Web Component Development With Servlet and JSP™ Technologies

Activity Guide - Solaris

SL-314-EE6

D65271GC11

Edition 1.1

July 2010

D68406

ORACLE

This page intentionally left blank.

This page intentionally left blank.

Table of Contents

About This Workbook

Course Goal

Upon completion of this lab workbook, you should be able to:

- Write servlets using the Java™ programming language (Java servlets)
- Create robust web applications using Struts, session management, filters, and database integration
- Write pages created with the JavaServer Pages™ technology (JSP™ pages)
- Create easy-to-maintain JSP pages using the Expression Language, JSP Standard Tag Library (JSTL), and the Struts Tiles framework
- Create robust web applications that integrate Struts and JSP pages

This workbook presents the lab exercises for each module of the Student Guide.

Conventions

The following conventions are used in this course to represent various training elements and alternative learning resources.

Icons

Note – Indicates additional information that can help you, but is not crucial to your understanding of the concepts being described.

Typographical Conventions

`Courier` is used for the names of commands, files, directories, programming code, and on-screen computer output; for example:

Use `ls -al` to list all files.
`system% You have mail.`

`Courier` is also used to indicate programming constructs, such as class names, methods, and keywords; for example:

The `getServletInfo` method is used to get author information.
The `java.awt.Dialog` class contains `Dialog` constructor.

`Courier bold` is used for characters and numbers that you type; for example:

To list the files in this directory, type:
`# ls`

`Courier bold` is also used for each line of programming code that is referenced in a textual description; for example:

```
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
```

Notice the `javax.servlet` interface is imported to allow access to its life cycle methods (Line 2).

Courier italics is used for variables and command-line placeholders that are replaced with a real name or value; for example:

To delete a file, use the `rm` *`filename`* command.

Courier italic bold is used to represent variables whose values are to be entered by the student as part of an activity; for example:

Type **`chmod a+rwx`** ***`filename`*** to grant read, write, and execute rights for filename to world, group, and users.

Palatino italics is used for book titles, new words or terms, or words that you want to emphasize; for example:

Read Chapter 6 in the *User's Guide*.
These are called *class* options.

Additional Conventions

Java programming language examples use the following additional conventions:

- Method names are not followed with parentheses unless a formal or actual parameter list is shown; for example:

  "The `doIt` method..." refers to any method called `doIt`.

  "The `doIt()` method..." refers to a method called `doIt` that takes no arguments.

- Line breaks occur only where there are separations (commas), conjunctions (operators), or white space in the code. Broken code is indented four spaces under the starting code.

- If a command used in the Solaris™ Operating System (Solaris OS) is different from a command used in the Microsoft Windows platform, both commands are shown; for example:

  If working in the Solaris OS:

  ```
  $cd SERVER_ROOT/BIN
  ```

  If working in Microsoft Windows:

  ```
  C:\>CD SERVER_ROOT\BIN
  ```

Lab 1

Introduction to Java Servlets

Objectives

Upon completion of this lab, you should be able to:

- Explore the NetBeans™ IDE (Integrated Development Environment)
- Create and run a simple servlet

Exercise 1

In this exercise, you will create a simple servlet. In the process, you will begin to become familar with NetBeans, the IDE used throughout this class.

Preparation

No preparation is needed for this exercise.

Task – Create a New Project.

1. Start NetBeans by launching the icon on your desktop. The icon looks like a blue cube.
2. Select the New Project icon (or select File->new project)
3. Select a Java Web project, then click the button labeled Next >
4. Give the project the name `SL314m01lab1`, then click Next > again
5. Click Next > again, then click Finish

Task – Create a servlet

1. High on the left side of the NetBeans UI, notice the project explorer tree. This should show your new project, with Web Pages as a folder underneat it. Underneath Web Pages you will see WEB-INF and `index.jsp`. Right click on the `index.jsp` and select Delete. Answer the prompt to delete the file. Notice that the file disappears from the editor pane on the right.
2. Below the Web Pages entry, you will see a folder labeled Source Packages. Right click on that, and select new and then Servlet.
3. At the Class Name: prompt, enter the name `Lab1Servlet`
4. At the Package: prompt, enter the package name `SL314.m1`
5. Click Next >, notice that the Servlet Name is Lab1Servlet and the URL Pattern is /Lab1Servlet.
6. Click Finish.

Task – Examine the Servlet

1. On the right side of the NetBeans UI, you will now see the template servlet code created for you by the wizard. Notice that the editor is a "folding" editor. That is, it hides chunks of code that might be considered less important, and allows you the opportunity to do the same. The idea is that you can make code more readable by removing clutter from your editor. Notice that very near the bottom of the source file, there is a line that starts with a plus-sign in a box, and this text: HttpServlet methods. Click on the + sign on the left to edit the code.
2. Click the plus sign, and observe that the formerly hidden text is now visible. Notice that there are two methods, `doGet` and `doPost`, which both delegate directly to the `processRequest` method. (This was described in the module.)
3. Notice that both `doGet` and `doPost` are labeled `@Override`. Recall that this is a Java 5 language feature that will cause the compiler to object if the designated method is an overload, or unique method, rather than a genuine override. This gives confidence that `doGet` and `doPost` are actually the correctly formed methods.

