      <tr>
        <td>Bid increment:</td>
        <td><%=auction.getIncrement()%></td>
      </tr>
    </table>
        The bid functionality is not implemented.
    <% if (auction.isActive()) {
        out.println("<form>");
        out.println("<input type=\"submit\" value=\"BID!\">");
        out.println("</form>");
      }%>
    <br>
    <a href="AuctionRoomsServlet">back</a>
  </body>
</html>
```

16. Run the project and see how the auctions are displayed on the auction detail pages.



Clicking the Bid button will result on an error. The bid functionality is not implemented yet.

You have completed this practice. You can try this practice's challenge if you wish to more practice with scriptlets. Details for the challenge are on the following page. You can also open this lesson's example project and explore more source code as well as the quiz for this lesson. The project is located in `D:\labs\examples\lesson07`.

Practice 7-2: Challenge Practice: Creating the List of Auction Rooms Dynamically

Overview

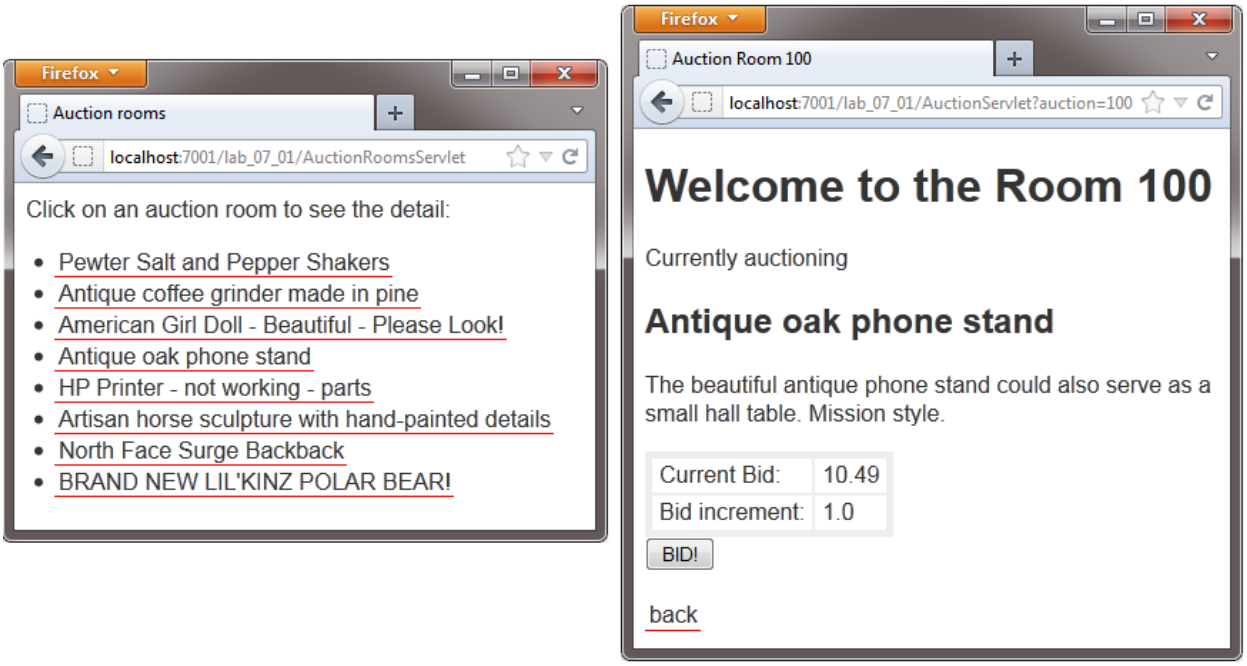
In this practice, you implement the view for the room list by using servlets and JSPs.

Assumptions

The practice 7-1 is completed.

Tasks

1. Modify the `com.lab0701.servlets.AuctionRoomServlet` class.
2. Add the dependency to the `AuctionService` by using the `Inject` annotation.
3. Get the list of auctions from the service.
4. Set the list of auctions in a request attribute of your choice.
5. Modify the Web Pages > `auctionRoomView.jsp` file.
6. Get the list of auctions from the request attribute that you used in the servlet.
7. Replace the static room list with a `FOR` loop over the list of auctions.
   Remember that you can split the `FOR` brackets in different JSP scriptlets.
8. In the link, you have to use `auction.getAuctionId` instead of the room number.
9. Replace the link text with the title for the auction by using a JSP expression scriptlet.



The solution for this challenge is located in the folder `D:\labs\solutions\lab_07_02`.

Additional Information

Remember that JSP pages are servlets and that they are translated and compiled when accessed. Changing a JSP usually causes the translation and compilation process to run again.

Scriptlets allows you to insert Java code inside the JSP pages. Scriptlets are inserted in the translated JSP, and then compilation runs. After a JSP has been compiled, the resulting servlet is instantiated as if it were a normal servlet and run just as a servlet would.

You can see that you can split `if` statements and `FOR` loops, leaving the opening and closing braces in different scriptlets. This is legal but it is hard to keep track of.

What do you think would be a better approach than splitting the braces in a code block?

Remember that you can open more examples related to this practice and lesson using NetBeans.

In the example project, you will find additional source code and quiz questions. The project is located in `D:\labs\examples\lesson07`.

Practices for Lesson 8: JSP Custom Tags

Chapter 8

Practices for Lesson 8

Practices Overview

In the practice for this lesson, you will create the auction list view using custom tags, and you will practice using Expression Language.

Practice 8-1: Creating a JSP View to List All the Auctions

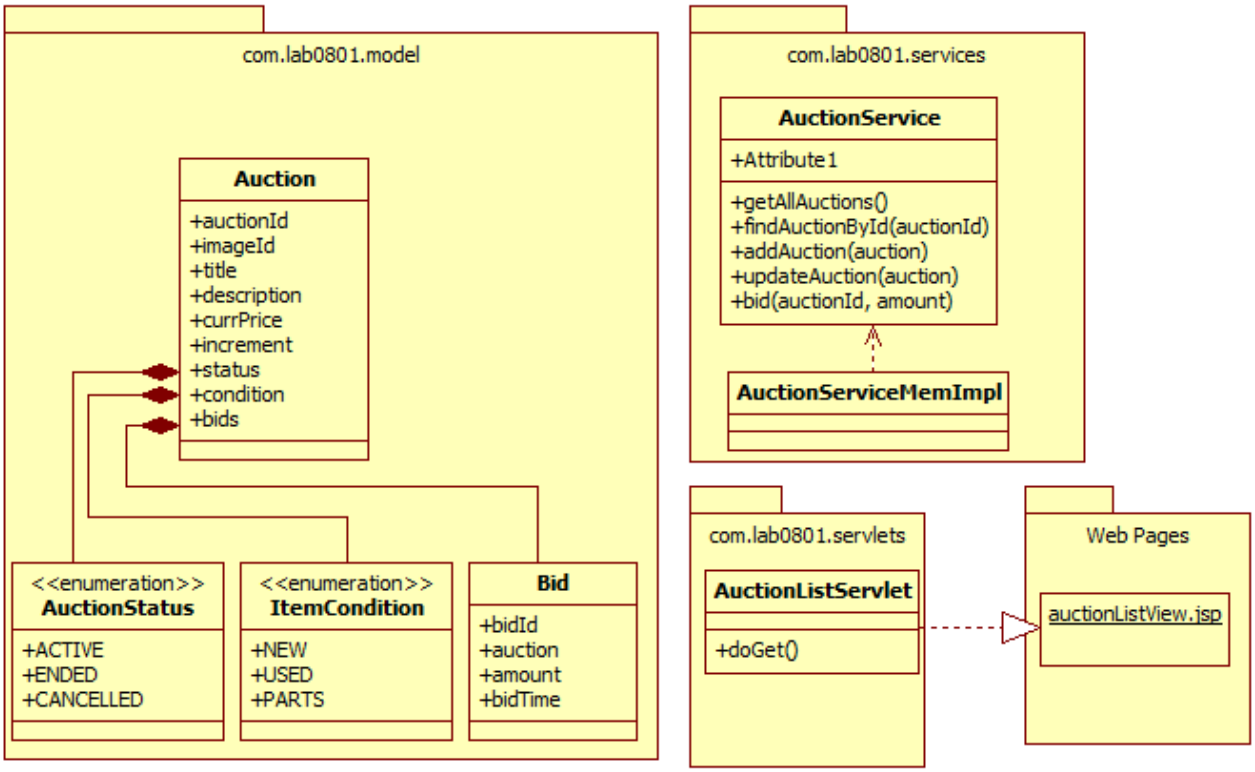
Overview

In this practice, you use the JSP Standard Tag Library (JSTL) and Expression Language (EL) to create a JSP page to display all the auctions in the system.

You start with an existing project built on the same auction model that you used in the previous practice.

The Auction Project for This Practice (lab_08_01)

The classes and pages available for this practice are outlined in the following diagram.



The model classes describe the properties of the `Auction` object. Note that the auction object has two enumerations inside and a collection of `Bid` objects to describe the bids made to the auction.

The `AuctionService` class now specifies more methods. Although you will likely use only one method, the others are provided and implemented in case you want to do some experimentation.

The `auctionListServlet` is mapped to the `index.html` URL so that the application starts right away in this servlet. This servlet redispatches to the `auctionListView.jsp` file.

The `auctionListView.jsp` has no scriptlets or JSP tags, but it contains HTML to provide a layout for the auction data. It contains text placeholders in the form of uppercase strings that you replace with the data from the auction or JSTL tags to include logic operations.

Practice Overview

For this practice, you modify the servlet and the JSP view to add dynamic data from the service class.

1. Open `com.lab0801.servlets.AuctionListServlet` and inject `AuctionService` using CDI annotations.
2. In the `doGet` method, retrieve the list of auctions from the `AuctionService` object and set that list to a request attribute named `allAuctions`.
3. Redispatch the request to `auctionListView.jsp`.
4. Open the `auctionListView.jsp` file, which is in Web Pages.
5. Add the `taglib` JSP directive to use the JSTL core tag library.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

6. Locate the XML comment `ITERATE_AUCTIONS` and add a `c:forEach` tag to iterate the auctions from the `allAuctions` request attribute. Use the following code for reference:

```
<c:forEach items="${allAuctions}" var="auction">
```

7. Close the tag below the `END_ITERATE_AUCTIONS` XML comment by using:

```
</c:forEach>
```

8. Use the `auction` variable inside the `forEach` tag to output the values of the auction.
9. Replace the auction text placeholders per the following table. You will use Expression Language to output the auction values.

Text Placeholder	Expression Language
AUCTION_TITLE	${auction.title}
AUCTION_DESCRIPTION	${auction.description}
AUCTION_STATUS	${auction.status}
AUCTION_CONDITION	${auction.condition}
AUCTION_CURRENTPRICE	${auction.currPrice}
AUCTION_INCREMENT	${auction.increment}

10. Find the XML comment `ITERATE_BIDS_ON_AUCTION` and add a `c:forEach` tag to iterate the bids, using the following code:

```
<c:forEach items="${auction.bids}" var="bid">
```

11. Close the `c:forEach` tag using `</c:forEach>` below the `END_ITERATE_BIDS_ON_AUCTION` XML comment.
12. Replace the placeholder for the bid per the following table:

Text Placeholder	Expression Language
BID_AMOUNT	${bid.amount}

13. You have to add an `IF` statement to display a message instead of the list of bids if no bids are available.
14. First add the `IF` statement to display the bids table if the bids are not empty; do so adding the following tag after the `IF_AUCTION_BIDS_NOT_EMPTY` XML comment and close it after the `END_IF_AUCTION_BIDS_NOT_EMPTY` comment:

```
<c:if test="${not empty auction.bids}">
```

15. Because there is no `else` in `IF` statements using JSTL `c:if` tags, you must add the opposite condition. Locate the `IF_AUCTION_BIDS_EMPTY` XML comment and add the following tag. Remember to close it after the `END_IF_AUCTION_BIDS_EMPTY` XML comment.

```
<c:if test="${empty auction.bids}">
```

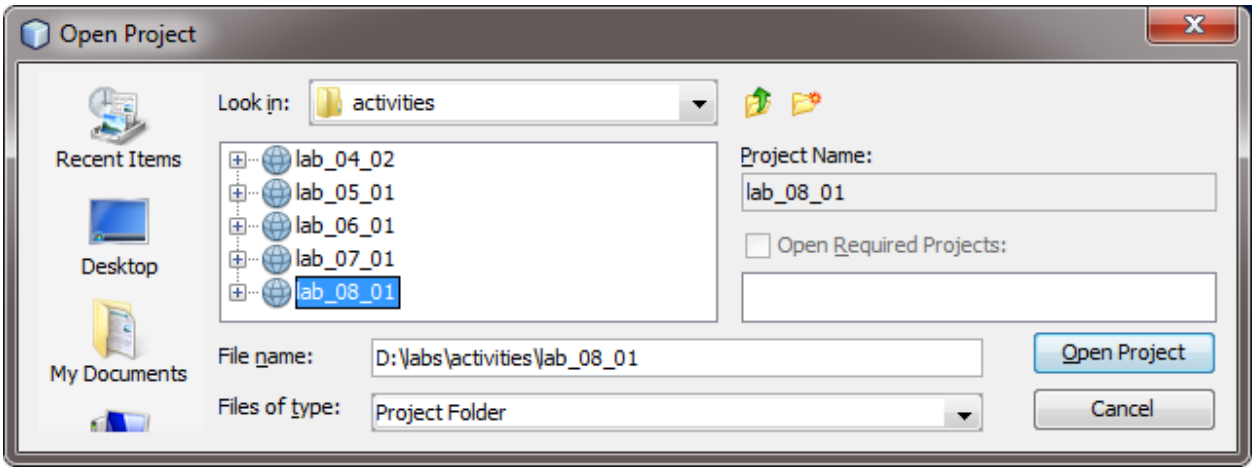
16. Use the following code tag to close the `c:if` tag:

```
</c:if>
```

17. Run the application, you will see all the auctions listed displaying the details in your web browser.

Step-by-Step Instructions

1. Open the `lab_08_01` project at `d:\labs\activities`.



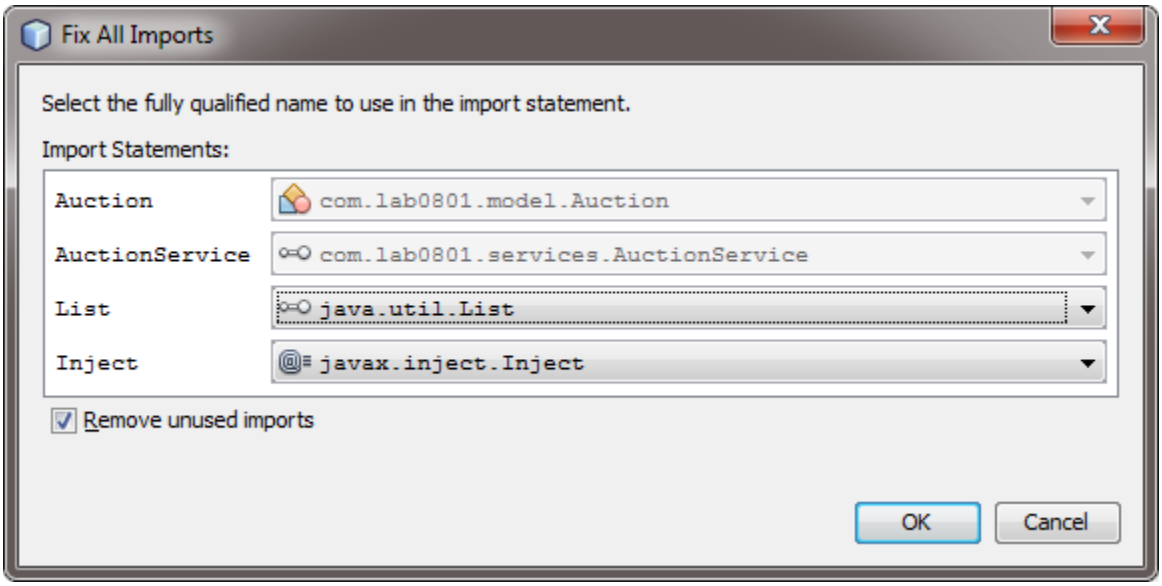
2. Open the `AuctionListServlet` and add the following code to inject the `AuctionService` in the servlet.

```
@Inject
private AuctionService auctionService;
```

3. Add the following code to the body of the `doGet` method to get the list of all the auctions, add the list to the request, and redispatch to the `auctionListView.jsp`.

