EXERCISE 1: Creating a Simple JSP Component

In this exercise, you create, deploy, and run a web application project with a simple JSP component.

This exercise contains the following sections:

- "Task 1 – Developing a Simple JSP Page"
- "Task 2 – Deploying and Testing the Sample Web Application"

Preparation

This exercise assumes that the application server is installed and configured in Netbeans.

Task 1 – Developing a Simple JSP Page

Complete the following steps from NetBeans:

1. From the menu select File then New Project.
2. Under Categories, click Web.
3. Under Project, click Web Application.
4. Click the Next button.
5. The new Web Application project should have the following characteristics:
    - Project Name: `SampleWebApplication`
    - Project Location: `/export/home/student/projects`
    - Set as Main Project: (checked)
6. Click the Next button.
7. The Server and Settings dialog should have the following information:
    - Server: `GlassFish v2`
    - J2EE Version: `Java EE 5`
    - Context Path: `/SampleWebApplication`
8. Click the Finish button.

9. An index.jsp file is created for you automatically. You can place any static HTML in a JSP page. Experiment with adding Java code to the JSP page.

The following is an example of what you might enter:

```
<%= new java.util.Date() %>
```

Task 2 – Deploying and Testing the Sample Web Application

Complete the following steps:

1. Deploy the `SampleWebApplication` Web project by right-clicking on the project folder in NetBeans and selecting Undeploy and Deploy.

2. Test the application by right-clicking on the project folder and selecting Run, or by pointing a web browser at:

```
http://localhost:8080/SampleWebApplication/index.jsp
```

EXERCISE 2: Troubleshooting a Web Application

This exercise contains the following sections:

- "Task 1 – Creating a Faulty Web Component"
- "Task 2 – Deploying and Testing the Faulty Web Application"
- "Task 3 – Viewing Application Server Error Messages"

In this exercise, you introduce an error into the `SampleWebApplication` project. Then, redeploy the application and review the errors generated by your change to the code.

Preparation

This exercise assumes that the previous exercise has been completed.

Task 1 – Creating a Faulty Web Component

Complete these steps:

1. Modify the `index.jsp` web component to produce an exception upon execution.

Enter the following into your index.jsp page:

```
<%
    Object o = null;
    o.toString();
%>
```

Task 2 – Deploying and Testing the Faulty Web Application

Complete the following steps:

1. Deploy the `SampleWebApplication` Web project.

2. Test the application by selecting Run or pointing a web browser at:

`http://localhost:8080/SampleWebApplication/index.jsp`

Notice the error message displayed in your web browser.

Task 3 – Viewing Application Server Error Messages

Tool Reference – Server Resources: Java EE Application Servers: Examining Server Log Files

Complete the following steps:

1. Bring up the Application Server output in the IDE.

2. Find where the web application causes an exception to be generated. The line should be similar to:

```
Servlet.service() for servlet jsp threw exception
java.lang.NullPointerException
```

3. Remove the error placed in the index.jsp file during Task 1 and redeploy the application.

EXERCISE 3: Creating a Simple Servlet Component

This exercise contains the following sections:

- "Task 1 – Coding the Servlet"
- "Task 2 – Deploying and Testing the Application"

In this exercise you create a simple servlet.

PREPARATION

This exercise assumes that the application server is installed and running.

Task 1 – Coding the Servlet

Complete the following steps:

1. Right-click the `SampleWebApplication` project select New then Servlet.

2. Enter for the following information for the servlet:

   - Class Name: `SimpleServlet`
   - Project: `SampleWebApplication`
   - Location: Source Packages
   - Package: `test`

3. Click Next.

4. Enter the deployment descriptor information for the servlet:

   - Class Name: `test.SimpleServlet`
   - Servlet Name: `SimpleServlet`
   - URL Pattern: `/SimpleServlet`

5. Click Finish. This creates the deployment descriptor information in the web.xml file. To see the contents of this file, open the Configure Files folder for your project. Then, double-click on the web.xml file.

6. Modify the `processRequest` method of the `SimpleServlet` servlet to display a dynamically generated message. First, remove the comments surrounding the `out.println()` statements. Modify your servlet using the following code as a guide:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
   response.setContentType("text/html;charset=UTF-8");
   PrintWriter out = response.getWriter();

   out.println("<html>");
   out.println("<head>");
   out.println("<title>Servlet SimpleServlet</title>");
   out.println("</head>");
   out.println("<body>");
   out.println("<h1>Servlet SimpleServlet at "+
                                    request.getContextPath()+"</h1>");
   out.println("Generated at: " + new java.util.Date());
   out.println("</body>");
   out.println("</html>");

   out.close();
}
```

7. Show the HTTP methods hidden below the `processRequest` method. Verify that both the `doGet` and doPost methods call the processRequest method.

Task 2 – Deploying and Testing the Application

Tool Reference – Java EE Development: Web Modules: Web Deployment Descriptors

Complete the following steps:

1. Open the `web.xml` deployment descriptor in the `SampleWebApplication` project.

2. Ensure that the servlet element inside the web-app element correctly specifies the Controller servlet. It should look like:

```
<servlet>
<servlet-name>SimpleServlet</servlet-name>
<servlet-class>test.SimpleServlet</servlet-class>
</servlet>
```

The servlet-class element specifies the fully-qualified name of the Java class implementing the servlet. The servlet-name element specifies the name by which the servlet is identified elsewhere in the descriptor.

Although they are the same in this example, this is not a requirement and frequently the values are different. It is possible for more than one servlet to be implemented by the same Java class.

3. Ensure that the servlet-mapping element located after the servlet element maps the `SimpleServlet` servlet to the `/SimpleServlet` URL pattern. It should look like:

```
<servlet-mapping>
<servlet-name>SimpleServlet</servlet-name>
<url-pattern>/SimpleServlet</url-pattern>
</servlet-mapping>
```

This element indicates that a URL of `SimpleServlet`, relative to the context root (/), maps to the `SimpleServlet` servlet. Thus, you can access the servlet using the

`http://localhost:8080/SampleWebApplication/SimpleServlet` URL.

4. Build the `SampleWebApplication` project. Correct any errors.

5. Deploy the `SampleWebApplication` Web project.

6. Test the application by selecting Run or by pointing a web browser at:

`http://localhost:8080/SampleWebApplication/SimpleServlet`

You should see a dynamically generated web page. This indicates that the servlet has been successfully invoked.