Task – Create the New Servlet Code

1. In the `processRequest` method, uncomment the skeleton code. This code creates a minimal HTML page, and will save you some typing.
2. Add a little of your own code to the servlet to personalize it. This could be as simple as adding your name, or something a little more complex, such as a calculation. Remeber, however, that the point of this exercise is familiarity with the tools, and the basic understanding of where code is placed in the servlet, so do not risk wasting time by being unduly ambitious. There's time in the course for that later.
3. After you have entered your code, if is not properly indented, right-click in the editor pane and select Format. Notice that the indentation is corrected to conform with the rest of the code.

Task – Run the New Servlet

1. Type Control-S (or select File->Save) to save the file.

2. Right click on the project and select Run from the popup menu. NetBeans will start the GlassFish™ server and Derby database, compile your servlet, and build the web application. It then attempts to run the application.
3. Because this is a web appplication, to run it requires that a browser be started. In this case, NetBeans automatically starts the browser. However, it points the browser at the base of the application, not at the servlet you just created, so you will get an error from the browser, and must modify the URL to find your servlet successfully.
4. Look at the top of the servlet class. You will see a line like this:

```
@WebServlet(name = "Lab1Servlet", urlPatterns =
{"/Lab1Servlet"})
```

5. Append the urlPatterns element (`/Lab1Servlet` in this example) to the URL in the browser, and set the browser to loading that composite URL.
6. The browser should now display the page created by your servlet, and you should recognize the additional behavior you placed into the template.

Task – Provide an Index Page

1. Find the entry Web Pages in the project browser. Be sure you're looking at the entry for this project. Right click on that and select New -> HTML ....
2. At the prompt HTML File Name: enter **`index`** and click Finish
3. In the resulting template, enter an HTML link (anchor) element that will jump to the URL currently in your browser's URL bar. If you have done everything as specified in these notes, that should be something like:

```
http://localhost:11331/SL314m1lab1/Lab1Servlet
```

Note – The port number might vary depending on the version of NetBeans, GlassFish, or installation conditions.

4. The full text of the link HTML might look like this:

```
Click <a
href="http://localhost:8080/SL314m1lab1/Lab1Servlet">
          here</a> to go to the servlet
```

5. Save the file, and run the project again.

6. This time, you should find that the browser immediately displays your index page, and when you click the link, you are taken to your servlet.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 2

Introduction to Java Server Pages

Objectives

Upon completion of this lab, you should be able to:

- Create a simple Java Server Page (JSP) page
- Send input to a JSP and process that input

Exercise 1

In this exercise, you will create an HTML form and a simple JSP that handles input from that form.

Preparation

No preparation is needed for this exercise.

Task – Create an HTMLForm

Note – From this point on, you will not be given detailed instructions in the use of NetBeans unless you have to do something that has not been previously described. If you have difficulty with the IDE, first consult the lab instructions for lab 1 to see if they contain a description of what you are trying to do. If you still cannot make the IDE do what you want it to, ask your instructor for assistance.

1. Create a new Java Web project named `SL314m02lab1`.
2. Delete the `index.jsp` file.
3. Create a new HTML file called `index.html`
4. Where the template states TODO write content enter the following HTML code:

```
<form action="simple.jsp">
   Please enter your favorite animal: 
   <input type="text" name="favoriteAnimal" value="Gnu"
/>
   <input type="submit" />
</form>
```

5. Save the file

Task – Create a JSP

1. Right click on Web Pages for this project, then select New->JSP... to create a new JSP file called simple (the file will be named `simple.jsp` automatically.
2. In the JSP, change the title to **`Animal Lover's Page`**

3. In the JSP, change the level-one heading to **`Hello Animal Lover`**
4. In the JSP, below the level-one heading, enter the following text:

```
<p>Your favorite animal is a
<%= request.getParameter("favoriteAnimal") %>
Wow, mine is too!
</p>
```

5. Save the file.

Task – Run the Application

1. Run the application, you should see a form that prompts you to enter your favorite animal
2. Type the name of an animal into the text box and click the submit button
3. You should see the output of your JSP, which should state your favorite animal, and express surprise that the computer's favorite is the same.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 3

Implementing an MVC Design

Objectives

Upon completion of this lab, you should be able to:

- Create a model component adhering to JavaBeans™ naming conventions
- Use a servlet to create a controller component, use that controller to invoke a model, and forward that model to a JSP view for display
- User a JSP to create a view component
- Use the Expression Language to read attributes from a JavaBeans compliant model

Exercise 1

In this exercise, you will create an MVC-based web application. The application will allow the user to determine the material traditionally associated with a given wedding anniversary.

Preparation

No preparation is needed for this exercise.

Task – Design a JavaBeans Compliant Model.

1. Consider the attributes needed to model the wedding anniversary problem. One writable attribute will be needed to represent the year number of the anniversary. Another, read-only, attribute will be needed to allow the view to present the name of the associated material.
2. Decide what you will call these attributes, and write them in the first colum of Table 3-1 below.
3. Decide what the necessary corresponding methods will be to support those attributes in the JavaBeans naming conventions. Write those methods in Table 3-1 below. Note that because one attribute is read-only, you will require a total of three methods, so one table cell will remain blank.