```
List<Auction> auctions = auctionService.getAllAuctions();
request.setAttribute("allAuctions", auctions);
request.getRequestDispatcher("auctionListView.jsp").forward(requ
est, response);
```

You may need to fix imports. Do so by pressing Ctrl + Shift + I or by selecting Source > Fix Imports.

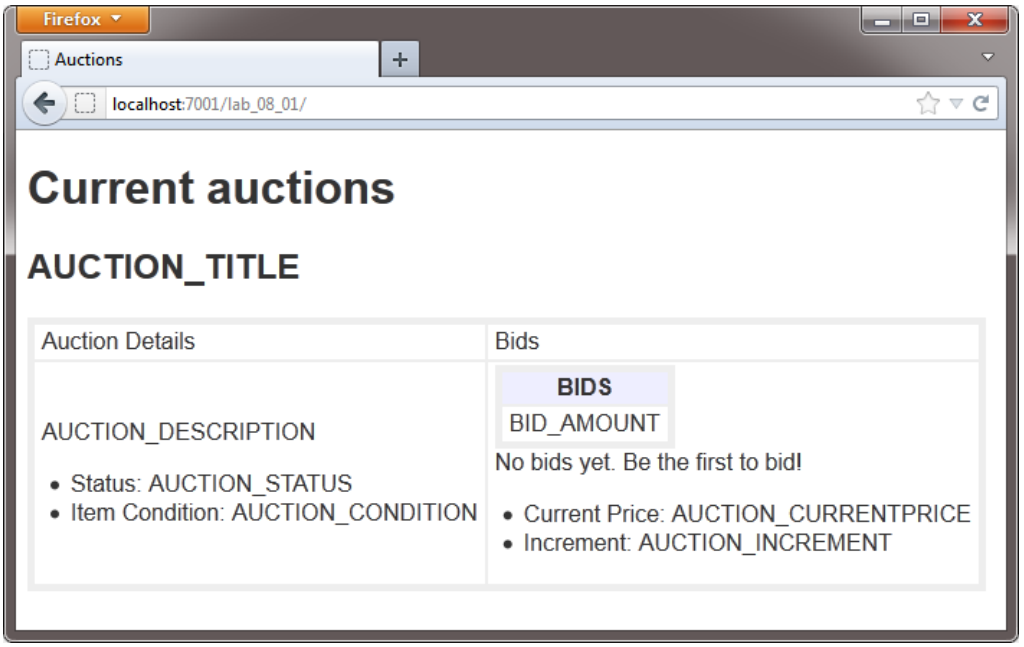


The servlet code will be similar to the following:

```
package com.lab0801.servlets;

import com.lab0801.model.Auction;
import com.lab0801.services.AuctionService;
import java.io.IOException;
import java.util.List;
import javax.inject.Inject;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "auctionListServlet", urlPatterns = {"/index.html"})
public class AuctionListServlet extends HttpServlet {

  @Inject
  private AuctionService auctionService;

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    List<Auction> auctions = auctionService.getAllAuctions();
    request.setAttribute("allAuctions", auctions);
    request.getRequestDispatcher("auctionListView.jsp").forward(request, response);
  }
}
```

4. Run the project to test the servlet. Right-click the project name and select Run. A browser window will open displaying the rendered `auctionListView.jsp` file.

   You need to modify this file to add dynamic data using the JSP Standard Tag Library (JSTL) and Expression Language (EL)



5. Open the `auctionListView.jsp` file.
6. Add the `taglib` page directive to use the JSTL core tag library.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
```

7. Iterate the auctions by adding the `c:forEach` tag after the `ITERATE_AUCTIONS` comment and closing it after the `END_ITERATE_AUCTIONS` XML comment.

```
    <h1>Current auctions</h1>
    <!-- ITERATE_AUCTIONS -->
    <c:forEach items="${allAuctions}" var="auction">
      <h2>AUCTION_TITLE</h2>
      <table>
        <thead>
          <tr><td>
          </tr>
        </tbody>
      </table>
      <!-- END_ITERATE_AUCTIONS -->
    </c:forEach>
  </body>
```

8. Replace the following text with Expression Language per the following table:

Text Placeholder	Expression Language
AUCTION_TITLE	${auction.title}
AUCTION_DESCRIPTION	${auction.description}
AUCTION_STATUS	${auction.status}
AUCTION_CONDITION	${auction.condition}
AUCTION_CURRENTPRICE	${auction.currPrice}
AUCTION_INCREMENT	${auction.increment}

For reference, the code, with the replaced text highlighted, is on the next page:

```
<h1>Current auctions</h1>
<!-- ITERATE_AUCTIONS -->
<c:forEach items="${allAuctions}" var="auction">
  <h2>${auction.title}</h2>
  <table>
    <thead>
      <tr>
        <td>Auction Details</td>
        <td>Bids</td>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>
          <p>${auction.description}</p>
          <ul>
            <li>Status: ${auction.status}</li>
            <li>Item Condition: ${auction.condition}</li>
          </ul>
        </td>
        <td>
          <!-- IF_AUCTION_BIDS_NOT_EMPTY -->
          <table>
            <thead>
              <tr><th>BIDS</th></tr>
            </thead>
            <tbody>
              <!-- ITERATE_BIDS_ON_AUCTION -->
              <tr>
                <td>
                  BID_AMOUNT
                </td>
              </tr>
              <!-- END_ITERATE_BIDS_ON_AUCTION -->
            </tbody>
          </table>
          <!-- END_IF_AUCTION_BIDS_NOT_EMPTY -->
          <!-- IF_AUCTION_BIDS_EMPTY -->
          No bids yet. Be the first to bid!
          <!-- END_IF_AUCTION_BIDS_EMPTY -->
          <ul>
            <li>Current Price: ${auction.currPrice}</li>
            <li>Increment: ${auction.increment}</li>
          </ul>
        </td>
      </tr>
```

9. To iterate the bids inside an auction locate `ITERATE_BIDS_ON_AUCTION` and `END_ITERATE_BIDS_ON_AUCTION` and add another `c:forEach` tag.

```
<c:forEach items="${auction.bids}" var="bid">
```

10. Replace `BID_AMOUNT` with the `${bid.amount}` expression.

    You do not want to display the table if there are no items in it. You can use a `c:if` tag to add a condition in the web page. Unfortunately, there is no way to specify an `else` clause using the `c:if` tag. You must invert the condition to achieve a similar effect.

11. Add the first `c:if` tag after the `IF_AUCTION_BIDS_NOT_EMPTY` XML comment.

```
<c:if test="${not empty auction.bids}">
```

12. Close the tag with a `</c:if>` tag after the `END_IF_AUCTION_BIDS_NOT_EMPTY` XML comment.

13. Add the second `c:if` tag after the `IF_AUCTION_BIDS_EMPTY` XML comment.

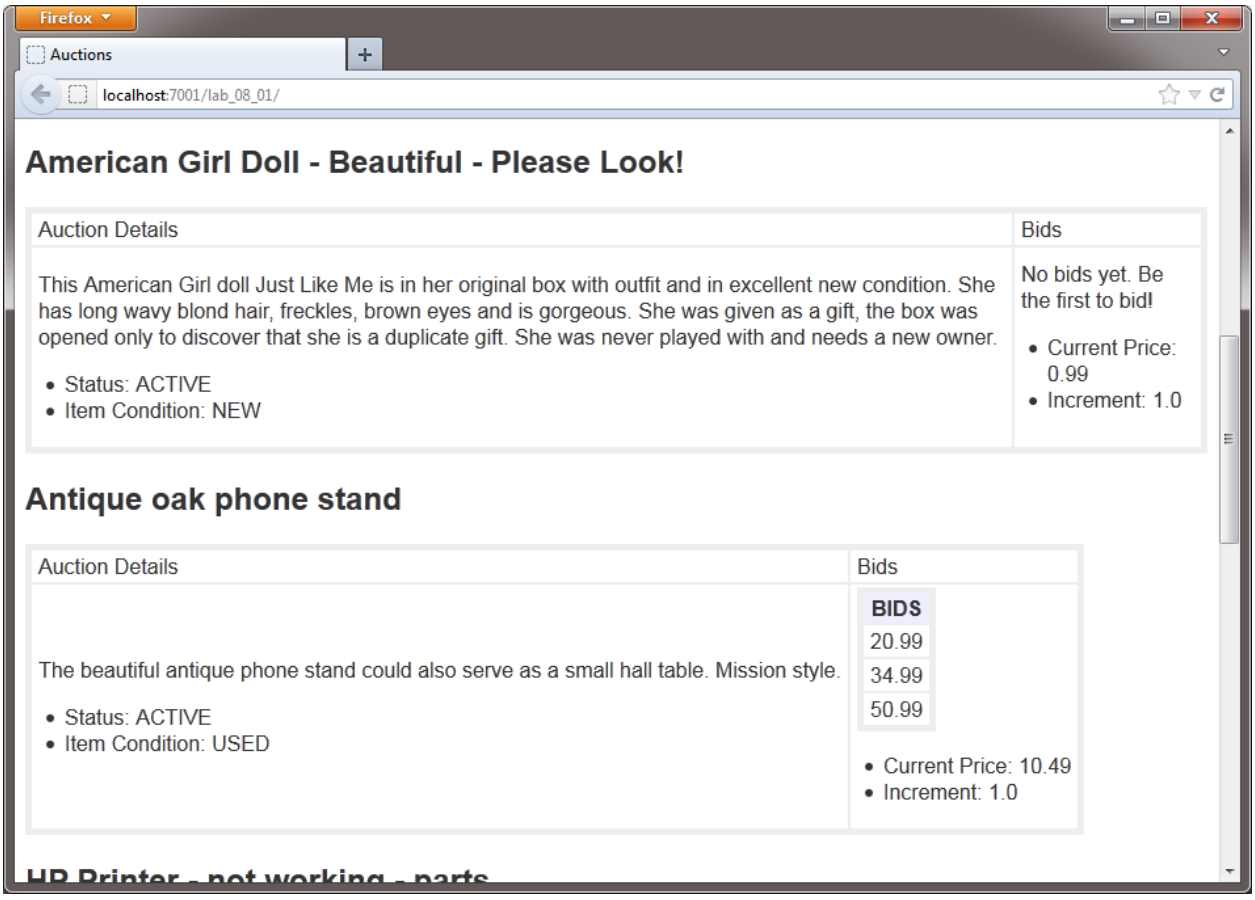
```
<c:if test="${empty auction.bids}">
```

14. Close the tag with a `</c:if>` tag after the `END_IF_AUCTION_BIDS_EMPTY` XML comment.

    Use the following code for reference. The modified text is highlighted.

```
<td>
  <!-- IF_AUCTION_BIDS_NOT_EMPTY -->
  <c:if test="${not empty auction.bids}">
    <table>
      <thead>
        <tr><th>BIDS</th></tr>
      </thead>
      <tbody>
        <!-- ITERATE_BIDS_ON_AUCTION -->
        <c:forEach items="${auction.bids}" var="bid">
          <tr>
            <td>
              ${bid.amount}
            </td>
          </tr>
          <!-- END_ITERATE_BIDS_ON_AUCTION -->
        </c:forEach>
      </tbody>
    </table>
    <!-- END_IF_AUCTION_BIDS_NOT_EMPTY -->
  </c:if>
  <!-- IF_AUCTION_BIDS_EMPTY -->
  <c:if test="${empty auction.bids}">
    No bids yet. Be the first to bid!
    <!-- END_IF_AUCTION_BIDS_EMPTY -->
  </c:if>
  <ul>
    <li>Current Price: ${auction.currPrice}</li>
    <li>Increment: ${auction.increment}</li>
  </ul>
```

15. Right-click the project name and select Run.
    You should see the auction list in your browser window.



You can find more examples related to this practice and lesson by opening and running the project in: `D:\labs\examples\lesson08` using NetBeans.

Practices for Lesson 9: Servlet Filters

Chapter 9

Practices for Lesson 9

Practices Overview

In the practice for this lesson, you will implement a servlet filter to redispatch the request to a mobile or desktop version based on the browser and user preferences.

Practice 9-1: Implementing a Servlet Filter to Display a Mobile Site

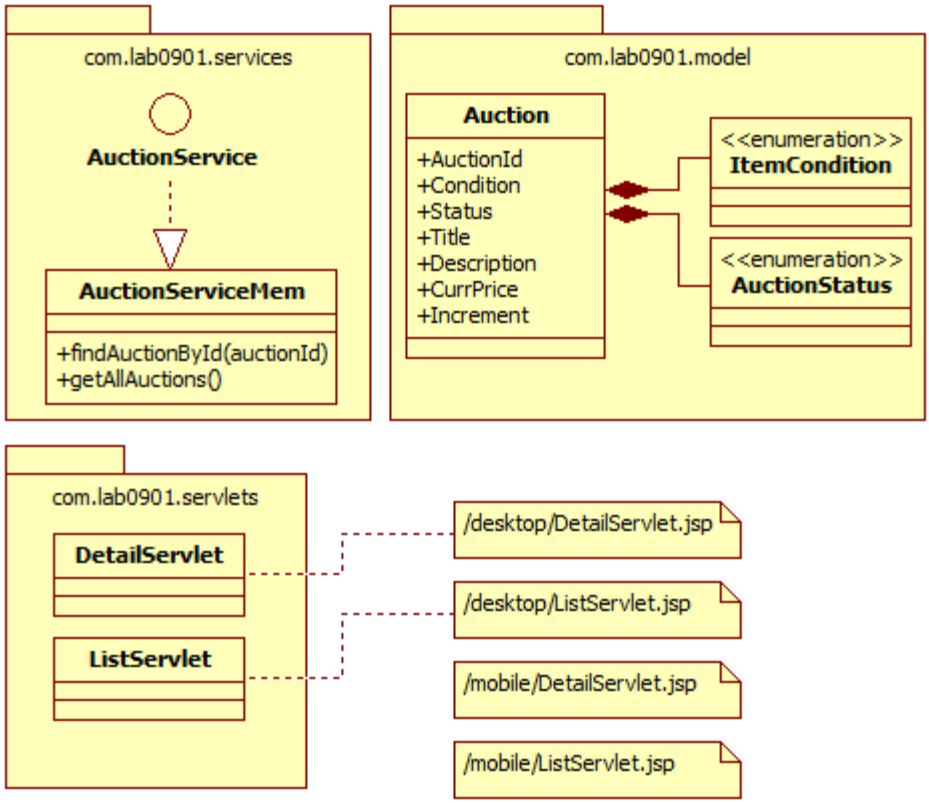
Overview

In this practice, you use a servlet filter to automatically redispatch requests made to servlets to JSP pages to render the output. Additionally the filter will decide to use a desktop or a mobile version of the JSP based on browser headers or user preferences stored in the session.

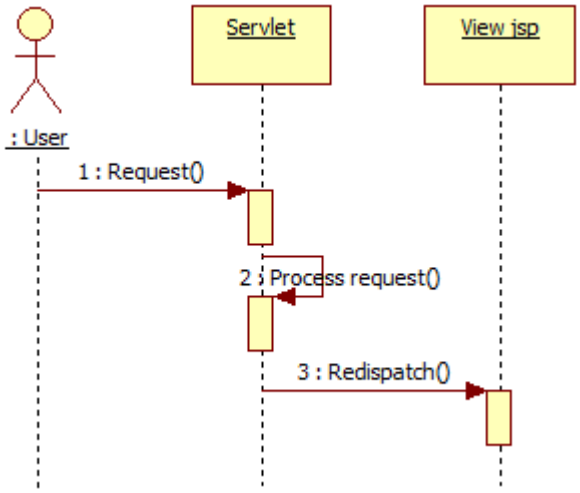
You will develop this practice's filter and servlets as part of the auctioning application from previous practices. For your convenience and to minimize practice time, you are provided with an existing project that you modify to complete this practice.

The Auction Project for This Practice (lab_09_01)

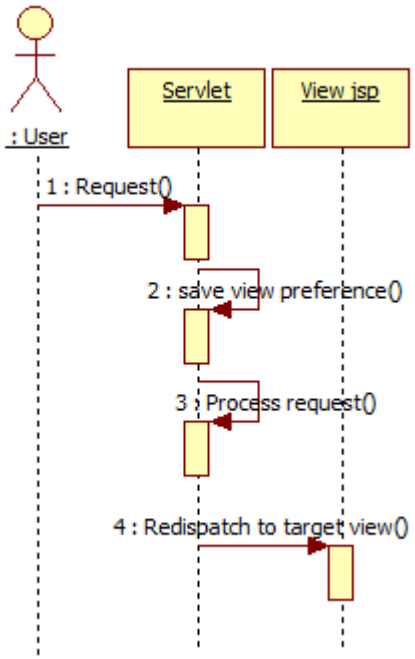
The classes and pages available for this practice are outlined in the following diagram.



The following diagram describes the current flow of the page:

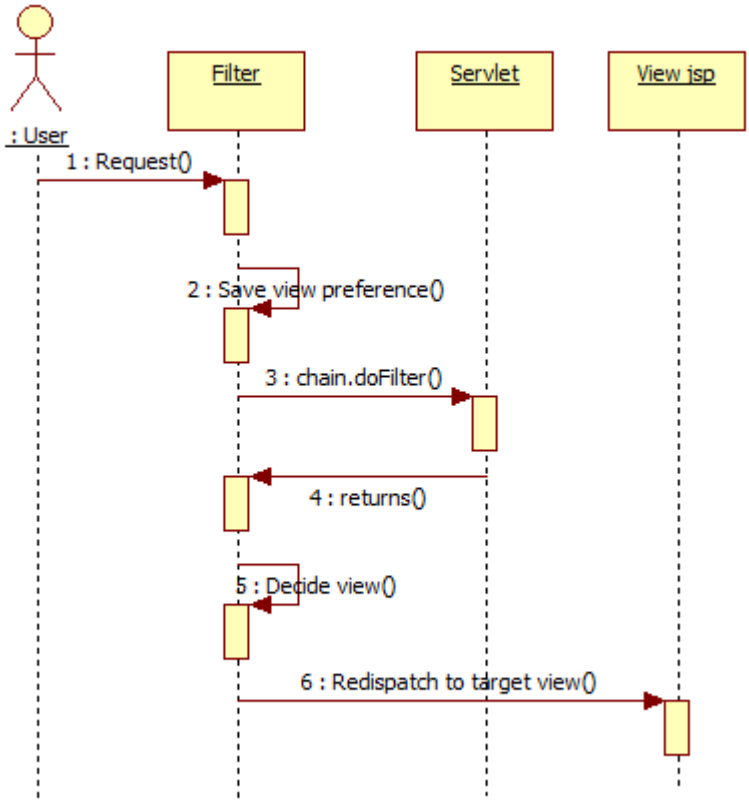


Adding the mobile preference to the servlet would result in the following flow:



This flow will be duplicated for all the servlets resulting in repeated code in all the servlets.

