COMP S380F Web Applications: Design and Development
Lab 4: JSP – Using JavaBean, Init Parameters & Configuring JSP properties in web.xml

In this lab, we will use JSP to redo the exercises of Lab 3. The following topics are covered:

- Using a JavaBean in JSP
- Setting Servlet Init Parameters in JSP
- Handling HTTP GET and POST requests with different JSP pages
- Configuring JSP properties in web.xml

Task 1: Cloning a Project from GitHub, Using a JavaBean in JSP

We will work on an existing web application, which is downloaded from a public repository in GitHub. The web application will have a visit counter counting the number of visits on the JSP pages.

1. In IntelliJ, clone the Lab 3 answer's Gradle Web Application project by clicking "Get from VCS" with the following properties:
   - Version control: **Git**
   - URL: **https://github.com/cskeith/380_2024.git**

   After opening the cloned project, click on the menu bar: Git > Branches… . Select the remote branch "origin/lab03ans" > Checkout to check out the Lab 3 answer project. Create a new branch "lab04ans" by clicking on menu bar: Git > New Branch… and checkout.
2. Update the project name and module name to "Lab04" by clicking on menu bar: File > Project Structure, and set up the new name in "**Project**", "**Modules**" (right-click on Lab03 and select "**Change Module Names…**") under Project Settings.
3. In the file **settings.gradle**, update `rootProject.name` to "Lab04". Then, set up Tomcat.
4. A JavaBean is a specially constructed Java class coded according to the JavaBeans API specifications. JavaBeans are usually used to represent a Data Transfer Object (DTO).
   - It provides **a default, no-argument constructor**.
   - It should be **serializable** and implement the Serializable interface.
   - It may have a number of **properties** (i.e., variables) which can be read or written.
   - All variables have **accessor** (get) and **mutator** (set) **methods**.

   Create the JavaBean class below, which will be instantiated for keeping the visitor count.
   - Package name: hkmu.comps380f
   - Class name: `VisitCounter`
   - One property: `int count` (equal to 0 initially)
5. Modify the JSP file `index.jsp` in the application root, as follows.

```
<%@page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Index page</title>
</head>
<body>
<jsp:useBean id="visit" scope="request"
             class="hkmu.comps380f.VisitCounter"/>
<jsp:setProperty name="visit" property="count"
                 value='<%= visit.getCount() + 1%>'/>
<h1>Example</h1>
The site was visited for
<jsp:getProperty name="visit" property="count"/>
times.
</body>
</html>
```

- The JSP action `<jsp:useBean id="beanName" class="package.Class" />` creates a new instance of the JavaBean or uses an existing one if it already exists. We can store the JavaBean object in one of the following four **scopes** using the `scope` attribute.
  - ➢ `page`: The JavaBean is not shared with other web components by default.
  - ➢ `request`: The JavaBean is stored in the `HttpServletRequest` object.
  - ➢ `session`: The JavaBean is stored in the `HttpSession` object.
  - ➢ `application`: The JavaBean is stored in the `ServletContext` object.
- The JSP action `<jsp:getProperty name="beanName" property="propertyName" />` reads and outputs the value of a bean property (by calling its getter method).
- The JSP action `<jsp:setProperty name="beanName" property="propertyName" value="propertyValue" />` modifies a bean property (by calling its setter method).

6. Run the project. Does the visitor counter in the index.jsp page increase?

7. Revise index.jsp such that the JavaBean is stored to the ServletContext object. Run the project again. Does the visitor counter in the index.jsp page increase?

Task 2: Using Servlet Init Parameters in JSP

1. Under the directory `/WEB-INF/`**`jsp`**`/`, create the following JSP page `visit.jsp`:

```
<%@page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title><%= config.getInitParameter("title")%></title>
</head>
<body>
<jsp:useBean id="visit" scope="application"
             class="hkmu.comps380f.VisitCounter"/>
<jsp:setProperty name="visit" property="count"
                 value='<%= visit.getCount() + 1%>'/>
<h1><%= config.getInitParameter("title")%></h1>
The site <%= application.getInitParameter("title")%>
was visited for <jsp:getProperty name="visit" property="count"/> times.
</body>
</html>
```