Table 3-1 JavaBeans Attribute and Method Names

Attribute Name	AccessorMethod	Mutator Method

4. Decide what key name you will use to store the model in the request when it is forwarded. Write that down here:

Table 3-2 Model Name When Stored in Request Scope

Model name in request scope	

5. Decide on the name of your model class, package name, and what, if any, base class it should extend. Write those down here:

Table 3-3 Bean Class and Ancestry

Bean class name	
Bean package	
Bean class extends	

Task – Implement the Model

1. In NetBeans, create a new web application project called `SL314m03lab1`. This will also be the context root for the application.
2. Create a new class for the model as you specified in Table 3-3 above and provide get and set methods as you specified in Table 3-1 above.
3. In the file `anniversaries.txt`, you will find lists of anniversary material names with the year to which they apply. These data are provided in a form intended to simplify incorporating into a Java class. Use this form if it suits you or simply extract the raw data if you prefer to use that in creating your model. The file can be found under the Files tab (just to the right of the Projects tab in NetBeans) at the root directory of the solution for this lab.
4. (Optional) Create a trivial main method in your model to test the basic functioning of the methods.

Task – Create an HTML Form to Submit the Request

1. You will need a regular HTML form to allow the user of your application to enter the anniversary year they are interested in. When you created the project, NetBeans created an `index.jsp` file. Delete this file and create a new HTML page called `index.html` instead.
2. In the `index.html` file, you will create a form that prompts for a number of years. Decide what you will call the parameter when it is submitted to the web application. Write this parameter name in the Table 3-4 below.

3. The form will invoke an action, which will be the controller servlet that you will create shortly. Decide what the context root of this application will be, and the URL that will invoke the controller. Write these too in the Table 3-4 below.

Table 3-4 Information Required for Assembling the Form

Parameter name	
Context root	
Servlet URL	

4. Create the form in the index.html file.

Task – Create a View Component Using a JSP

1. Decide what you will call your view JSP. Write this down here:

Table 3-5 JSP View Name

JSP name	

2. Create the JSP for the view. It must present the year and anniversary material name returned by the model. Recall the name of the model as in Table 3-2 above. This will be needed along with the attribute names from Table 3-1 above to construct the EL expressions.

Task – Create the Controller

1. The final step in constructing this application is to create a controller to tie it all together. Create an HttpServlet, and ensure that it responds to the URL that you selected in Table 3-4 above.
2. Modify the processRequest method so that it:
   a. Extracts the year parameter named in Table 3-4 above.
   b. Creates an instance of the model.
   c. Sets the year value in the model (consider the data type—is this a String or an int? If you convert, do not try to help the user but simply swallow any exceptions and produce a clean output. Error handling is a topic of later modules.)

d. Stores the model in the request scope using the name in Table 3-2 above.

e. Forwards to the view, named in Table 3-5 above.

Task – Run the Application

1. Right-click the project and run it.
2. Your browser should be launched automatically, and should show the input form. Enter a number of years, and press the submit button.
3. You should see appropriate output in the subsequent browser page.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 4

The Servlet's Environment

Objectives

Upon completion of this lab, you should be able to:

- Extract and examine HTTP request headers
- Create an HTML form
- Create and use an HttpSession to create a simple shopping cart

Exercise 1

In this exercise, you will list the HTTP headers that are sent by your browser.

Preparation

No preparation is needed for this exercise.

Task – Create `index.jsp` and a View JSP

In this project, the MVC model will be simplified a little. The intention is that you will create a servlet to act as controller and a JSP to act as view, but will use a String object to carry the result data directly from the servlet to the JSP.

1. Create a new project called `SL314m4l1`.
2. Edit the automatically created `index.jsp` file so that it carries a single link that will lead to a relative URL, `/ListHeaders`. You may use an HTML anchor to do this, or if you're feeling ambitious, a submit button on an otherwise empty form.
3. Add a new JSP to the project, call this file `HeadersView.jsp`
4. Add to the view introductory text such as **`These are the headers of your browser`**.
5. Add an EL expression that will output the value of a text element in the request scope called `headerList`.

Task – Create the Controller Servlet

1. Add a servlet to the project. Make the fully qualified name of the servlet **`sl314.m4.HeaderServlet`**.
2. Set the URL pattern for the servlet to **`ListHeaders`**.
3. In the servlet, declare a `StringBuilder` called `result`. Set the value of the result to **`<UL>\n`**.
4. Obtain the enumeration of header names from the request object.
5. For each entry in the enumeration:
   a. Append **`<LI>`** to result.

b. Append the header name to `result`.

c. Append **`=`** to `result`.

d. Extract the value of the header from the request and append that value to `result`.

e. Append **`</LI>\n`** to the `result`.

6. Append **`</UL>\n`** to `result`.
7. Add an attribute to the request called `headerList` and set the value to `result`.
8. Forward to `HeadersView.jsp`.
9. Finally, run the example and verify that you see headers such as host, user-agent, accept and others.

Note – As you create the HTML in your servlet, consider the implications this has for maintainability. Previous discussions have already identified this as a bad practice. Can you determine why this is being done in this example? How do you think this could be avoided?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 5

Container Facilities For Servlets & JSPs

Objective

Upon completion of this lab, you should be able to:

- Identify critical elements of the deployment descriptor, `web.xml`

Exercise 1

In this exercise, you will investigate the deployment descriptor and the container's servlet support.

Preparation

No preparation is needed for this exercise.