By using a filter and moving all the mobile-related code to the filter, you can leave the servlets unmodified and avoid repeating code.



Practice Overview

For this practice, you need to create a servlet filter and modify the two existing servlets: x and y.

1. Open the `lab_09_01` project in `D:\labs\activities`.
2. Run the application, and note how it is redirecting to desktop version views and the switch view link does not work.
3. Create an empty servlet filter named: `com.lab0901.filters.MobileSiteFilter`.

   Use the following code for reference for an empty filter.

```
@WebFilter(filterName = "MobileSiteFilter", servletNames =
{"DetailServlet", "ListServlet"})
public class MobileSiteFilter implements Filter {


  @Override
  public void doFilter(ServletRequest request, ServletResponse
response, FilterChain chain) throws IOException,
ServletException {
    // go to the filtered resource.
    chain.doFilter(request, response);
  }


  @Override
  public void init(FilterConfig filterConfig) throws
ServletException {
  }


  @Override
  public void destroy() {
  }
}
```

   The code above will create the filter needed for this practice. It is mapped to the existing servlets.

4. Add the following code after the `chain.doFilter(request, response);` method call to redispatch to a JSP named the same as the invoked servlet:

```
    if (request instanceof HttpServletRequest) {
      HttpServletRequest httpRequest = (HttpServletRequest)
request;
      String contextPath = httpRequest.getContextPath();
      String requestURI = httpRequest.getRequestURI();
      String path = requestURI.substring(contextPath.length());
      String viewPath = "/desktop" + path + ".jsp";
      try {
        RequestDispatcher requestDispatcher =
request.getRequestDispatcher(viewPath);
        requestDispatcher.forward(request, response);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
```

5. Remove the redispatch code from the servlets. Redispatching will be done in the filter.
6. Run the application again and test that it works the same as it used to.
7. Add the following code in the filter class to add a method to check whether the browser is a mobile browser:

```
  private boolean isMobile(HttpServletRequest httpRequest) {
    String agentHeader = httpRequest.getHeader("User-Agent");
    return agentHeader != null &&
agentHeader.toLowerCase().contains("mobile");
  }
```

8. Replace `String viewPath = "/desktop" + path + ".jsp";` with: `String viewPath = (isMobile(httpRequest)? "/mobile" : "/desktop") + path + ".jsp";`

9. Use the mobile browser link on your desktop to open a browser that simulates a mobile device and open the URL http://localhost:7001/lab_09_01/ in it.

10. Add the following method to parse the request parameter for the site mode switch.

```
  private void setMobileIfAvailable(HttpServletRequest
httpRequest) {
    String siteMode = httpRequest.getParameter("siteMode");
    if (siteMode != null) {
      boolean isMobile = siteMode.equals("mobile");
      httpRequest.getSession().setAttribute("mobileEnabled",
isMobile);
    }
  }
```

11. Add a call to the `setMobileIfAvailable` method in the `doFilter` method before the `viewPath` is set.
12. Modify the isMobile method, adding the following code to use the session attribute if available.

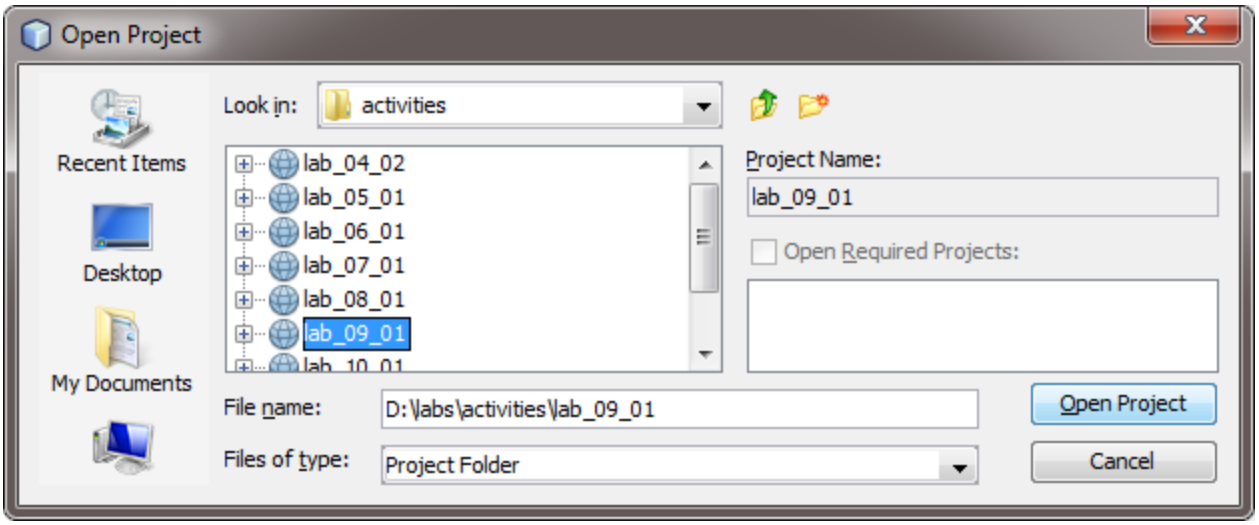
```
  private boolean isMobile(HttpServletRequest httpRequest) {
    Boolean mobileEnabled;
    mobileEnabled = (Boolean)
httpRequest.getSession().getAttribute("mobileEnabled");
    if (mobileEnabled != null) {
      return mobileEnabled;
    }
    String agentHeader = httpRequest.getHeader("User-Agent");
    return agentHeader != null &&
agentHeader.toLowerCase().contains("mobile");
  }
```

13. Test the application with any browser, the application should switch correctly from the desktop to the mobile version.

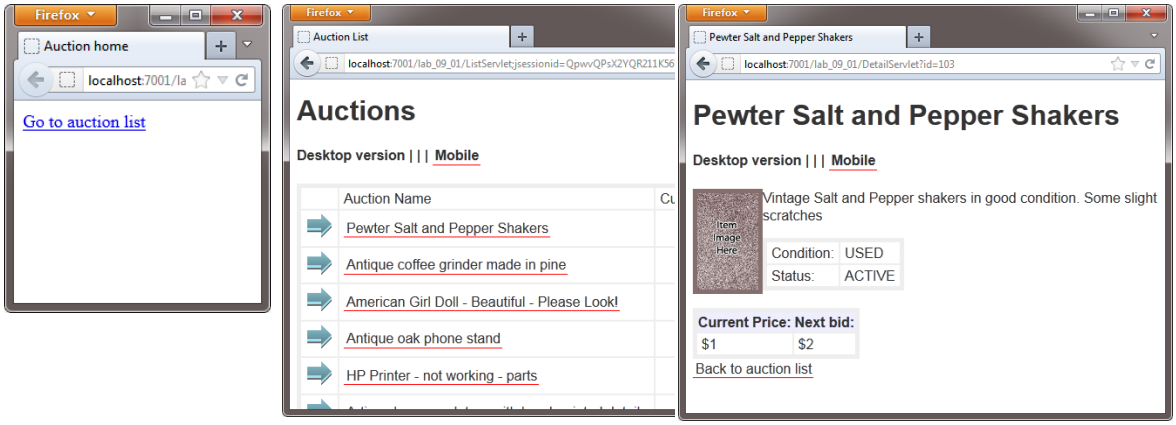
    You may check the code for the servlets and the JSP's for more reference material.

Step-by-Step Instructions

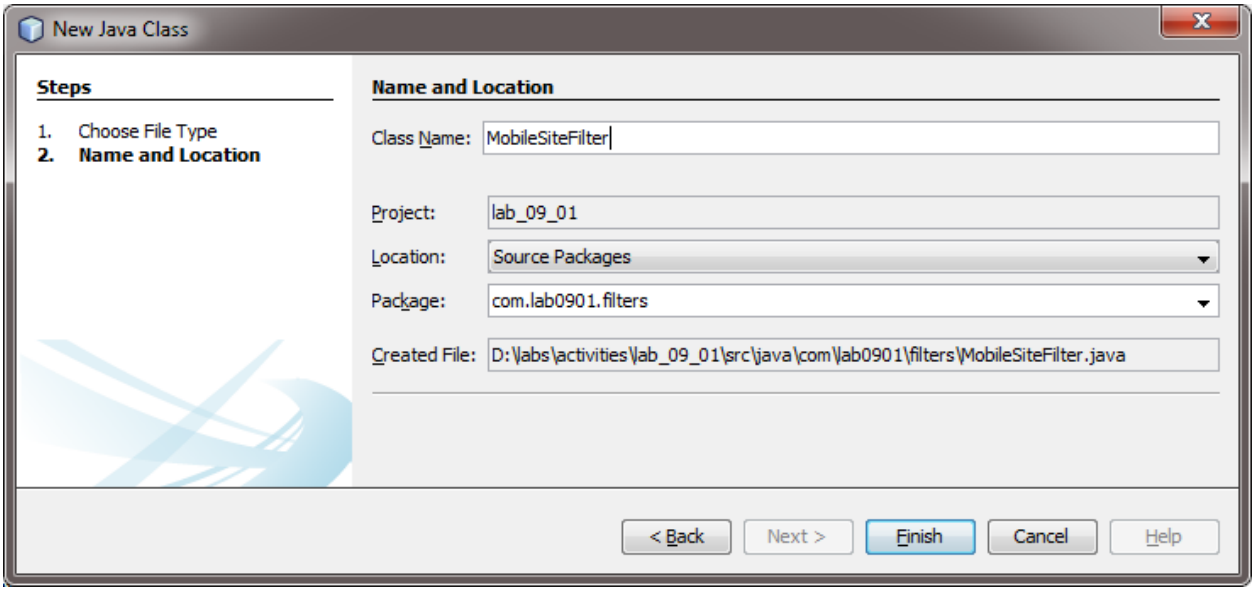
1. Open the project `lab_09_01` in `D:\labs\activities\`.



2. Right-click the Project name in the Projects tab and select Run. A browser opens the URL http://localhost:7001/lab_09_01/.



3. Create a new servlet filter class. Right-click Source Packages and select New > Java Class, name it `MobileSiteFilter`, and put it in the `com.lab0901.filters` package.



4. Make the class implement the `javax.servlet.Filter` interface. Add the following code to do so:

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MobileSiteFilter implements Filter {

  @Override
  public void init(FilterConfig filterConfig) throws
ServletException {
  }

  @Override
  public void doFilter(ServletRequest request, ServletResponse
response, FilterChain chain) throws IOException,
ServletException {
    chain.doFilter(request, response);
  }

  @Override
  public void destroy() {
  }
}
```

5. Add the `WebFilter` annotation to filter requests with this filter:

```
@WebFilter(filterName = "MobileSiteFilter", servletNames =
{"DetailServlet", "ListServlet"})
```

Remember to add the import or fix it using the Fix Imports command.

```
import javax.servlet.annotation.WebFilter;
```

6. Run the application again. Because the filter does nothing yet, the application should be the same.
7. The filter redispatches requests to JSPs that are named the same as the servlet. There are two folders with such JSPs: `/mobile` and `/desktop`. The servlets redispatch to the desktop versions. Migrate this functionality to the filter by adding the following code in the `doFilter` method after the `chain.doFilter(request, response)` call.

```
    if (request instanceof HttpServletRequest) {
      HttpServletRequest httpRequest = (HttpServletRequest)
request;
      setMobileIfAvailable(httpRequest);
      String contextPath = httpRequest.getContextPath();
      String requestURI = httpRequest.getRequestURI();
      String path = requestURI.substring(contextPath.length());
      String viewPath = "/desktop" + path + ".jsp";
      try {
        request.getRequestDispatcher(viewPath).forward(request,
response);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
```

8. Add the import `javax.servlet.http.HttpServletRequest`. Use the fix imports command or add it manually to the code.

The code of the servlet filter should look like this in NetBeans:

```
package com.lab0901.filters;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;

@WebFilter(filterName = "MobileSiteFilter",
        servletNames = {"DetailServlet", "ListServlet"})
public class MobileSiteFilter implements Filter {

  @Override
  public void init(FilterConfig filterConfig) throws ServletException {
  }

  @Override
  public void doFilter(ServletRequest request, ServletResponse response,
      FilterChain chain)
          throws IOException, ServletException {
    chain.doFilter(request, response);
    if (request instanceof HttpServletRequest) {
      HttpServletRequest httpRequest = (HttpServletRequest) request;
      String contextPath = httpRequest.getContextPath();
      String requestURI = httpRequest.getRequestURI();
      String path = requestURI.substring(contextPath.length());
      String viewPath = "/desktop" + path + ".jsp";
      try {
        request.getRequestDispatcher(viewPath).forward(request, response);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }

  @Override
  public void destroy() {
  }
}
```

9. Open the `com.lab0901.servlets.DetailServlet` class and remove the `request.getRequestDispatcher("/desktop/DetailServlet.jsp").forward (request, response);` line.
10. Open the `com.lab0901.servlets. ListServlet` class and remove the `request.getRequestDispatcher("/desktop/ListServlet.jsp").forward (request, response);` line.
11. Run the application again. The functionality stays the same and now the redispatch is happening in the filter.
12. Add the following method to the filter class:

```
private boolean isMobile(HttpServletRequest httpRequest) {
  String agentHeader = httpRequest.getHeader("User-Agent");
  return agentHeader != null &&
agentHeader.toLowerCase().contains("mobile");
}
```

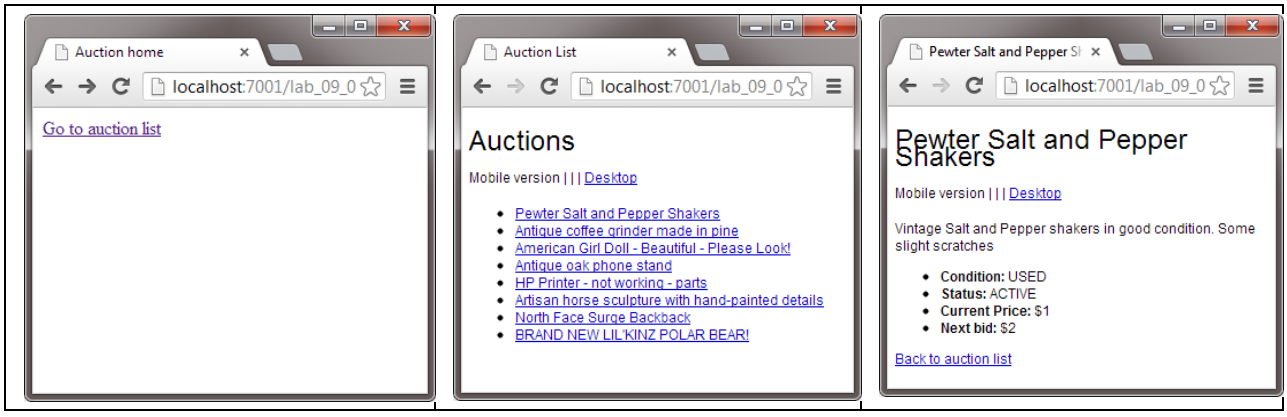
13. In the `doFilter` method, replace the `viewPath` value with:

```
String viewPath = (isMobile(httpRequest) ? "/mobile" :
"/desktop") + path + ".jsp";
```

14. Run the application and open the URL http://localhost:7001/lab_09_01/ using the mobile

Chrome browser link on your desktop.

This browser version emulates a mobile device.



15. Open it in another browser and notice the difference in presentation.
16. Add the following method to the filter class to add a user preference switch:

```
  private void setMobileIfAvailable(HttpServletRequest
httpRequest) {
    String siteMode = httpRequest.getParameter("siteMode");
    if (siteMode != null) {
      boolean isMobile = siteMode.equals("mobile");
      httpRequest.getSession().setAttribute("mobileEnabled",
isMobile);
    }
  }
```

17. Modify the `isMobile` method. Add the following at the beginning of the method:

```
    Boolean mobileEnabled;
    mobileEnabled = (Boolean)
httpRequest.getSession().getAttribute("mobileEnabled");
    if (mobileEnabled != null) {
      return mobileEnabled;
    }
```

18. Modify the `doFilter` method. Add the following before the `viewPath` declaration:

```
      setMobileIfAvailable(httpRequest);
```

19. Run the application and test the site switch buttons.

You can find more examples related to this practice and lesson by opening and running the project in: `D:\labs\examples\lesson09` using NetBeans.

Use the following code from NetBeans for reference on the code for the filter for this practice:

```
package com.lab0901.filters;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;

@WebFilter(filterName = "MobileSiteFilter",
servletNames = {"DetailServlet", "ListServlet"})
public class MobileSiteFilter implements Filter {