2. In web.xml, modify the `<servlet>` and `<servlet-mapping>` for the two JSP pages, and check the following context init parameter. Use **`<jsp-file>`** instead of **`<servlet-class>`**.
   - Properties for myServlet1:
     - Mapped URL: /visit1
     - Servlet Init Parameter: “title” = “Page for myServlet1”
   - Properties for myServlet2:
     - Mapped URL: /visit2
     - Servlet Init Parameter: “title” = “Page for myServlet2”
   - Context Init Parameter: “title” = “Visitor Counter Site”

3. Delete the servlet `VisitCounterServlet.java`.

4. After successfully running the project, check the translated Servlet code for the two JSP pages. For Windows OS, it can be found under the directory:
   `%USERPROFILE%``\AppData\Local\JetBrains\IntelliJIdea2023.`***`XXX`***`\tomcat\`***`<webappID>`***
   `\work\Catalina\localhost\Lab04\org\apache\jsp\WEB_002dINF\jsp`
   Note that the correct ***`webappID`*** directory has an update time latest among all directories.
   Question: Is there any code for synchronization when using the counter JavaBean?

5. Like a Servlet, we can define the init and destroy methods for a JSP page. Add the following jspInit() and jspDestroy() methods **using JSP declarations (`<%! ... %>`)**.

```
public void jspInit() {
    System.out.println("JSP servlet " + this.getServletName() + " has started.");
}

public void jspDestroy() {
    System.out.println("JSP servlet " + this.getServletName() + " has stopped.");
}
```

JSP declarations will be put ***outside*** the service() method of the converted Servlet.
Run the project again to check when the two methods will be executed.

Task 3: Configuring JSP properties in web.xml

The output of the JSP pages contain many extra new lines created by JSP tags. Remove some of them by setting **`<trim-directive-whitespaces>`** to true in **`<jsp-config>`** in web.xml, as follows:

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <page-encoding>UTF-8</page-encoding>
        <trim-directive-whitespaces>true</trim-directive-whitespaces>
        <default-content-type>text/html</default-content-type>
    </jsp-property-group>
</jsp-config>
```

Task 4: Handling HTTP GET and POST requests with different JSP pages

In this task, we use a Servlet to display different JSP pages upon receiving HTTP GET and POST requests. The output for the POST request depends on a request parameter with multiple values.

1. Modify the servlet `MultiValueParameterServlet.java`, as follows:

```
@WebServlet(name = "multiValueParameterServlet",
        urlPatterns = "/checkboxes")
public class MultiValueParameterServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        RequestDispatcher view
                = request.getRequestDispatcher("/WEB-INF/jsp/MultiValueForm.jsp");
        view.forward(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        RequestDispatcher view
                = request.getRequestDispatcher("/WEB-INF/jsp/MultiValueResult.jsp");
        view.forward(request, response);
    }
}
```

2. Upon receiving a HTTP GET request, the following HTML form will be shown:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello User Application</title>
</head>
<body>
<form action="checkboxes" method="POST">
    Select the fruits you like to eat: <br/>
    <input type="checkbox" name="fruit" value="Banana"/> Banana <br/>
    <input type="checkbox" name="fruit" value="Apple"/> Apple <br/>
    <input type="checkbox" name="fruit" value="Orange"/> Orange <br/>
    <input type="checkbox" name="fruit" value="Guava"/> Guava <br/>
    <input type="checkbox" name="fruit" value="Kiwi"/> Kiwi <br/>
    <input type="submit" value="Submit"/>
</form>
</body>
</html>
```

3. The selected fruits will be sent via HTTP POST request to the same URL. We can access the request parameter storing the selected fruits, as follows:

```
String[] fruits = request.getParameterValues("fruit");
```

   Note that **JSP declarations (`<%! ... %>`)** will be put ***outside*** the service() method of the converted Servlet, while **JSP scriptlet (`<% ... %>`)** will be put ***inside*** the service() method.

   Question: For the above declaration, should we use JSP declaration or JSP scriptlet?

4. The output for the POST request is:

   <u>Case 1: `fruits` is null</u>

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello User Application</title>
</head>
<body>
<h1>Your Selections</h1>
You did not select any fruits
</body>
</html>
```

   <u>Case 2: `fruits` is not null</u>

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello User Application</title>
</head>
<body>
<h1>Your Selections</h1>
<ul>
    <li>Fruit 1</li>
    <li>Fruit 2</li>
    <li>...</li>
</ul>
</body>
</html>
```