Task – Investigate Projects Using `web.xml`

1. Create a new Java Web project named `SL314m05lab1`
2. Create a servlet. Accept the default name `NewServlet`, and place it in the package `sl314m5.lab`. Click Next >
3. Select the checkbox Add information to deployment descriptor (`web.xml`)
4. Modify the URL Pattern(s) field so that it reads `/PathToServlet/*`
5. Click on the New button to the right of the table labeled Initialization Parameters. Type into the newly created row of the table the values **`KeyName`** and **`Data Value`** under the headings Name and Value respectively.
6. Click Finish
7. Notice that under the project folders Web Pages->WEB-INF there is a new node named `web.xml`. Double click on that element.
8. From the editor pane, select the General tab, then open the Context Parameters region. (Click on the plus sign).
9. Select Add... in the Context Parameters region, enter a parameter name and a value.
10. Select XML at the top of the editor window, toward the right hand end. You should see an XML file somewhat like this:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <web-app [...]>
3       <context-param>
4           <param-name>MyParameter</param-name>
5           <param-value>My Parameter Value</param-value>
6       </context-param>
7       <servlet>
8           <servlet-name>NewServlet</servlet-name>
```

```
9           <servlet-class>sl314m5.lab.NewServlet</servlet-class>
10          <init-param>
11              <param-name>KeyName</param-name>
12              <param-value>Data value</param-value>
13          </init-param>
14      </servlet>
15      <servlet-mapping>
16          <servlet-name>NewServlet</servlet-name>
17          <url-pattern>/PathToServlet/*</url-pattern>
18      </servlet-mapping>
19      <session-config>
20          <session-timeout>
21              30
22          </session-timeout>
23      </session-config>
24  </web-app>
```

11. Examine the XML file and determine how a triggering URL is matched to the servlet class that must run in response to that URL.
12. Examine the XML file and determine how the key-value pairs you specified are represented. Note also how one couple is associated specifically with the servlet, while the other is at almost the top level of the XML structure and is outside the servlet definition.

Task – Code the Servlet to Use the Configuration Information

1. Edit the servlet source file, writing your code in the `processRequest()` method that has been added by NetBeans.
2. Extract a reference to the `ServletContext` object.

Note – If you want to know what methods are available directly on the `HttpServlet` object, you can check the documentation, or simply type **`this.`** in the editor and wait for NetBeans to prompt you with a list. You should find the method needed to get the servlet init parameter very easily.

3. Store the servlet context in a variable.

Note – To declare the variable of type `ServletContext`, you will need to add an import. NetBeans will prompt you to do this using a little lightbulb and red-ball icons in the left margin of the browser. Click that icon and select the entry Add import for `javax.servlet.ServletContext`.

4. Using the servlet context reference, locate a method that you think might provide access to a logging facility. Using this method, write a message to the web-container's logging stream that indicates that the servlet has been invoked and outputs the time of that invocation.

Note – You can use `"message " + new java.util.Date()` to create a message that includes the current date and time.

Note – You will also find `log` methods defined on the servlet itself, through the this reference.

5. Using the same servlet context reference, determine how you can obtain the context parameter that you stored in the `web.xml` file. Recall from the lecture that this is referred to in the APIs as an `initParameter`. As before NetBeans typing completion can be used to help.
6. Output the key and value pair in a meaningful message to the log stream.
7. Extract the value of the servlet initialization parameter. Notice that this value is extracted directly from the `HttpServlet` object (`this`).
8. Write the name and value of the parameter to the log stream.
9. Create a minimal HTML page so that the servlet will generate some visible output on the browser. Consider including the current date and time here too.

Note – If no output were created, the test of logging and parameter handling would still work, however, it would be rather disturbing to have zero output on the browser. This further demonstrates the usefulness of the logging facility, as it allows you to see progress in a servlet or JSP even when the application is malfunctioning

Task– Prepare the Index Page and Run the Application

1. Edit the `index.jsp` file of the project. Create a link in it. That link should jump to `PathToServlet/blah`.
2. Run the program.
3. Look at the GlassFish v3 Domain tabs. One shows the building and deployment of the application. The other shows the output from the logging requests.
4. Determine that the key/value pairs in the logging output are correct.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 6

More View Facilities

Objectives

Upon completion of this lab, you should be able to:

- Understand and use the four scopes
- Understand and use the expression language, EL
- Recognize and use the EL implicit objects

Exercise 1

In this exercise, you will investigate EL implicit objects and the JSTL `forEach` tag.

Preparation

No preparation is needed for this exercise.

Task – List HTTP Headers

1. Create a new Java Web project named `SL314m06lab1`.
2. In the resulting `index.jsp` page, change the title and heading to **`HTTP Headers`**
3. Create a taglib directive to make the core JSTL tag library available with a prefix of `c`. Consult "Tag Example" in Module 6, "More View Facilities," in the student guide if you need a reminder of the syntax for this.
4. Prepare an unnumbered list in the body of the document.
5. Inside the unnumbered list tags, place a `c:forEach` tag that will enumerate the elements of the HTTP headers array. Consult "EL Implicit Objects" in Module 6, "More View Facilities," in the student guide if you need help in finding the header array.
6. Test the program (note that there is no servlet in this exercise).

Exercise 2

In this exercise, you will investigate using EL to access complex data structures.

Preparation