  @Override
  public void doFilter(ServletRequest request, ServletResponse response,
          FilterChain chain)
          throws IOException, ServletException {
    chain.doFilter(request, response);
    if (request instanceof HttpServletRequest) {
      HttpServletRequest httpRequest = (HttpServletRequest) request;
      String contextPath = httpRequest.getContextPath();
      String requestURI = httpRequest.getRequestURI();
      String path = requestURI.substring(contextPath.length());
      setMobileIfAvailable(httpRequest);
      String viewPath = (isMobile(httpRequest) ? "/mobile" : "/desktop") + path + ".jsp";
      try {
        request.getRequestDispatcher(viewPath).forward(request, response);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }

  private void setMobileIfAvailable(HttpServletRequest httpRequest) {
    String siteMode = httpRequest.getParameter("siteMode");
    if (siteMode != null) {
      boolean isMobile = siteMode.equals("mobile");
      httpRequest.getSession().setAttribute("mobileEnabled", isMobile);
    }
  }

  private boolean isMobile(HttpServletRequest httpRequest) {
    Boolean mobileEnabled;
    mobileEnabled = (Boolean) httpRequest.getSession().getAttribute("mobileEnabled");
    if (mobileEnabled != null) {
      return mobileEnabled;
    }
    String agentHeader = httpRequest.getHeader("User-Agent");
    return agentHeader != null && agentHeader.toLowerCase().contains("mobile");
  }

  @Override
  public void init(FilterConfig filterConfig) throws ServletException {
  }

  @Override
  public void destroy() {
  }
}
```

Practices for Lesson 10: File Upload

Chapter 10

Practices for Lesson 10

Practices Overview

In the practice for this lesson, you will use the servlet 3.0 file upload feature to add pictures to the auctions added by the user.

Practice 10-1: Implementing File Upload to Add Images to Auctions

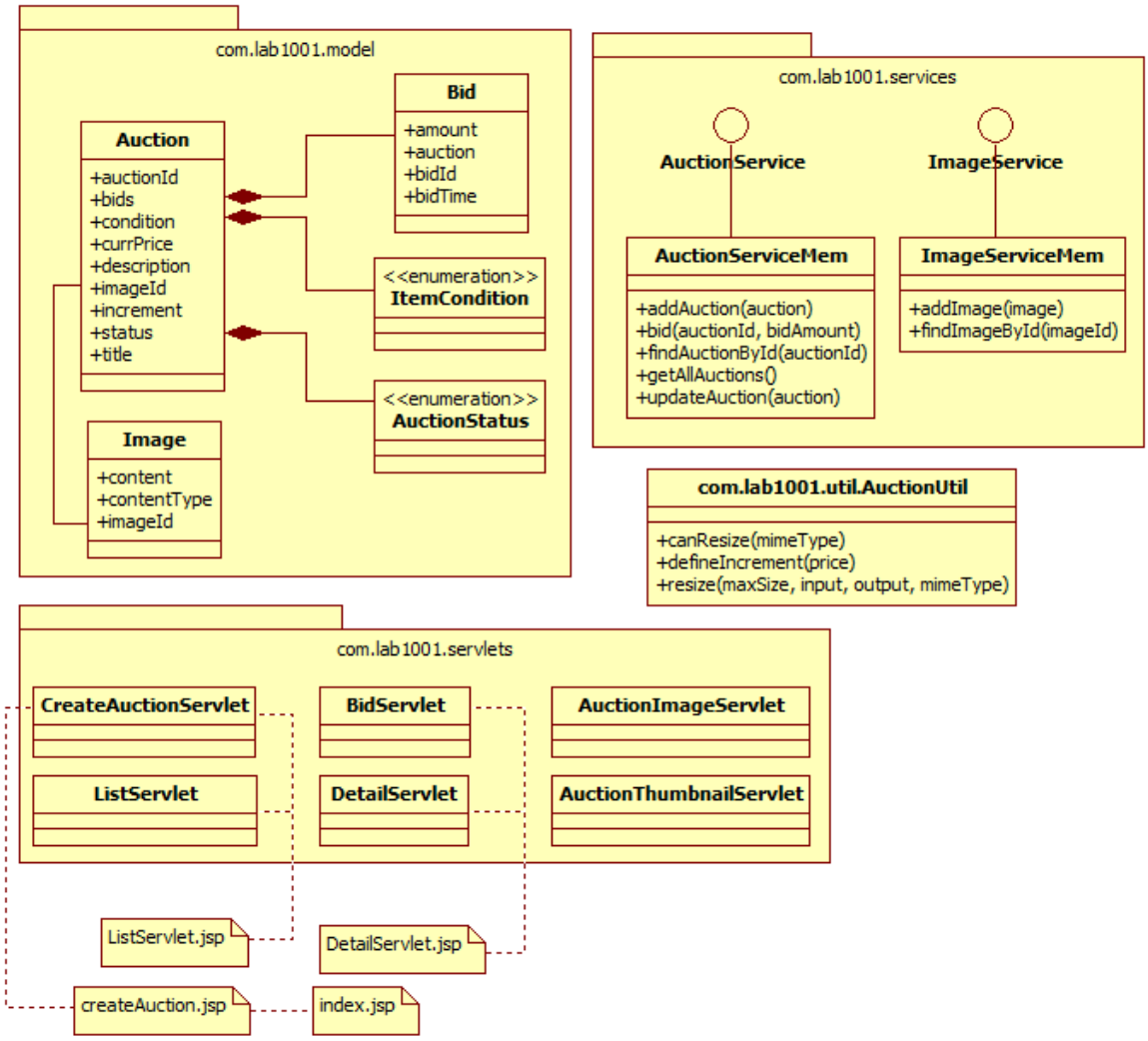
Overview

In this practice, you use the servlet 3.0 `MultipartConfig` feature to add pictures to the auctions added in the application.

You develop this practice using the auctioning application from previous practices. For your convenience and to minimize time setting up the entire application domain, you use an existing project that you modify to complete this practice.

The Auction Project for This Practice (lab_10_01)

The classes and pages available for this practice are outlined in the following diagram.



The application has the following features available; you may test it by running the application and browsing its pages:

- Create auctions (without an image)
- List all auctions
- See the detail of an auction
- Make a bid on an auction

Practice Overview

1. Open the `lab_10_01` project in `D:\labs\activities\`.
2. Run the project and test the functionality already built in. In this practice, you will modify:
   - The auction listing to add thumbnails for the pictures
   - The auction detail view to add a full size picture of the auctions
   - The create auction process to add the auction picture
3. Modify the `ListServlet.jsp` page replacing the `Thumbnail` string with:

```
<c:url var="thumbnailUrl" value="/AuctionThumbnailServlet">
  <c:param name="imageId" value="${auction.imageId}"/>
</c:url>
<img src="${thumbnailUrl}"/>
```

4. Modify the `com.lab1001.servlets.AuctionThumbnailServlet` Java servlet class (mapped to `/AuctionThumbnailServlet`). Inject the `ImageService` instance and add the following to the `doGet` method to render the thumbnail.

```
int imageId = Integer.parseInt(request.getParameter("imageId"));
Image image = imageService.findImageById(imageId);
response.setContentType(image.getContentType());
AuctionUtil.resize(64, image.getInputStream(),
response.getOutputStream(), image.getContentType());
```

5. Modify the `DetailServlet.jsp` page by replacing the `Image` string with:

```
<c:url var="ImageUrl" value="/AuctionImageServlet">
  <c:param name="imageId" value="${auction.imageId}"/>
</c:url>
<img src="${ImageUrl}" style="float: left"/>
```

6. Modify the `com.lab1001.servlets.AuctionImageServlet` Java servlet class. Inject the `ImageService` and get the image in the same manner you did with the `AuctionThumbnailServlet` and copy the `InputStream` image to the response `OutputStream`. Set the `contentType` of the response in the same way you did in `AuctionThumbnailServlet`.

   Use the following code to copy the streams:

```
OutputStream out = response.getOutputStream();
byte[] buffer = new byte[1024];
int readBytes = 0;
InputStream in = image.getInputStream();
while ((readBytes = in.read(buffer)) > 0) {
  out.write(buffer, 0, readBytes);
}
in.close();
out.close();
```

7. Modify the `createAuction.jsp` page by adding `enctype="multipart/form-data"` to the opening form tag and replacing the `Image input?` String with the following HTML input:

```
<input type="file" name="imageFile">
```

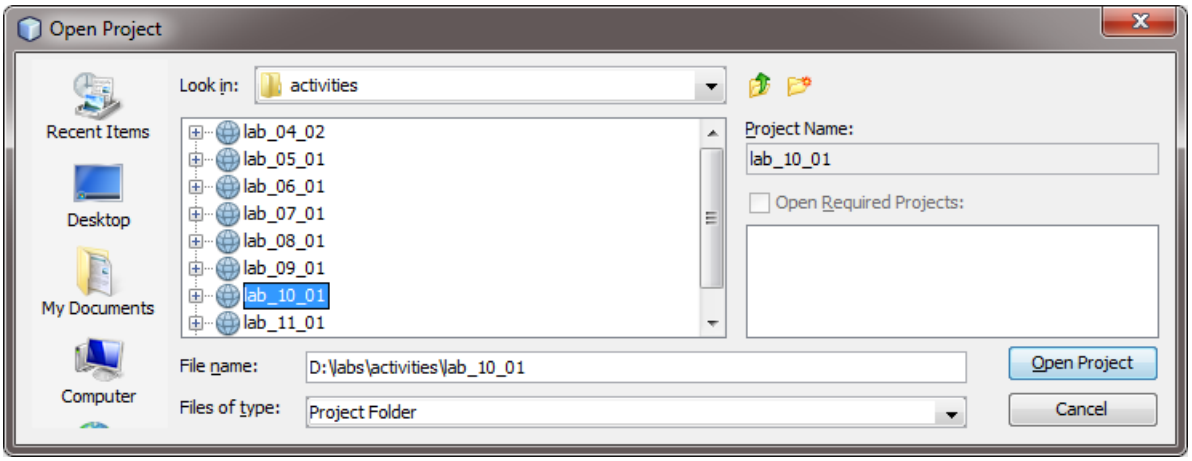
8. Modify the `com.lab1001.servlets.CreateAuctionServlet` adding the `@MultipartConfig` annotation.
9. Inject the `ImageService` in the servlet.
10. In the `doPost` method, create an image object and copy the uploaded file stream to the image output stream using the following code:

```
Part part = request.getPart("imageFile");
Image image = new Image();
OutputStream out = image.getOutputStream();
InputStream in = part.getInputStream();
byte buffer[] = new byte[4048];
int bytesRead;
while ((bytesRead = in.read(buffer)) > 0) {
  out.write(buffer, 0, bytesRead);
}
in.close();
out.close();
image.setContentType(part.getContentType());
imageService.addImage(image);
```

11. Set the `ImageId` in the auction to the ID of the `Image` object calling: `auction.setImageId(image.getImageId())` before the `auctionService.addAuction(auction)` method call.

Step-by-Step Instructions

1. Open the `lab_10_01` project in `D:\labs\activities\`.



2. Right-click the project name and select Run from the menu.
3. Test the application features. You will modify the following:
   - The auction listing to add thumbnails for the pictures
   - The auction detail view to add a full size picture of the auctions
   - The create auction process to add the auction picture

   The predefined auctions already contain images. The auction images are stored in memory and are manipulated using the `Image` model object and the `ImageService` service interface.

   The auction images are indexed by an ID this ID is stored in the auction.
4. Open `Web Pages` > `ListServlet.jsp`.
5. Locate the `Thumbnail` string and replace it with:

```
<c:url var="thumbnailUrl" value="/AuctionThumbnailServlet">
  <c:param name="imageId" value="${auction.imageId}"/>
</c:url>
<img src="${thumbnailUrl}"/>
```

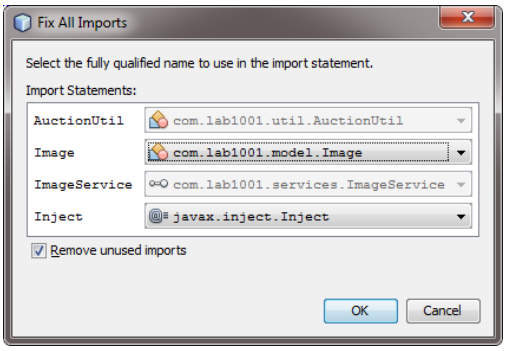
6. Open the `com.lab1001.servlets.AuctionThumbnailServlet` Java servlet class.
7. Inject the ImageService service instance adding the following code:

```
@Inject
private ImageService imageService;
```

8. Add the following code to the `doGet` method to render the thumbnail using this servlet.

```
int imageId = Integer.parseInt(request.getParameter("imageId"));
Image image = imageService.findImageById(imageId);
response.setContentType(image.getContentType());
AuctionUtil.resize(64, image.getInputStream(),
response.getOutputStream(), image.getContentType());
```

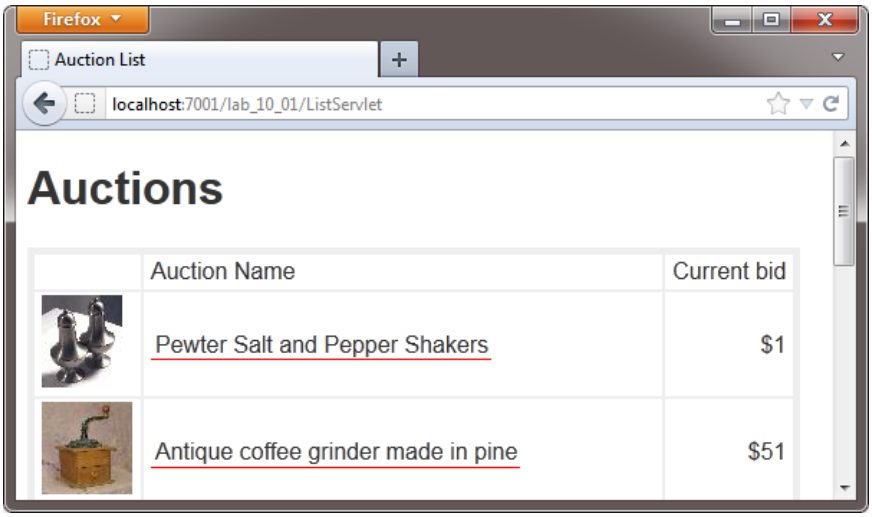
9. Fix imports by using the menu command or by pressing Ctrl + Shift + I.



Use the following code for reference:

```
@WebServlet(name = "AuctionThumbnailServlet",
        urlPatterns = {"/AuctionThumbnailServlet"})
public class AuctionThumbnailServlet extends HttpServlet {

  @Inject
  private ImageService imageService;

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    int imageId = Integer.parseInt(request.getParameter("imageId"));
    Image image = imageService.findImageById(imageId);
    response.setContentType(image.getContentType());
    AuctionUtil.resize(64, image.getInputStream(), response.getOutputStream(),
            image.getContentType());
  }
}
```

10. Run the application. Notice that the thumbnails are displayed for each auction.



11. Open `Web Pages` > `DetailServlet.jsp`.
12. Locate the `Image` string and replace it with the following code:

```
<c:url var="ImageUrl" value="/AuctionImageServlet">
  <c:param name="imageId" value="${auction.imageId}"/>
</c:url>
<img src="${ImageUrl}" style="float: left"/>
```

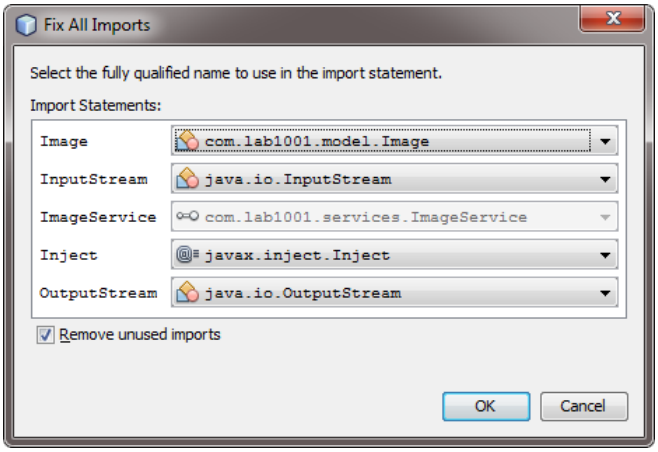
13. Open the `com.lab1001.servlets.AuctionImageServlet` class and add the following code to inject the `ImageService` instance:

```
@Inject
private ImageService imageService;
```

14. Add the following code to the `doGet` method to render the full-scale image using this servlet.

```
int imageId = Integer.parseInt(request.getParameter("imageId"));
Image image = imageService.findImageById(imageId);
response.setContentType(image.getContentType());
OutputStream out = response.getOutputStream();
byte[] buffer = new byte[1024];
int readBytes = 0;
InputStream in = image.getInputStream();
while ((readBytes = in.read(buffer)) > 0) {
  out.write(buffer, 0, readBytes);
}
in.close();
out.close();
```

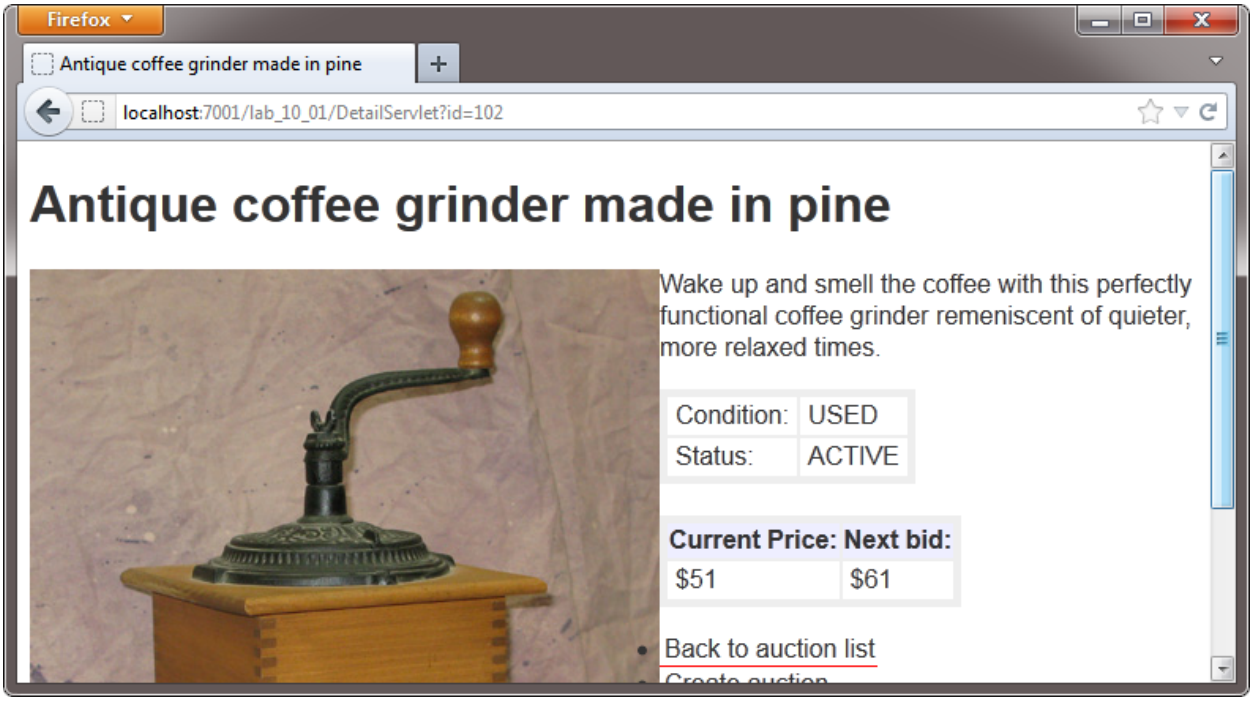
15. Fix imports in the code.



Use the following code for reference:

```
@WebServlet(name = "AuctionImageServlet",
        urlPatterns = {"/AuctionImageServlet"})
public class AuctionImageServlet extends HttpServlet {

  @Inject
  private ImageService imageService;

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    int imageId = Integer.parseInt(request.getParameter("imageId"));
    Image image = imageService.findImageById(imageId);
    response.setContentType(image.getContentType());
    OutputStream out = response.getOutputStream();
    byte[] buffer = new byte[1024];
    int readBytes = 0;
    InputStream in = image.getInputStream();
    while ((readBytes = in.read(buffer)) > 0) {
      out.write(buffer, 0, readBytes);
    }
    in.close();
    out.close();
  }
}
```

16. Run the application, you should see full-scale images in the detail view of the auctions.



17. Open Web Pages > createAuction.jsp.
18. Locate the opening `form` tag and add `enctype="multipart/form-data"` so that the tag looks like:

```
<form action="${createURL}" enctype="multipart/form-data"
method="post">
```

19. Locate the `Image input?` string and replace it with the following HTML code:

```
<input type="file" name="imageFile">
```

20. Open the `com.lab1001.servlets.CreateAuctionServlet` Java servlet class.

    Notice that this servlet already adds auctions to the system the only thing missing is the image.
21. Add the `@MultipartConfig` annotation to the servlet class.
22. Inject the `ImageService` instance.

```
@Inject
private ImageService imageService;
```

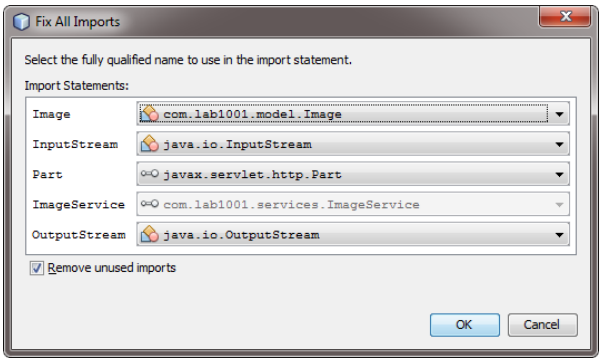
23. Add the following code to the beginning of the `doPost` method to create the `Image` object.

```
Part part = request.getPart("imageFile");
Image image = new Image();
OutputStream out = image.getOutputStream();
InputStream in = part.getInputStream();
byte buffer[] = new byte[4048];
int bytesRead;
while ((bytesRead = in.read(buffer)) > 0) {
  out.write(buffer, 0, bytesRead);
}
in.close();
out.close();
image.setContentType(part.getContentType());
imageService.addImage(image);
```

24. Add the following line before the `auctionService.addAuction(auction)` method call.

```
auction.setImageId(image.getImageId());
```

25. Resolve any import problems by using the Fix Imports command.



Use the following code for reference:

```
@WebServlet(name = "CreateAuctionServlet", urlPatterns = {"/CreateAuctionServlet"})
@MultipartConfig
public class CreateAuctionServlet extends HttpServlet {

  @Inject
  private AuctionService auctionService;
  @Inject
  private ImageService imageService;

  @Override
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    Part part = request.getPart("imageFile");

    Image image = new Image();
    OutputStream out = image.getOutputStream();
    InputStream in = part.getInputStream();

    byte buffer[] = new byte[4048];
    int bytesRead;
    while ((bytesRead = in.read(buffer)) > 0) {
      out.write(buffer, 0, bytesRead);
    }
    in.close();
    out.close();

    image.setContentType(part.getContentType());

    imageService.addImage(image);

    String auctionTitle = request.getParameter("auctionTitle");
    String auctionDescription = request.getParameter("auctionDescription");

    Auction auction = new Auction();
    auction.setTitle(auctionTitle);
    auction.setDescription(auctionDescription);
    auction.setImageId(image.getImageId());
    auctionService.addAuction(auction);
    request.setAttribute("message", "Auction created");
    request.getRequestDispatcher("/ListServlet").forward(request, response);
  }
}
```

26. Run the application and add an auction. You may use the images in `D:\labs\resources\images`.

    In this application, all the storage is handled in memory. If you restart the application, all the auctions will be back to their start state.

    You may keep exploring and modify the project to experiment with it.

    You can find more examples related to this practice and lesson by opening and running the project in: `D:\labs\examples\lesson10` using NetBeans.

Practices for Lesson 11: Security

Chapter 11

Practices for Lesson 11

Practices Overview

In these practices, you will implement web-application security restrictions in the auction web application.

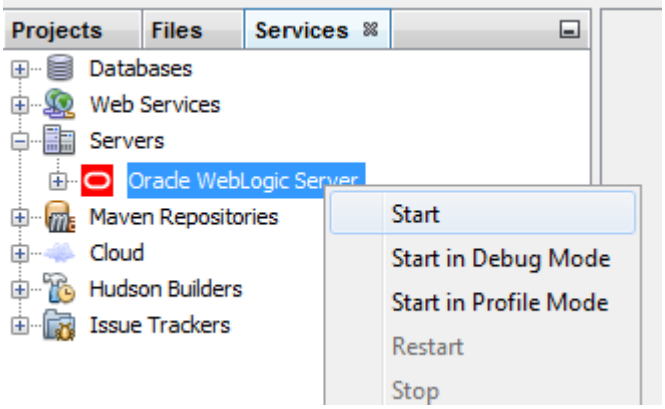
Practice 11-1: Configuring WebLogic Security Realms

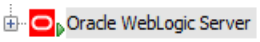
Overview

In this practice, you configure a security realm in WebLogic to enable web applications to use it to authenticate users.

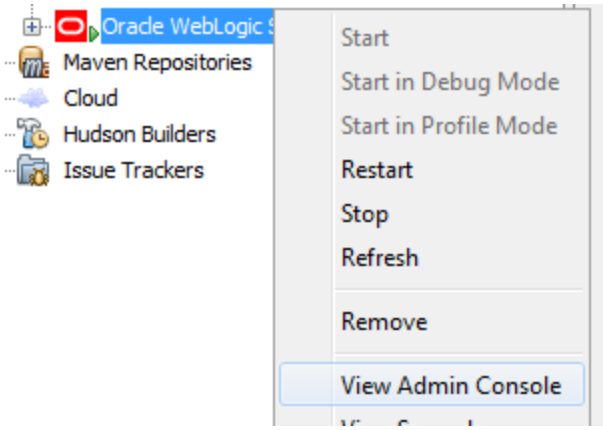
Step-by-Step Instructions

1. In NetBeans, open the Services tab and expand the Servers node.
2. Right-click Oracle WebLogic Server and select Start, to start the server.



3. Wait until the icon changes to display a green arrow. 

4. Right-click the Oracle WebLogic Server again and select View Admin Console.

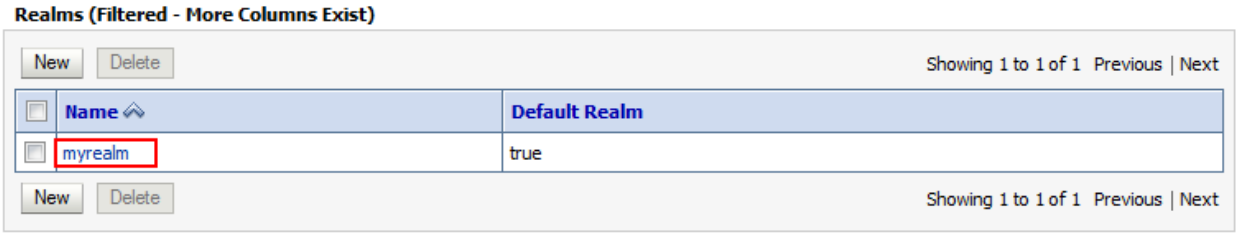


A browser opens displaying the WebLogic Console login page.

5. Log in using the username `weblogic` and the password `welcome1`.

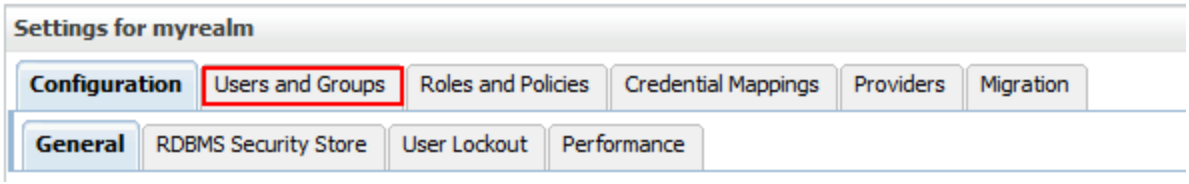
6. Click the Security Realms link.

7. Click the `myrealm` link.

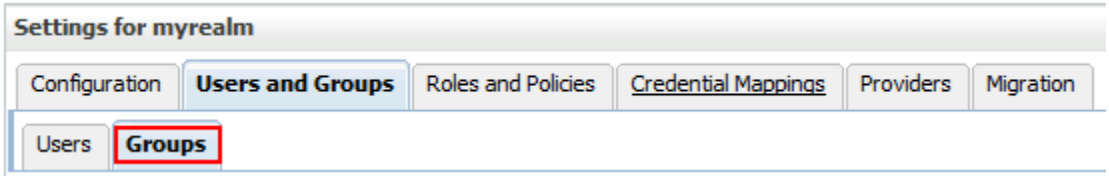


You use this page to create and configure the security realms in your server. You can share the same security realm in multiple applications. In this case, you add groups and users to the default security realm.

8. On the “Settings for myrealm” page, click the “Users and Groups” tab.

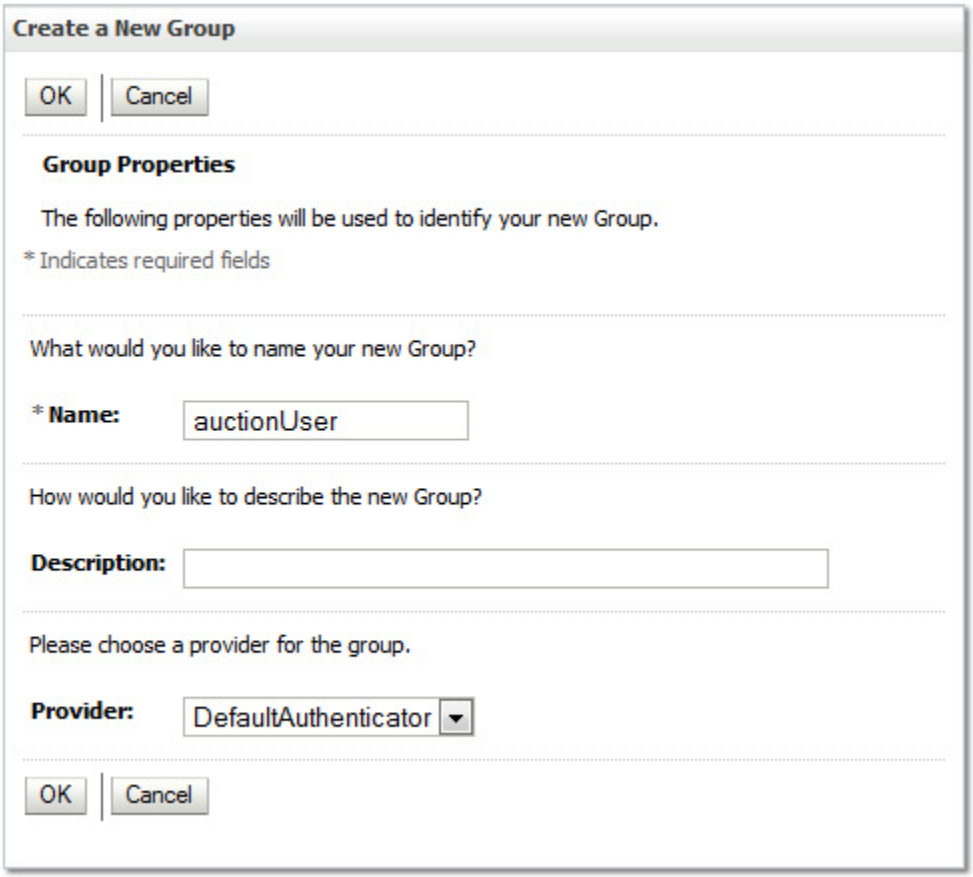


9. Click the Groups tab.

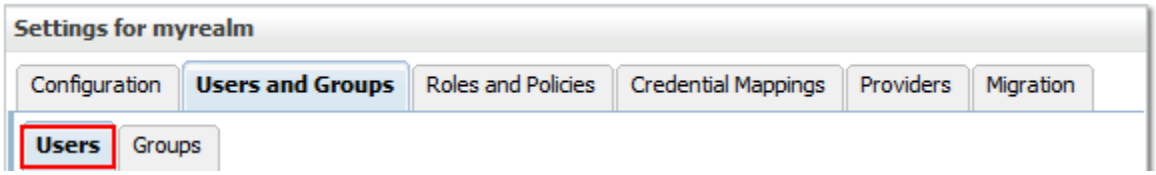


10. Click the New button to create a new user group.

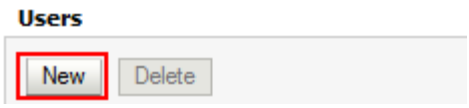
11. Set the group name to `auctionUser`, leave the description empty, and select `DefaultAuthenticator` for the provider. Click OK.



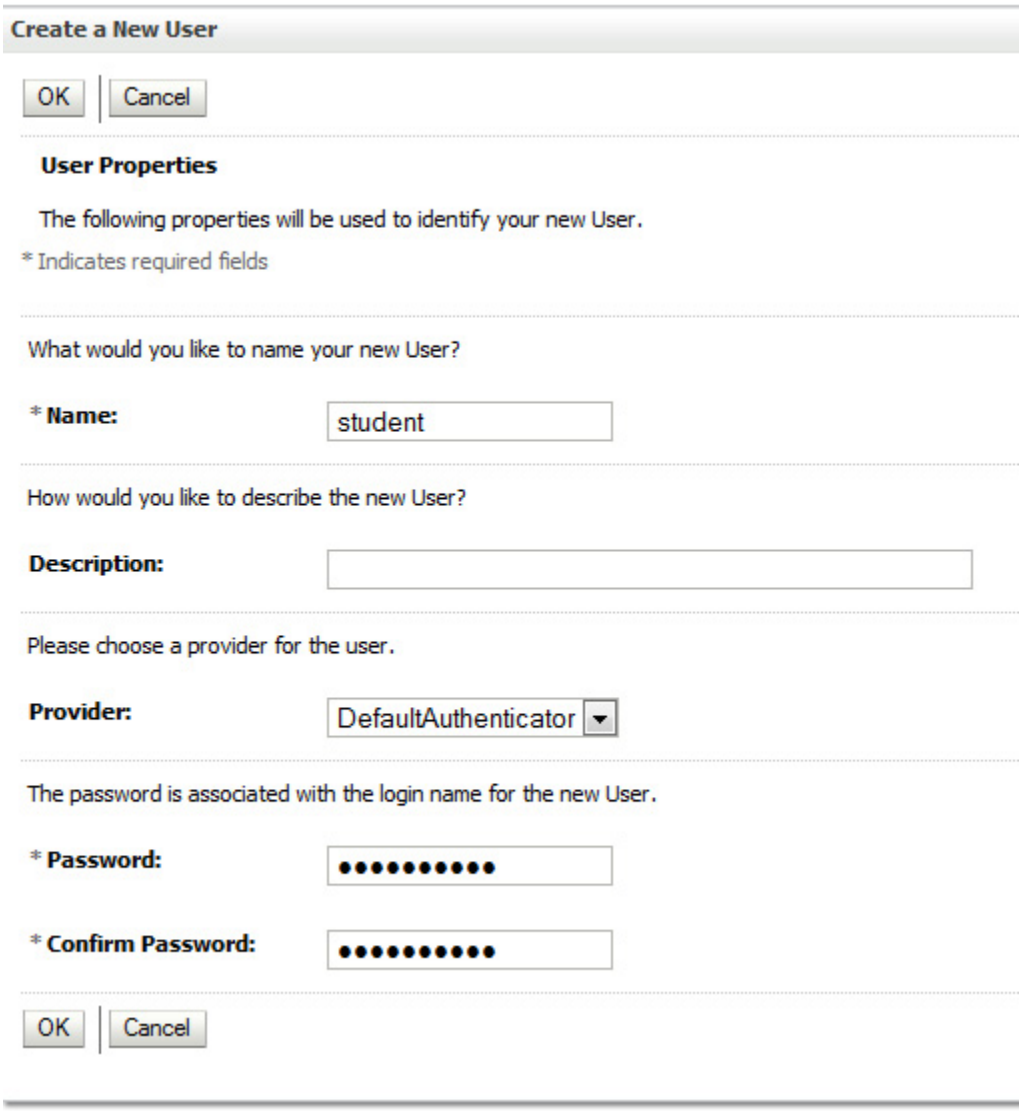
12. Click the Users tab.



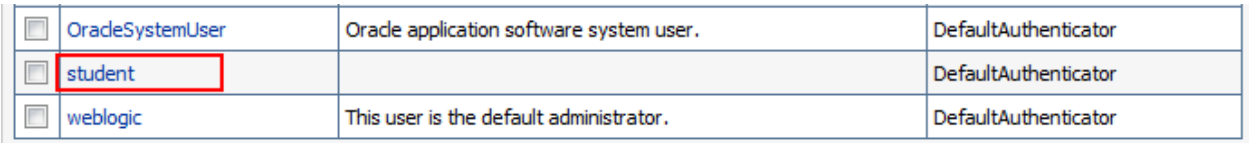
13. Click the New button to create a new user.



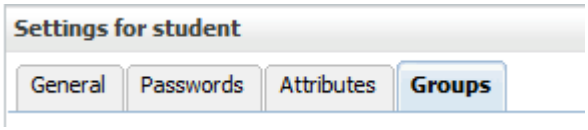
14. Enter a name and description for the user, select `DefaultAuthenticator` as the provider, and set a password and confirm it. Click OK.



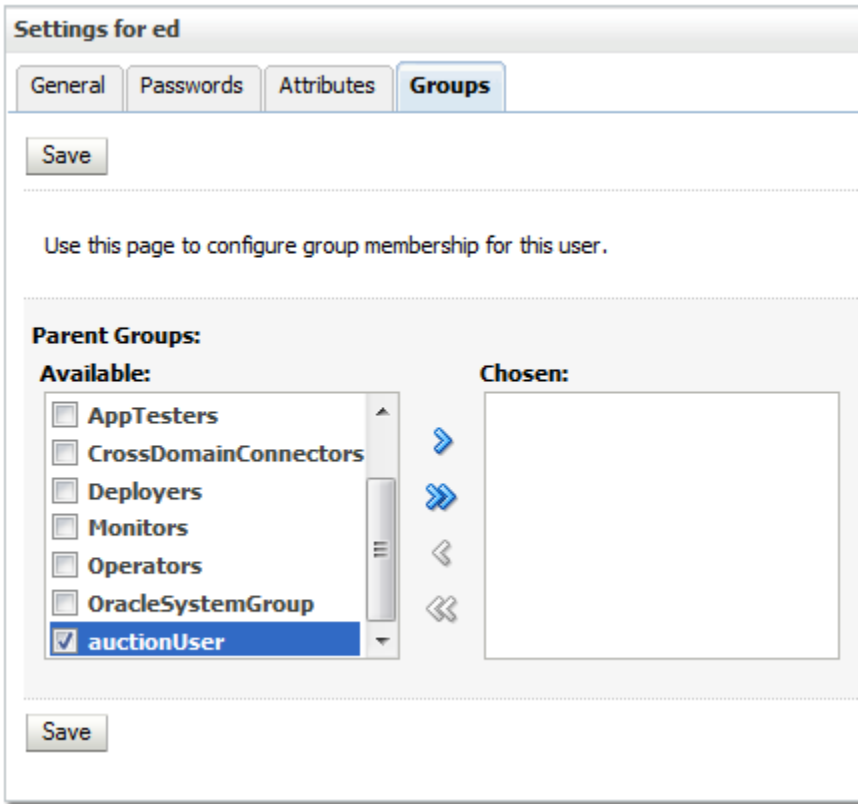
15. Click the newly created user.



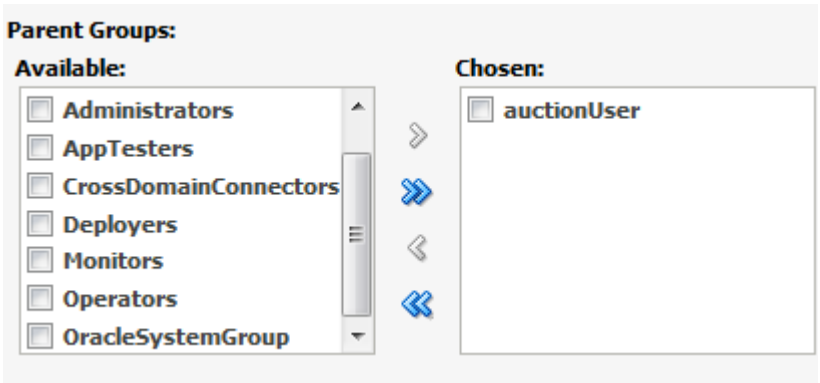
16. Click the Groups tab.



17. Select the auctionUser group and click the right blue single arrow to move auctionUser to the Chosen list.
18. Click Save.



>



You may add more users and groups to use in this practice.

Practice 11-2: Configuring Web-Application Security

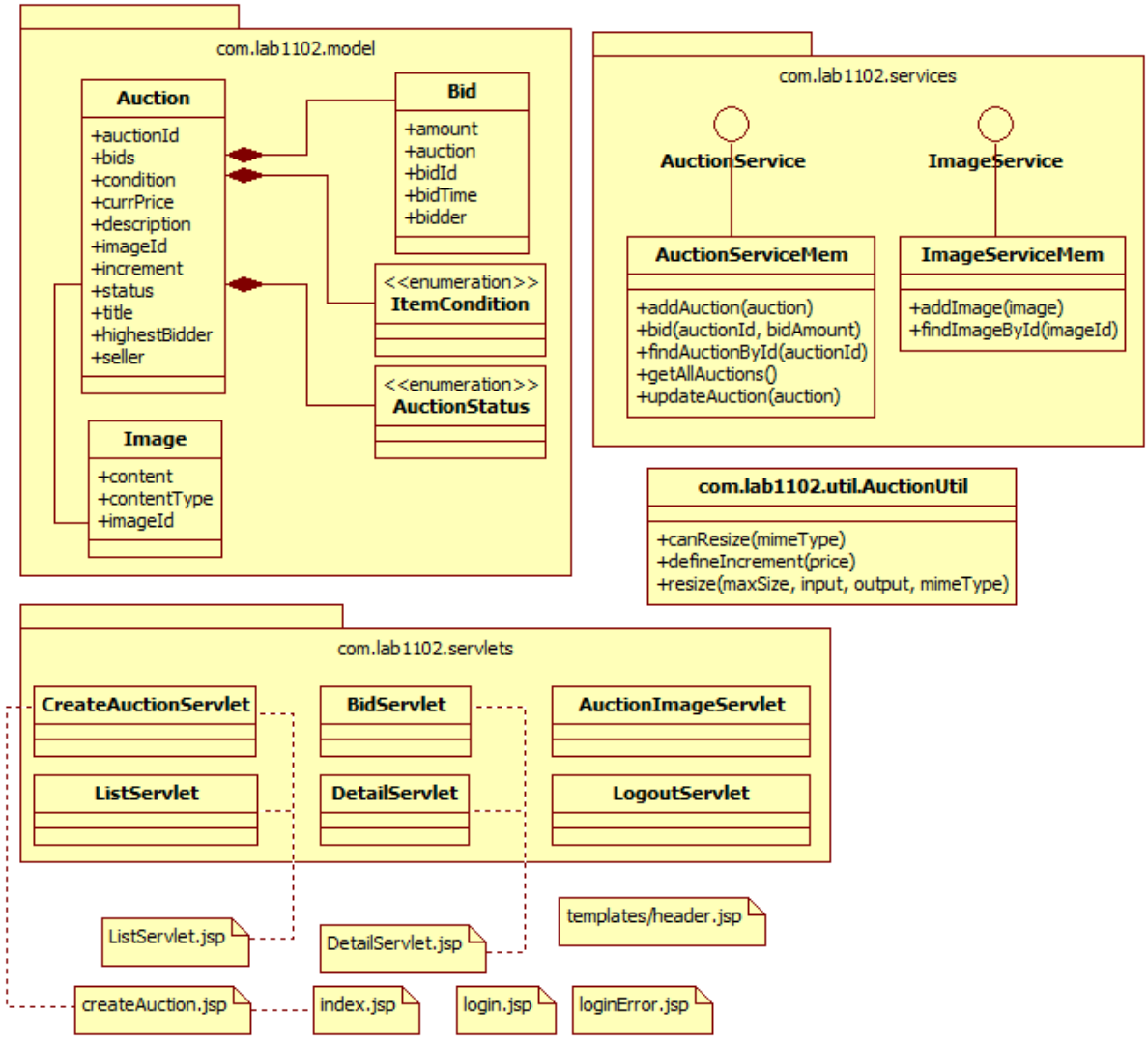
Overview

In this practice, you configure the auction application to use the security realm from the previous practice to authorize users.

You develop this practice using the application from previous lessons. For your convenience and to minimize application setup and problems, you modify an existing project to complete the practice.

The Auction Project for This Practice (lab_11_02)

The classes and pages available for this practice are outlined in the following diagram.



This practice has an intermediate level of difficulty, more detailed than the practice overviews and less detailed than the step-by-step instructions of earlier practices. You are encouraged to experiment with the source code to create new behavior in the application.

Practice Tasks

1. Open the project `lab_11_02` in `D:\labs\activities\`.
2. Run the application and explore its contents. Pay special attention to the pages that you want to secure for authenticated users. Use the table in the practice overview introduction for reference.
3. Open the `weblogic.xml` configuration file in Web Pages > `WEB-INF/weblogic.xml` (or you may use Configuration Files > `weblogic.xml`).
4. Add the following code after the `<context-root>` element to map WebLogic user groups or individual users to web-application security roles.