No preparation is needed for this exercise.

Task

1. Open the project named `SL314m06lab2` and look at the classes `domain.Customer` and `domain.Address`
2. Examine the launch page `index.jsp`, and the servet class `web.Controller`, to determine the basic structure of the application. Note that the application is missing the view `CustomerView.jsp` to which the controller servlet forwards.
3. Create the view class `CustomerInfo.jsp`. Arrange that it displays the name and three addresses of the customer
4. Test your application for the customer with the ID 1 (only one exists).
5. There are two distinct ways (syntactically) that you can access the three customer addresses—using arrays or using the three distinct fields. Whichever you used in step 3, modify your view to use the other approach, and retest your application.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 7

Developing JSP Pages

Objectives

Upon completion of this lab, you should be able to:

- Describe the translation of a JSP to a Servlet
- Understand the use of JavaBeans with `jsp:useBean`, `setProperty`, and `getProperty` tags

Exercise 1

In this exercise, you will investigate the compilation of JSPs to servlets.

Preparation

No preparation is needed for this exercise.

Task – Investigate JSP Translation.

1. Create a new Java Web project, called `SL314m07lab1`.
2. Add a little body text to the JSP.
3. Run the project.
4. Right click the JSP and select View Servlet from the pop-up menu.
5. Examine the contents of the servlet, and notice how it corresponds to your original JSP.
6. Make the following changes to your JSP, and with each change, re-run the project, then examine the servlet source again.
   a. Add an initialize declaration for a `private int` variable.
   b. Add a page directive to import `java.util.*`.
   c. Add an expression to output the result of doubling the `private int` variable you added in step a above.
   d. Add code to iterate over the numbers 1 to 10, printing a message as part of an unnumbered list each time round. Be sure to use explicit curly braces for the loop boundaries.
   e. Edit the code from step d. above by removing the curly braces. What happens now? Examine the generated servlet and determine what went wrong.

Note – In step e above, NetBeans will complain of an error in the servlet itself. Ignore this, and attempt to run the file regardless.

Exercise 2

In this exercise, you will use the `jsp:useBean`, `setProperty` and `getProperty` tags.

Preparation

No preparation is needed for this exercise.

Task – Investigate the `jsp:useBean` Tag

1. Open the project called `SL314m07lab2`.
2. Examine the `index.jsp` page. This contains a form that triggers the `setAddress.jsp` page and submits five parameters to that page.
3. Examine the class `domain.Address`. This is a simple JavaBean that holds address information. Notice that the field names declared in the form and the attribute names of the Java Bean are identical.
4. Create a new JSP, called `setAddress.jsp`.
5. Add code to the `setAddress.jsp` to use the `<jsp:useBean` tag to create a JavaBean.
6. In the setAddress.jsp add code using the `<jsp:setProperty` tag to populate all the fields of the addressBean directly from the input properties. If you are in doubt, refer to the section under the heading "The setProperty Tag" on page 7-27 of the student guide.
7. Add an unnumbered list, and in it, print out each element of the bean using the `<jsp:getProperty` tag.
8. Create a second unnumbered list, and in it, print out each element of the bean using scriptlet code. Recall from "The useBean Tag" on page 7-26 of the student guide that the `<jsp:useBean` tag creates a variable in the generated servlet code. That variable has the name provided in the `id` element of the `<jsp:useBean` tag.
9. Test your application and determine that the data are transferred from the form into the bean and are then displayed by the `setAddress.jsp` page.

Task – Retrieve Session Information Using JSP Tags

1. Create a new page called `showAddress.jsp`.
2. Using elements of the `showAddress.jsp` page for assistance, create a standalone page that shows the current values of a bean called `addressBean` that is in the session scope. Note that this page will be loaded by typing its URL directly. The goal is to demonstrate that the bean created in the previous task is still accessible because it is in the session scope.
3. Test the page to show the values from the previous form submission.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 8

Developing JSP Pages Using Custom Tags

Objective

Upon completion of this lab, you should be able to:

- Use custom tags in creating a JSP

Exercise 1

In this exercise, you will investigate the use of core JSTL tags.

Preparation

No preparation is needed for this exercise.

Task – Investigate the `c:out` Core Tag.

1. Create a new Java Web project, called `SL314m08lab1`
2. Into the `index.jsp` file, add a `taglib` directive for the core JSTL library. Refer to "JSTL Functional Areas" on page 8-13 in the student guide for details of the prefix and URI that are expected for this.
3. Create a paragraph in your JSP that contains the following literal text:
```
<script type='text/javascript'>
alert('How annoying');
</script>
<h1 this is not a heading
X > Y... is that true & is the other true too?
```
4. Ignore the errors for now and run the application. Notice how the text is interpreted. Some of it disappears, and some behaves strangely.
5. Surround each of these lines with a **`<c:out`** tag. Consult "JSTL out Tag" on page 8-11 of the student guide for a reminder of the syntax.
6. Run the application again and observe the behavior. Notice that the default value of `escapeXml` is `true`, and this has created output characters that present like the input text, even though they are not achieved directly. You will see in Module 12, "Implementing Security" that this can be a valuable tool in protecting against cross-site scripting attacks.

Task – Investigate the `c:remove` Core Tag.

1. Open the project `SL314m08lab2`. Examine the `Address` JavaBean (this is the same one you have seen before).

2. Into the `index.jsp` file, add a `jsp:useBean` directive to create an instance of the Address bean in the session scope.
3. Add a `jsp:setProperty` tag to set a literal value of **`1234 Acacia Gardens`** for the property `address1` o f the bean.
4. Add a `jsp:getProperty` tag to output the value of the `address1` property (to prove that it is present and assigned).