```
<security-role-assignment>
  <role-name>user</role-name>
  <principal-name>auctionUser</principal-name>
</security-role-assignment>
```

5. Open the Web Pages > `WEB-INF/web.xml` file (or Configuration Files > `web.xml`).
6. Add the following XML element to add the security role to the application.

```
<security-role>
  <role-name>user</role-name>
</security-role>
```

7. Add the following to configure the login pages and security realm.

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>myrealm</realm-name>
  <form-login-config>
    <form-login-page>/login.jsp</form-login-page>
    <form-error-page>/loginError.jsp</form-error-page>
  </form-login-config>
</login-config>
```

8. Configure the security constraint for the create auction page by adding the following:

```
  <security-constraint>
    <display-name>create auction</display-name>
    <web-resource-collection>
      <web-resource-name>createAuction</web-resource-name>
      <url-pattern>/createAuction.jsp</url-pattern>
      <url-pattern>/CreateAuctionServlet</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>user</role-name>
    </auth-constraint>
  </security-constraint>
```

   The login and login error pages are already created, but the `login.jsp` page requires proper configuration on the HTML login form to make it work properly.
9. Open the Web Pages > `login.jsp` file.

10. Locate the form HTML element and add `action="j_security_check"` to the opening tag.
11. Add names to the elements so that the user input is named `j_username` and the password input is named `j_password`. Use the following code for reference (Only the form code is shown.):

```
<form action="j_security_check" method="POST">
  User:
  <input type="text" name="j_username">
  Password:
  <input type="password" name="j_password">
  <input type="submit" value="login">
</form>
```

12. Save the files, run the application, and try to create an auction. You are prompted to log in before adding the auction. Use the username and password that you set on Practice 11-1.

    When you create an auction, you need the username to set it up in the seller property of the auction.
13. Open the `com.lab1102.servlets.CreateAuctionServlet` Java file.
14. Locate the `//Set user as highest bidder and seller` comment and add the following code after it.

```
String user = request.getRemoteUser();
auction.setHighestBidder(user);
auction.setSeller(user);
```

    `request.getRemoteUser()` will return the username with which the user authenticated.
15. Run the application again and create an auction. You should see your username posted in the auction.
16. Bidding on an auction requires authentication, too. Open the `com.lab1102.servlets.BidServlet` Java file.

    This servlet uses programmatic security instead of declarative (annotations or `web.xml`).
17. Locate the `boolean authorized = false` line. Add the following above the declaration to ensure that the user is prompted to log in when he tries to bid:

```
if(!request.authenticate(response)){
  return;
}
```

18. Replace `boolean authorized = false;` with:

```
boolean authorized = request.isUserInRole("user");
```

19. Replace the `String user = null;` with:

```
String user = request.getRemoteUser();
```

Use the following code for reference.

```
@WebServlet(name = "BidServlet", urlPatterns = {"/BidServlet"})
public class BidServlet extends HttpServlet {

  @Inject
  private AuctionService auctionService;

  @Override
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    if (request.getParameter("id") == null) {
      response.sendRedirect(getServletContext().getContextPath() + "/ListServlet");
      return;
    }

    int auctionId = Integer.parseInt(request.getParameter("id"));
    float bidAmount = Float.parseFloat(request.getParameter("bidAmount"));

    // Get user and verify roles.
    if (!request.authenticate(response)) {
      return;
    }

    boolean authorized = request.isUserInRole("user");
    String user = request.getRemoteUser();

    if (authorized) {
      String result = auctionService.bid(auctionId, user, bidAmount);
      request.setAttribute("message", result);
    } else {
      request.setAttribute("message", "User not authorized.");
    }
    request.getRequestDispatcher("/DetailServlet").forward(request, response);
  }
}
```

20. Open Web Pages > `templates/header.jsp`.
21. Add the following code to the file to add the logout link in all the pages that use the template:

```
<c:if test="${not empty pageContext.request.remoteUser}">
  <div style="text-align: right">
    <c:url value="/LogoutServlet" var="URLLogout"/>
    <a href="${URLLogout}">LOGOUT</a>
  </div>
</c:if>
```

22. Open the `com.lab1102.servlets.LogoutServlet` Java file.
23. Add the following code to the `doGet` method to do a logout.

```
    try {
      request.logout();
      request.getSession().invalidate();
    } finally {
      response.sendRedirect(getServletContext().getContextPath()
+ "/index.jsp");
    }
```

24. Run the application and test all the security features that you added.

Security might be also configured using annotations. Experiment by adding the following annotation to any of the servlets available and see what happens:

```
@ServletSecurity(@HttpConstraint(rolesAllowed={"user"}))
```

You can find more examples related to this practice and lesson by opening and running the project in: `D:\labs\examples\lesson11` using NetBeans. Use Practice 11-1 to create the users and groups required by the example project.

Practices for Lesson 12: Database Integration

Chapter 12

Practices for Lesson 12

Practices Overview

In these practices, you will integrate the auction application with a database by using the Java Persistence API (JPA).

After the first practice, additional practices are presented for you to expand on topics that you may want to reinforce. These optional activities have solutions in the `D:/labs/solutions/lab_12_01` folder.

Practice 12-1: Integrating the Auction Application by Using JPA for Database Access

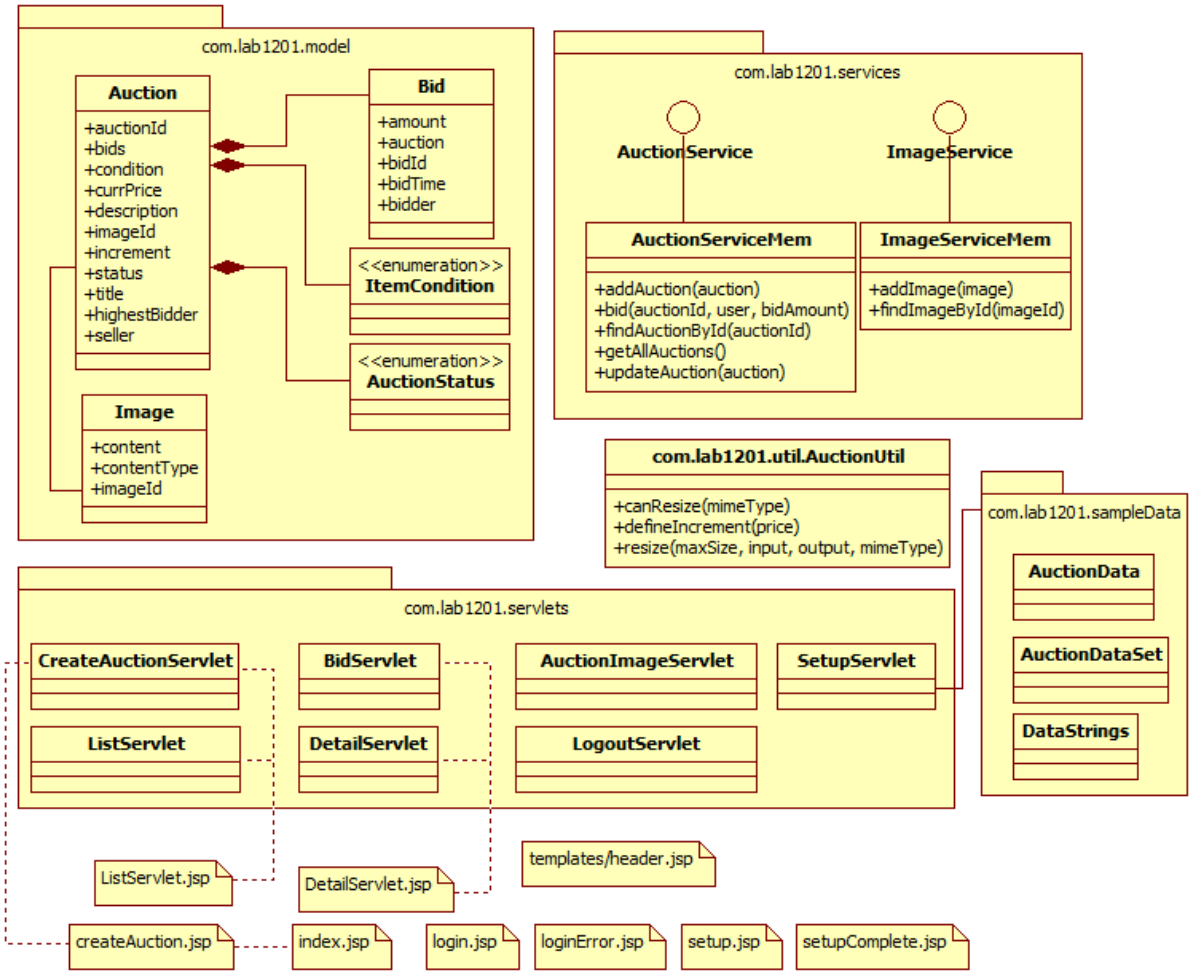
Overview

In this practice, you configure the auction application to integrate it with a database using JPA. You have to create the database and the tables to store the objects and then map objects as entities to persist them in the database tables.

You develop this practice using the application from previous lessons. For your convenience and to minimize application setup and problems, you modify an existing project to complete the practice.

The Auction Project for This Practice (lab_12_01)

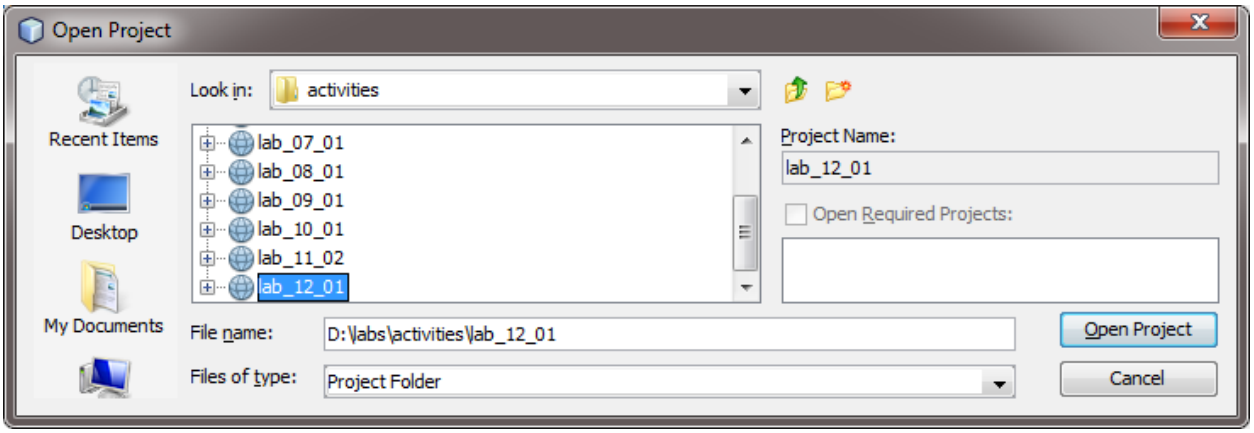
The classes and pages available for this practice are outlined in the following diagram.



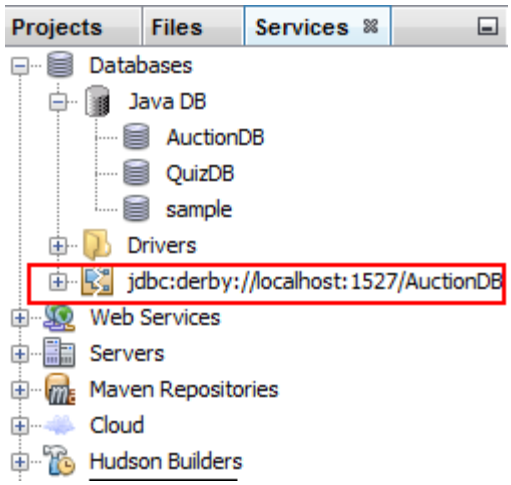
This practice has an intermediate level of difficulty, more detailed than the practice overviews and less detailed than the step-by-step instructions of earlier practices. You are encouraged to experiment with the source code to create new behavior in the application.

Practice Tasks

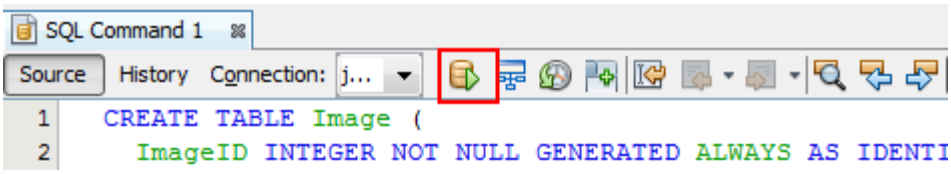
1. Open the `lab_12_01` project in `D:\labs\activities\`.



2. Run the project and test the application functionality. To generate the default auctions, click the `Create Default Data` link. You can create and bid on auctions, but all the data is in memory and restarting the application (clicking Run again in NetBeans) will reset the application data to its initial state.
3. Look at the `com.lab1201.servlets.SetupServlet` code. Notice how it creates the auctions and bids using the services in the com.lab1201.services package. Modifying the service's implementation to use the database will complete the integration with the database.
4. Open the `Services` tab in NetBeans and open the `Databases` node.
5. Right-click `Java DB` and select `Start Server` to start the database server.
6. Right-click Java DB, select Create Database, and use the following database details to create the database:
   - Database Name: `AuctionDB`
   - Username: `oracle`
   - Password: `welcome1`
7. Right-click the new AuctionDB database and select Connect. A new node `jdbc:://derby://localhost:1527/AuctionDB` is created.



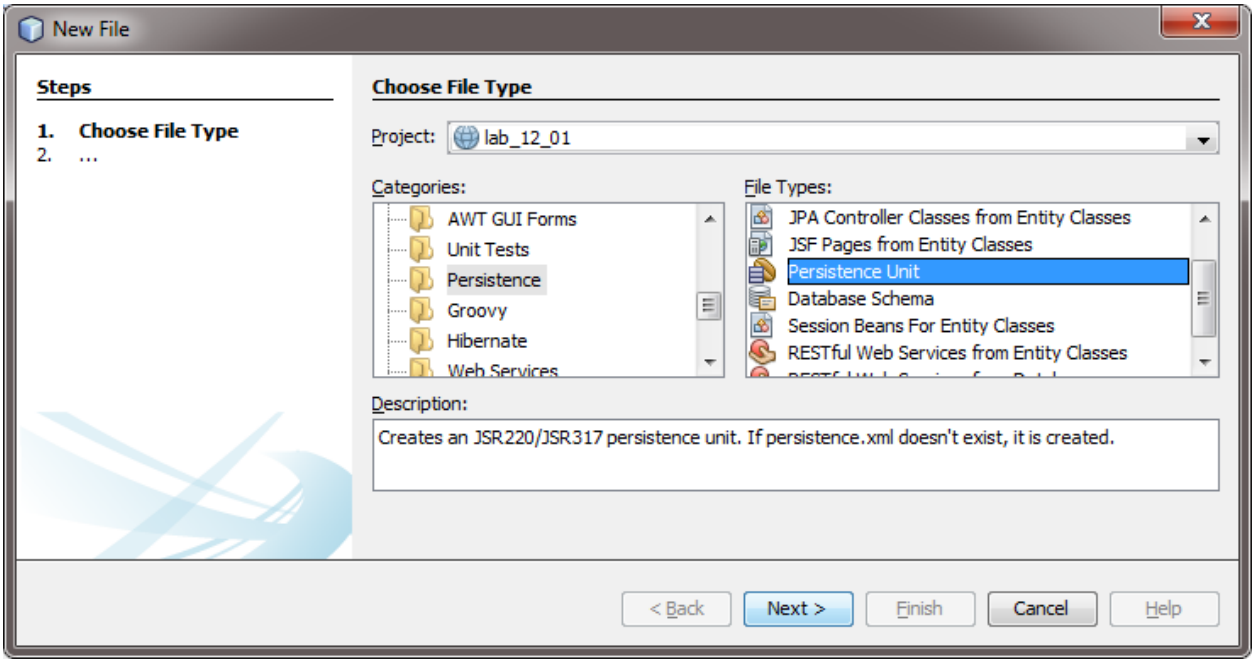
8. Right-click the newly created node and select Execute Command. Execute the following SQL script on the database by typing it in the editor and clicking the Run SQL button or by pressing Ctrl + Shift + E).



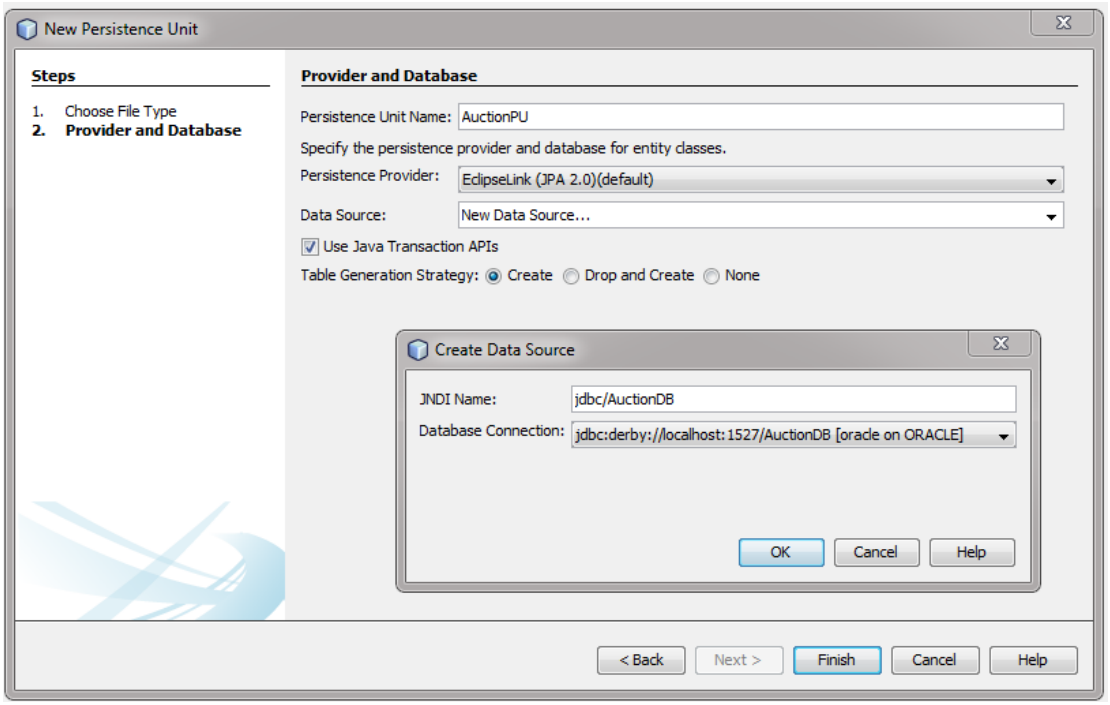

```
CREATE TABLE Image (
  ImageID INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,
  Content BLOB NOT NULL,
  ContentType VARCHAR(50) NOT NULL,
  PRIMARY KEY (ImageID)
);

CREATE TABLE Auction (
  AuctionID INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,
  ImageID INTEGER NOT NULL,
  Title VARCHAR(60),
  Description VARCHAR(1000),
  Seller VARCHAR(20) NOT NULL,
  HighestBidder VARCHAR(20) NOT NULL,
  CurrPrice REAL NOT NULL,
  Increment REAL NOT NULL,
  Status VARCHAR(15) NOT NULL,
  ItemCondition VARCHAR(15) NOT NULL,
  Version INTEGER NOT NULL,
  PRIMARY KEY (AuctionID)
);

CREATE TABLE Bid (
  BidID INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,
  Bidder VARCHAR(40) NOT NULL,
  AuctionID INTEGER NOT NULL,
  Amount REAL NOT NULL,
  BidTime TIMESTAMP NOT NULL,
  PRIMARY KEY (BidID)
);
```

This SQL script generates tables that map the Auction, Bid, and Image objects to the database.

9. Go to the Projects tab, where you will create a persistence unit to save the objects in the database. To create the persistence unit, right-click the project and select New > Other, and then, under the Persistence category, create a persistence unit.



10. Name the persistence unit `AuctionPU`. From the Data Source drop-down list, select New Data Source. A popup window will appear. Name the JNDI resource `jdbc/AuctionDB` and select the `AuctionDB` connection from the Database Connection selector.



11. Accept all dialog boxes by clicking OK. The `persistence.xml` configuration will be created, you may open it in the Projects tab > lab_12_01 > Configuration Files > `persistence.xml`.

    You can verify that the persistence unit and data source, two new files `lab_12_01/src/conf/persistence.xml` and `lab_12_01/setup/datasource-1.jdbc.xml`, are created. The `persistence.xml` file configures JPA and the `datasource-1.jdbc.xml` file is used by NetBeans to create the JNDI resource in WebLogic server when the application is deployed.
12. You have to make all the persisted classes entities. Open the `com.lab_12_01.model.Image` class.
13. Add the `@Entity` annotation to the class and annotate the image ID instance variable with:

Variable	Annotations
imageId	`@GeneratedValue(strategy = GenerationType.IDENTITY)` `@Id`

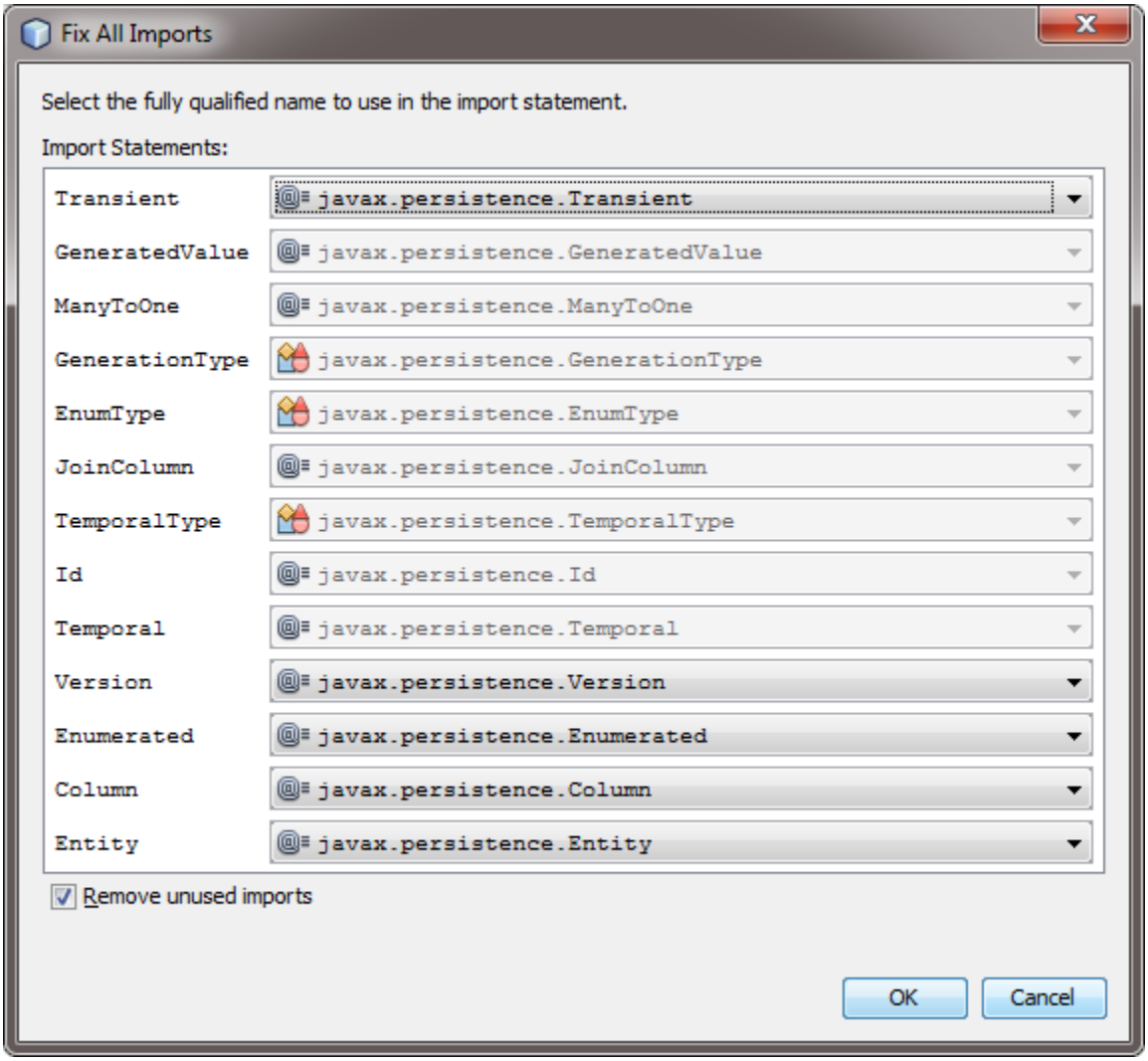
14. In the `com.lab_12_01.model.Bid` class, add the `@Entity` annotation and annotate the class instance variables using the following table:

Variable	Annotations
bidId	`@GeneratedValue(strategy = GenerationType.IDENTITY)` `@Id`
auction	`@ManyToOne` `@JoinColumn(name = "AUCTIONID")`
bidTime	`@Temporal(TemporalType.TIMESTAMP)`

15. In the `com.lab_12_01.model.Auction` class, add the `@Entity` annotation and annotate the class instance variables using the following table:

Variable	Annotations
auctionId	`@GeneratedValue(strategy = GenerationType.IDENTITY)` `@Id`
status	`@Enumerated(EnumType.STRING)`
condition	`@Enumerated(EnumType.STRING)` `@Column(name = "ITEMCONDITION")`
bids	`@OneToMany(mappedBy = "auction")`
version	`@Version`

16. Add the proper imports to the modified classes (NetBeans menu > Source > Fix Imports).

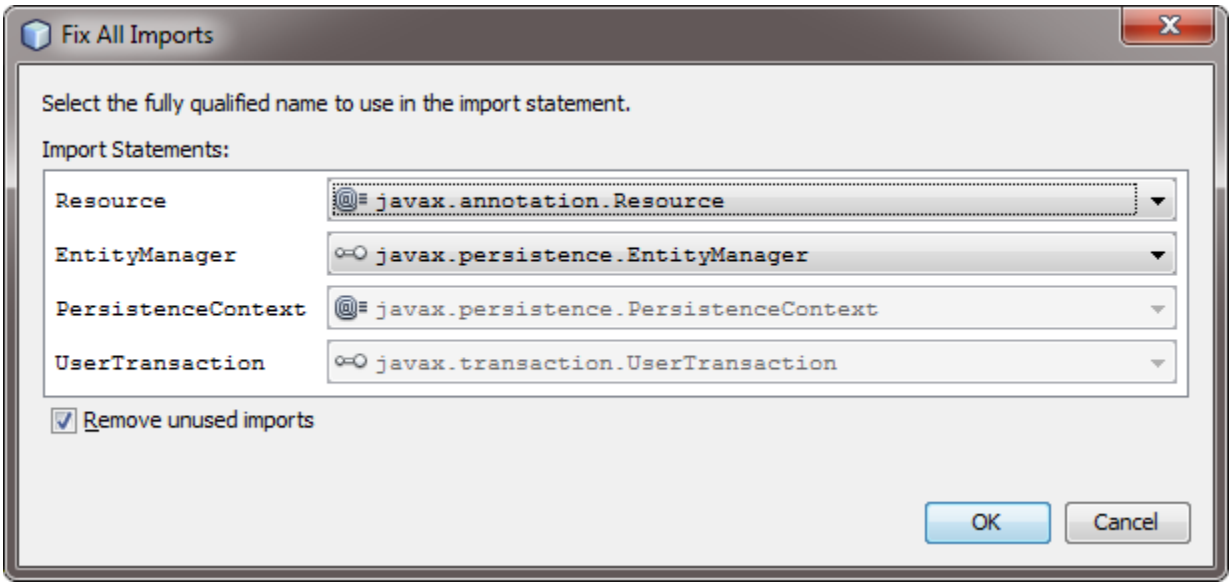


You can use the solution files in `D:\labs\solutions\lab_12_01` if you need some reference on the Auction, Bid, and Image Java files.

17. Open the `com.lab1201.services.impl.ImageServiceImpl` class and add the following to it to inject the `PersistenceUnit` along with the `UserTransaction` object. You need to modify only the service implementation to use the database.

```
@PersistenceContext(unitName = "AuctionPU")
private EntityManager em;
@Resource
private UserTransaction utx;
```

18. Fix imports.



19. Locate the `findImageById` and replace its contents with the following JPA code:

```
Image image = null;
try {
  image = em.find(Image.class, imageId);
} catch (Exception e) {
  LOG.log(Level.SEVERE, e.getMessage());
}
return image;
```

20. Locate the `addImage` method and replace its contents with the following JPA code:

```
try {
  utx.begin();
  em.persist(image);
  utx.commit();
  return image;
} catch (Exception ex) {
  Logger.getLogger(ImageServiceImpl.class.getName()).log(
Level.SEVERE, null, ex);
  throw new RuntimeException(ex);
}
```

21. To clean up the class, remove the `images` and `currentImageId` instance variables and any references to them.

    You may test the application now. The application works just as it used to, but with the difference that images are stored in the database.

22. Open the `com.lab1201.services.impl.AuctionServiceImpl` class and add the following to it to inject the `PersistenceUnit` along with the `UserTransaction` object. Remember to fix imports after adding the code.

```
@PersistenceContext(unitName = "AuctionPU")
private EntityManager em;
@Resource
private UserTransaction utx;
```

23. Locate the `getAllAuctionsMethod` and replace its contents with the following JPA code:

```
Query query = em.createQuery("Select a FROM Auction a");
return query.getResultList();
```

Alternatively, you may use the following code (using JPA Criteria)

```
try {
  CriteriaBuilder cb = em.getCriteriaBuilder();
  CriteriaQuery<Auction> cq = cb.createQuery(Auction.class);
  Root<Auction> auction = cq.from(Auction.class);
  cq.select(auction);
  cq.orderBy(cb.asc(auction.get("auctionId")),
cb.asc(auction.get("currPrice")));
  TypedQuery<Auction> auctionQuery = em.createQuery(cq);
  return auctionQuery.getResultList();
} catch (Exception e) {
  LOG.log(Level.SEVERE, e.getMessage());
}
return Collections.emptyList();
```

24. Locate `addAuction` and `findAuctionById`. These methods are be modified in the same manner that you used to modify the `ImageServiceImpl` methods. Try to figure out how to modify them. If you need more reference, the modifications are provided here:

`addAuction` method:

```
try {
  utx.begin();
  em.persist(auction);
  utx.commit();
  return auction;
} catch (Exception e) {
  LOG.log(Level.SEVERE, e.getMessage());
  throw new RuntimeException(e);
}
```

`findAuctionsById` method:

```
Auction auction = null;
try {
  auction = em.find(Auction.class, auctionId);
} catch (Exception e) {
  LOG.log(Level.SEVERE, e.getMessage());
}
return auction;
```

25. Locate the `updateAuction` method and replace its contents with:

```
try {
  utx.begin();
  int auctionId = auction.getAuctionId();
  em.find(Auction.class, auctionId);
  auction = em.merge(auction);
  utx.commit();
  return auction;
} catch (Exception ex) {
  Logger.getLogger(AuctionServiceImpl.class.getName()).log(
Level.SEVERE, null, ex);
  throw new RuntimeException(ex);
}
```

The `bid` method does not require any modification, because adding a bid is adding a `Bid` object to the auction and when an auction is updated, the list of bids for that auction is persisted as well.

26. To clean up the class, remove the `auctions, bids, currentAuctionId,` and `currentBidId` instance variables and any references to them.
27. Run the application again and notice how the data is persisted even when the application restarts, because it is stored in the database now.
28. Test the application: Create auctions, create the default data, bid on auctions, and use different users (create more users using lab_11_02). This completes this practice.

Additional Information

Running the Example Project

The example for this lesson contains a complete project that makes use of every feature explained on the course.

Before opening the example project, you must create a database. Follow steps 4 to 7 of Practice 12-1 and replace the step 6 settings with:

- Database Name: `QuizDB`
- Username: `oracle`
- Password: `welcome1`

You must also follow the setup instructions from the examples in Lesson 11, because the current example requires the security setup from the previous example.

You can open and run the example project. It contains a guided setup for the database tables and default data.

The example integrates all the topics in the course in a single application; you can look at the code, modify it, and figure out how it works.

Additional Practices

The next section presents you with challenges that you can do to practice previous topics. You are not required to finish them, but doing so may help you reinforce the concepts and topics of previous practices.

Practice 12-2: Challenge Practice: Implementing a Latest Bids View [Servlets and JSP]

Overview

You want to add a view to display the latest (or all) the bids done in the application.

Assumptions

You completed Practice 12-1.

It is recommended that you back up your completed practice in case you want to restart the practice.

Tasks

1. Create a new interface `BidService` in the `com.lab1201.services package`.
2. Add a method `List<Bid> getBids()` to the `BidService` interface.
3. Create a new class `BidServiceImpl` that implements the `BidService` interface in the `com.lab1201.services.impl` package, and add the `@ApplicationScoped` annotation to the class.
4. Add the JPA `EntityManager` and `UserTransaction` objects by using annotations.
5. Implement the `getBids` method with a simple JPA query: `Select b From Bid b`, and return the results of the query.
6. Create a new servlet `BidListServlet` in the `com.lab1201.servlets package`. If you create a new class instead of using the New Servlet wizard in NetBeans, extend `HttpServlet` and add the `@WebServlet` annotation.
7. Inject the `BidService` (remember to use the interface) by using the `@Inject` annotation.
8. Modify the `doGet` or `processRequest` method to store the results of `bidService.getBids()` in the request (write down the name of the attribute that you want to use), and then use the request to redispatch to the `bidList.jsp` page.
9. Create a new JSP `bidList.jsp` page to display the list of bids.
10. Use the `c:forEach` JSTL tag to iterate over the request attribute you stored in the `BidListServlet`.
11. Display the contents of each bid using Expression Language.
12. Modify the `index.jsp` file to add a link to the `BidListServlet`.

Practice 12-3: Challenge Practice: Extending the Setup to Create Tables [JDBC Integration]

Overview

You want to add SQL DDL statements in the setup service using JDBC with JNDI lookup or plain JDBC connection management.

When you click the Create Default Data link in the homepage of the application, it executes the `com.lab1201.servlets.SetupServlet`. If you want to create the tables as part of the process, you must add a service to do that and modify the `SetupServlet`.

Assumptions

You completed Practice 12-1.

It is recommended that you back up your completed practice in case you want to restart the practice.

Tasks

1. Create a new interface `SetupService` in the `com.lab1201.services` package.
2. Add a `void setup` method to the `SetupService` interface.
3. Create a class `SetupServiceImpl` that implements the `SetupService` interface in the `com.lab1201.services.impl` package.
4. Implement the setup method to drop existing tables and create them in the database. You can use the constants defined in `com.lab1201.sampleData.DataStrings` to drop and create the tables.
5. Use the following code to get a JDBC connection using JNDI:

```
javax.naming.Context ctx = new javax.naming.InitialContext();
javax.sql.DataSource ds = (javax.sql.DataSource)
ctx.lookup("jdbc/AuctionDB");
java.sql.Connection connection = ds.getConnection();
```

   Alternatively, you can obtain the connection using plain JDBC:

```
Class.forName("org.apache.derby.jdbc.ClientDriver")
.newInstance();
java.sql.Connection connection = DriverManager
.getConnection(DataStrings.URL, DataStrings.USER,
DataStrings.PASS);
```

6. Use the connection to create a statement and execute the SQL DDL statements:

```
java.sql.Statement statement = connection.createStatement();
try{ statement.execute(DataStrings.QUERY_DROP_BID);}
catch(Exception e){}
try{ statement.execute(DataStrings.QUERY_DROP_AUCTION);}
catch(Exception e){}
try{ statement.execute(DataStrings.QUERY_DROP_IMAGE);}
catch(Exception e){}
statement.execute(DataStrings.QUERY_CREATE_IMAGE);
statement.execute(DataStrings.QUERY_CREATE_AUCTION);
statement.execute(DataStrings.QUERY_CREATE_BID);
connection.commit();
```

   Refer to the `SetupServiceImpl.java` file in the solutions directory to see how to close JDBC connection and statements.
7. Open the `com.lab1201.servlets.SetupServlet` Java servlet and inject the `SetupService` interface.
8. Call the `setupService.setup` method at the beginning of the `processRequest` method.
9. Test your application. It should drop and create the tables when it creates the default data.

Practice 12-4: Challenge Practice: Creating a "Setup-Required" Home Page If the Database Does Not Exist [Listeners and Filters]

Overview

You want to detect whether the database has started and created correctly, and, if not, redirect the user to a page with setup instructions.

To do this you create a `ServletContextListener` to check whether a query can be done (using JPA or JDBC) and set a `ServletContext` attribute with the result. The filter will intercept all calls and redirect the user to an instruction page if setup is required. Both the listener and the filter can be configured by using annotations or the `web.xml` file.

Assumptions

You completed Practice 12-1.

It is recommended that you back up your completed practice in case you want to restart the practice.

Tasks

1. Create a new web application listener implementing the `ContextListener` interface, name it `DatabaseCheckContextListener`, and put it in the `com.lab1201.listeners` package.

   You can use the New Web Application Listener wizard in NetBeans or create a Java Class and implement the `ContextListener` interface.
2. Add the JPA `EntityManager` object using the `@PersistenceContext(unitName = "AuctionPU")` annotation to the listener.
3. Remove all existing code (if any) from the `contextInitialized` and `contextDestroyed` methods.
4. Modify the `contextInitialized` method. You want to check whether a query can be done and store the result in a context attribute.
5. Get the servlet context object from the `ServletContextEvent` object using `servletContextEvent.getServletContext()`.
6. Create a `boolean querySuccess` variable to store whether the query could be executed.
7. Add a `try/catch(Exception)` block and do a JPA query on the Auction object.
8. Set the `querySuccess` to true after the query and to false in the catch block.

9. Set the attribute `databaseReady` inside the `servletContext` object with the `querySuccess` value.

   Use the following code for reference:

```
ServletContext servletContext = servletContextEvent.getServletContext();
boolean querySuccess;
try {
  Query query = entityManager.createQuery("Select a FROM Auction a");
  query.getResultList();
  querySuccess = true;
} catch (Exception e) {
  querySuccess = false;
}
servletContext.setAttribute("databaseReady", querySuccess);
```

10. Create a new `WebListener` to intercept all the requests and if the context attribute `databaseReady` is false, redirect to a setup-required page.

    NetBeans has a wizard for filters, which inserts a lot of code. If you use the wizard, clean up all the class contents and re-implement all the filter interface methods with an empty body.

11. Create a new Java class `SetupCheckFilter` in the `com.lab1201.filter` package and implement the `javax.servlet.Filter` interface.
12. Add the `@WebFilter(filterName = "SetupCheckFilter", urlPatterns = {"/*"})` annotation to the class.
13. Modify the `doFilter` method to verify whether setup is needed.
14. Get the `ServletContext` from the request by using `request.getServletContext()`.
15. Get the `databaseReady` from the servlet context attribute using:
    `Boolean databaseReady = (Boolean) servletContext.getAttribute("databaseReady");`
16. If `databaseReady` is `not null` and is `true`, invoke the `chain.doFilter` method. Otherwise (`else`), use the request to redispatch to the `setupInstructions.jsp` page.
17. Create the `setupInstructions.jsp` page with the instructions from the `index.jsp` page.

    You can add a “Setup Required” header to the instructions page, but remember to remove all links from it.

    Additionally, you can remove the instructions from the `index.jsp` page because you added a setup check.

18. To test the application, you can either delete the database or stop the database server and restart the server.

    You may need to restart the server, because the connection is managed by WebLogic and stored in a JNDI resource. You can also use JDBC to validate the connection.

Practice 12-5: Challenge Practice: Adding a Role to Create Auctions [Security]

Overview

You want to add more security to the application by adding another role to the application that can create auctions.

Assumptions

You completed Practice 12-1.

It is recommended that you back up your completed practice in case you want to restart the practice.

Tasks

1. Start Oracle WebLogic Server if it is not running and open the Administration Console at: `http://localhost:7001/console/`
2. Log in with the username `weblogic` and password `welcome1`.
3. Open the `Security Realms` configuration.
4. Open the `myrealm` security realm.
5. Go to the `Users and Groups` tab and then click the `Groups` sub-tab.
6. Create a new user group by clicking the New button, and complete the form to create the `auctionCreators` group.
7. Go to the Users sub-tab and create a new user.
8. Click the newly created user and open the Groups tab.
9. Add the `auctionCreators` group and save the user.
10. Open NetBeans and, in the project, open the `weblogic.xml` configuration file.
11. Add another Security-role-assignment to the file to add the `auctionCreators` principal to the application. Use the following code for reference:

```
<security-role-assignment>
  <role-name>auctionCreators</role-name>
  <principal-name>auctionCreators</principal-name>
</security-role-assignment>
```

12. Open the `web.xml` configuration file and add another `security-role` element for the `auctionCreators` role.
13. In the `web.xml` file, locate the security constraint for `createAuction`.jsp and `/CreateAuctionServlet` and modify the `role-name` to be `auctionCreators`.
14. Run the application and test the new role by creating an auction.

    You may modify `CreateAuctionServlet` directly and add the `ServletSecurity` annotation and then remove the `Security-Constraint` from the `web.xml` file.