5. Run the application to demonstrate that the bean has been created and the value assigned.
6. Add the `taglib` directive to allow the use of the core tag library.
7. After the value of the property has been printed out, add a `c:remove` tag to remove the bean, then repeat the code you used to output the value further up the file.
8. Run the application and demonstrate that the bean was removed, and therefore the attempt to output the value of the address field fails.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 9

More Controller Facilities

Objectives

Upon completion of this lab, you should be able to:

- Understand fundamental concurrency problems
- Add a filter to a servlet

Exercise 1

In this exercise, you will demonstrate a problem that can arise in a concurrency situation. The project simulates a very simple bank account, modeling only the balance.

Preparation

No preparation is needed for this exercise.

Task – Witness the Failure

Note – Certain versions of Firefox browser will not operate correctly in this task. The browser does not send concurrent requests to the same URL from separate windows or tabs. An alternative browser is required for this exercise.

1. Open the project `SL314m09lab1` and run the project.
2. In your browser, select Deposit and deposit **`100`**, then select Withdraw and withdraw **`50`** from the account. Satisfy yourself that the project is capable of the essential computation for this simulation.

Note – The page will take about five seconds to load after you click Submit. This is not because the simulation is particularly complex, but you will examine the reason in Task 2.

3. Open a second tab in your browser, and load the base page of the application in that too. It should show the same balance that was left after step two.
4. Without clicking Submit yet in either tab, set one tab up to deposit **`100`** and the other to withdraw **`200`**.
5. In less than five seconds, click Submit on both tabs. When the pages return, notice that each shows a balance that would have been correct if the other operation had not occurred, but is wrong for the aggregate of both operations.
6. In either of the tabs, deposit a zero amount. Notice that the balance shown is definitely wrong.

Task – Examine the Cause of the Failure

1. Open the `domain.Account` class in the NetBeans editor window.
2. Observe that in the `deposit` and `withdraw` methods, the computation consists of:
   a. Read the balance
   b. Modify the balance
   c. Store the new balance

   This logic is correct, however, a delay has been deliberately added after step a. this means that if two threads execute these methods concurrently, each will start with the unmodified balance, and then write it back. In effect, the last one to complete wins. This is called a write-write conflict.
3. Consider how you might fix this problem. Unfortunately, the solution of removing the sleep, and of modifying the code to read `this.balance += amount` does not solve the problem. It greatly reduces the chance that you will see it, especially in a lab environment with one user testing the code, but it is not a correct solution. The problem remains that the execution of the += operator still involves reading the value, modifying it, and then storing the result. Although the time delay between the read and the write is small, it is not zero.

   In effect, whenever variable data are shared between threads, this kind of problem might arise. A full solution may often be achieved using synchronization or database transactions, however, details of these are beyond the scope of the class.
4. Modify the methods `deposit` and `withdraw` by marking them `synchronized`. Do not remove or alter any of the existing code.
5. Run steps 3 and 4 from Task 1 above again. Notice this time that the simulation takes longer to complete, but the answer is correct.

Exercise 2

In this exercise, you will add a filter to an existing servlet.

Preparation

No preparation is needed for this exercise.

Task – Examine the Starting Point.

1. Open the project `SL314m09lab2` and run the project.
2. Select different operations and observe the application's behavior.

Task – Install a Login Mechanism in the Application

1. Add code to the `ApplicationController` to attempt to read an attribute `user` from the current session. Set the value found (or `null` if not found) in an attribute `user` in the request scope.
2. Modify the `quote.jsp` and `advertize.jsp` pages to conditionally welcome the user if the `user` attribute is not `null`. Use the core JSTL tag library to provide the conditional behavior for this.
3. Create a new JSP called `login.jsp`. This should collect a username on a form, and submit this along with the parameter `operation` set to login to the `ApplicationController`. You might need to use the following HTML tag to achieve this:
   `<input type="hidden" name="operation" value="Login"/>`
4. Add a JSP called `welcome.jsp`. This should welcome the new user and be the target of the controller's forwarding in the case that the operation is Login.
5. Modify the `ApplicationController` so that if the operation parameter is login it creates an `HttpSession`, and embeds the provided user name in that session.
6. Test that the application now addresses a logged-in user by name. You will need to navigate to the `login.jsp` page manually.

Task – Install a Security Filter in the Application

1. Create a `ServletFilter`, called `SecurityFilter`. Configure it to intercept calls to the `ApplicationController`. As you go through the wizard, select REQUEST for the Dispatch Conditions.
2. Edit the template-generated filter:
   a. Delete the methods `doBeforeProcessing` and `doAfterProcessing`.
   b. Delete everything in the `doFilter` method except for the call to `chain.doFilter`.
   c. Delete the methods `getFilterConfig` and `setFilterConfig`.
   d. Delete the method `toString`.
   e. Delete the method `sendProcessingError`.
   f. Delete the method `log`.
   g. In the `init` method, delete everything except the line that stores the `filterConfig` parameter in the instance variable of the same name.
3. In the `doFilter` method, before the call to `chain.doFilter`, add code to do the following:
   a. Determine what the operation to be performed is (use a `request.getParameter("operation")` call).
   b. If the operation is anything other than Quote, proceed to call `chain.doFilter` and end processing in this `doFilter` method.
4. For Quote operations only, proceed as follows:
   a. Determine if there is a session associated with this request, and if so, determine if the session contains an attribute called `user`.
   b. If the `user` attribute exists and is not empty, processing proceeds with the call to `chain.doFilter`.
   c. If the session does not exist, or if the `user` attribute does not exist or is empty, processing proceeds by forwarding through a `RequestDispatcher` configured to dispatch to the page `login.jsp`.
5. Run the program and verify:
   a. Any user, regardless of whether they are logged in or not, can log in, and can access the advertisement.

b. When an attempt is made to get a Quote, this is successful only if the user has logged in already. Otherwise, the request results in a redirection to the login page.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 10

More Options for the Model

Objective

Upon completion of this lab, you should be able to:

- Create applications that use Java Persistence API for persistent storage

Exercise 1

In this exercise, you will create a simple database application that provides a list of songs.

Preparation

No preparation is needed for this exercise.

Task – Create a New Project and an Entity.

1. Create a new project called `SL314m10lab1`.
2. Create a new Entity:
   a. Called `Song`.
   b. In the package `entity`.
   c. With a Primary Key Type of `Long`.
   d. Create a Persistence Unit for the entity. Use the default name, the default provider, the data source `jdbc/__default`, the transaction API, and a generation strategy of Create.
3. Add fields to the entity for `String artist`, and `String songName`. Provide simple `get` and `set` methods for these. Be sure to annotate the fields using the `@Column` .

Task – Create a controller

1. Create a servlet called `Controller`, in the package `web`.
2. In the controller, extract a parameter from the request called `operation`. This value should be one of `list` or `add`. Based on this operation the controller should either list all the songs in the database, or add a new one.
3. Arrange for the servlet to have use of two injected resources. These will be provided using the annotated variable declarations
   `@PersistenceContext EntityManager em;`
   and
   `@Resource UserTransaction utx;`

4. To implement the list operation, look at the example in "Java Persistence API Example" on page 10-27 of the student guide. Assuming you have an entity manager variable called `em`, you can use this call to create a list of the results:
   `em.createQuery("select s from Song s").getResultList()`
5. To implement the add operation, the controller should expect two additional parameters to the HTTP request, these will be `artist` and `song`. Use these to create and initialize a new `Song` object, then begin a user transaction, persist the new `Song`, and commit the user transaction.
6. Regardless of the operation invoked, the controller should forward to a JSP called `ListSongs.jsp`, and should provide an attribute in the request scope that carries the songs that should be displayed.

Task – Create JSPs

1. Create a JSP called `AddSong.jsp` that includes a form. The form should have two fields: `Artist` and `Song Title`. When submitted, this JSP should trigger the add operation in the controller servlet.
2. Create a JSP called `ListSongs.jsp`. This should show the list of songs created by the controller and be the target of the redirection from the controller after each operation. The servlet should include a button labeled Refresh that triggers the controller's list action.
3. Expect to use the JSTL core tag library to create this JSP, as you will need to iterate over the elements of the List that is passed to the page by the controller.

Task – Test the Application

1. Test the application by adding a few songs and ensuring that the list of songs grows with each new addition.

Task – Optional: Add Edit, Delete, and Find Operations

1. Should you have spare time after finishing this lab, try adding additional functions to complete the database set.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 11

Asynchronous Servlets and Clients

Objectives

Upon completion of this lab, you should be able to:

- Create servlets that use the asynchronous model
- Create web pages that use asynchronous JavaScript updates

Exercise 1

In this exercise, you will create a servlet that generates delayed responses using the asynchronous servlet API and asynchronous JavaScript™ techniques.

Preparation

No preparation is needed for this exercise.

Task – Create a New Project and Asynchronous Servlet

1. Create a new project named `SL314m11lab1`. In the project create a servlet called `Update` in the package `web`.
2. Set the servlet to support asynchronous operation. Do this using the element `asyncSupported = true` in the `@WebServlet` annotation.
3. Delete the contents of the `processRequest` method.
4. In the now-empty processRequest method:
   a. Start the asynchronous processing and keep a handle on the `AsyncContext` in a `final` variable.
   b. Consult the sample code "Simple Asynchronous Client Example" on page 11-10 of the student guide for more guidance on these steps.
5. Create an anonymous inner class, implementing the `Runnable` interface. In the `run` method of this class:
   a. Extract the `request` and `response` objects from the `AsyncContext` and store them in local variables.
   b. Introduce a delay between 2 and 15 seconds.
   c. Select a word from this list at random:
   wonderful, unexpected, strange, elegant
   (you can add more adjectives if you choose, they will simply be used to construct a changing random remark). Return the chosen word as the entire body of the response (do not include any HTML tags whatsoever).
   d. Call the `complete` method on the `AsyncContext`.

6. Test the code. Run the application. To do this, build and deploy the application, and then enter the URL:
`http://localhost:8080/SL314m11lab1/Update` directly into the browser's URL window.

Task – Create an Asynchronous Client

1. Edit the `index.jsp` file and the following text just after the top-level heading (`</h1>`):
`<p>Oh my, that was... <span id="adjective">let me think...</span>!</p>`
2. Create a JavaScript block, after the end of the `</body>` tag. The script opening and closing tags are exemplified in"Simple Asynchronous Client Example" on page 11-10 of the student guide.
3. In the JavaScript block, add the following elements. Refer again to "Simple Asynchronous Client Example" on page 11-10 of the student guide if you need hints or guidance on the JavaScript syntax.
   a. Add variable declarations for two variables called `req` and `toUpdate`.
   b. Use the `document.getElementById` method to obtain a reference to the `adjective` span that was created in step one of this task.
   c. Initialize the `req` variable using either a new `XMLHttpRequest`, or a new `ActiveXObject`, depending on the browser support.
   d. Declare a function called `sendRequest()`.
   e. In the `sendRequest()` function, install a `readystatechange` handler. In the handler, when a successful response is received, update the `innerHTML` element of the `toUpdate` variable using the `responseText` of the request.
   f. After the declaration of the `readystatechange` handler, call the `open` and `send` methods on the `req` object to send the HTTP request to the server. The request should send a `GET` request, to the URL `Update`.
   g. Close the definition of the `sendRequest` method, then call the `sendRequest` method directly.

Task – Test the Application.

1. Run the application.
2. The page should be displayed and show text that says
   `Oh my, that was... let me think...!`
3. After a few seconds, the text `let me think...` should change, and continue changing at variable intervals of a few seconds.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 12

Implementing Security

Objectives

Upon completion of this lab, you should be able to:

- Restrict access to regions of your web application to validated users
- Enforce an SSL-encrypted connection to a GlassFish-hosted website

Exercise 1

In this exercise, you will use role based security to restrict access to elements of a web application.

Preparation

No preparation is needed for this exercise.

Task – Verify the Project Prior to Adding Security

1. Open the project `SL314m12lab1` and run it. This is essentially the same as Lab 9, but without the filter. Verify that you can get to all the pages, and can log in or not as you choose.

Task – Configure Users in the GlassFish Server

1. Select the Services tab (second tab to the right of the Projects tab, at the top left of the NetBeans window).
2. Open the Servers tree element and right click the GlassFish Domain. Select View Admin Console from the popup menu.
3. When the admin console starts in your web browser, open the Tree on the left side of the window. Find the hierarchy: Common Tasks > Configuration > Security > Realms > file, and click the file element.
4. The right hand pane shows the page Edit Realm. Click the button Manage Users just below this title.
5. Click the New button to add a user. Add the User IDs **`Alice`** and **`Maverick`** in each case, use the User ID (with the same capitalization) as the password.

Task – Assign Users to Roles

1. In the Projects tab, open the tree SL314m12lab1 > Web Pages > Web Inf. Right-click the `sun-web.xml` file and select edit.
2. In the editor pane at the right, select the Security tab and click the button at the right Add Security Role Mapping.

3. Edit the field Security Role Name which currently contains the text role1 and change the role name to **`Users`**.
4. Click the button Add Principal and enter the user name **`Alice`** then click OK.
5. Type Control-S to save the file.

Task – MandateLogin Before Access

1. Right click on the project, and select New > Standard Deployment Descriptor (`web.xml`).

Note – If this element isn't visible on the menu, select other from the bottom of the list, then select Web and the Standard Deployment Descriptor element will be the last entry in the list on the right hand side.

2. Select the Security tab, then open the Login Configuration element and select the radio button for Basic.
3. Open the Security Roles element, click the Add button, and enter Users, then click OK.
4. Open the Security Constraints element (click the Add Security Constraint button). Enter a resource name (**`TheConstraint`** is fine).
5. Click the Add button located just under the Name column heading in the Web Resource Collection block. In the dialog box that pops up, enter a resource name (**`MyResource`** will serve), then enter the URL Pattern **`/ApplicationController`**. Ensure that All HTTP methods is selected and then click OK.
6. Check the checkbox Enable Authentication Constraint, then click the Edit button to the right of Role Name(s). Select the role Users and click the Add > button. Click OK.

Task – Run the Application and Verify Login Requirement

1. Run the program, and verify that before you can reach either the Quote or Advertisement pages, you are forced to log in as Alice. You should be rejected from the pages if you login as Maverick, since this user was not added to the Users role.

Task – Pick Up the Username from the Environment

1. Edit the Application Controller servlet so that it does not look for a username in the parameters, and does not provide a login operation
2. In place of the old mechanism you just removed, extract the user name from the `Principal` object that can be found from the `request`.

Task – Modify the Application to Use Form-based login

1. Edit the `login.jsp` form that will be used to prompt for user login. This should submit to the action `j_security_check`. The user name field must be called `j_username`. Create a password field, which must be called `j_password`.
2. Arrange two buttons on the form. One should be of type submit and labeled Login. The other should be of type reset and labeled Clear.
3. Edit the `web.xml` file. Change the Login Configuration from Basic to Form. In both the fields for Form Login Page and Form Error Page, enter **`login.jsp`**
4. Test that the application now uses the form you created for login, and that failed logins are also directed to the same login page.

Task – Provide a Logout Mechanism

1. To this point, there has been no proper way to logout once you are logged in. To provide for this, add a new operation Logout to the `ApplicationController` servlet. When invoked, this should invoke the method `session.invalidate()`, and dispatch back to the `index.jsp` page.
2. Add another option Logout to the `index.jsp` page.
3. Test that the application now allows a user-controlled logout operation.

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications