

Unit - 4 JSP

(a) Write a program in java to handle servlet parameters. 8

(b) What is JSP ? Explain the JSP application design with

(a) Write a Program in Java to Handle Servlet Parameters

In this example, we will create a simple servlet that handles HTTP request parameters. The program will read parameters from the request, process them, and return a response based on those parameters.

Servlet Program to Handle Parameters

The servlet will:

1. Receive data from a form submitted via HTTP (such as a user's name and email).
2. Process that data (in this case, just display it back to the user).

We will assume that the data comes from an HTML form with method GET or POST.

HTML Form (index.html)

This HTML page collects user data and sends it to the servlet via POST or GET request.

html

Copy code

```
<!DOCTYPE html>
<html>
<head>
  <title>Servlet Parameter Example</title>
</head>
<body>
  <h2>Enter Your Details</h2>
  <form action="handleServlet" method="POST">
    Name: <input type="text" name="username"><br><br>
    Email: <input type="text" name="email"><br><br>
    <input type="submit" value="Submit">
  </form>
</body>
```

</html>

In this form:

- The form sends two parameters (username and email) to the servlet when submitted.
- The action="handleServlet" specifies that the form data will be sent to the servlet with the URL pattern /handleServlet.

Servlet Program (HandleServlet.java)

java

Copy code

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
import java.io.*;
```

```
public class HandleServlet extends HttpServlet {
```

```
    @Override
```

```
    protected void doPost(HttpServletRequest request,  
        HttpServletResponse response)
```

```
        throws ServletException, IOException {
```

```
        // Get the "username" and "email" parameters from the request  
        String username = request.getParameter("username");  
        String email = request.getParameter("email");
```

```
        // Set the response content type to HTML  
        response.setContentType("text/html");
```

```
        // Get the output stream to send the response to the client  
        PrintWriter out = response.getWriter();
```

```
        // Write the HTML response  
        out.println("<html>");  
        out.println("<head><title>Servlet Parameters</title></head>");  
        out.println("<body>");  
        out.println("<h2>Submitted Parameters</h2>");  
        out.println("<p>Name: " + username + "</p>");  
        out.println("<p>Email: " + email + "</p>");  
        out.println("</body>");  
        out.println("</html>");
```

```
    }
```

```
}
```

Explanation of the Servlet Code:

1. **HttpServlet:** The HandleServlet class extends HttpServlet to process HTTP requests. It overrides the doPost() method, which handles POST requests. If you want to handle GET requests, you can override doGet() instead.
2. **request.getParameter():** This method retrieves the parameters sent in the HTTP request. In this case, it retrieves the username and email parameters from the form.
3. **Response:** The PrintWriter object (out) is used to send the response back to the client. The servlet generates an HTML response displaying the submitted values.
4. **Deployment:** This servlet should be configured in the web.xml file or via annotations (if using servlet 3.0 or higher).

web.xml Configuration

If you're using web.xml to configure the servlet, it would look something like this:

xml

Copy code

```
<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">
  <servlet>
    <servlet-name>HandleServlet</servlet-name>
    <servlet-class>HandleServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HandleServlet</servlet-name>
    <url-pattern>/handleServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

How to Run the Servlet Example:

1. **Set up a servlet container** like Apache Tomcat.
2. **Compile** the servlet code and place it in the appropriate directory (WEB-INF/classes).
3. **Deploy** the index.html page and HandleServlet to the servlet container.
4. Open a browser and go to the URL where the servlet is hosted (e.g., http://localhost:8080/YourWebApp/index.html).
5. Enter values in the form and submit. The servlet will display

the parameters.

(b) What is JSP? Explain the JSP Application Design.

What is JSP (JavaServer Pages)?

JavaServer Pages (JSP) is a server-side technology that enables the creation of dynamic, data-driven web pages. It allows developers to embed Java code in HTML pages by using special tags, making it easier to build dynamic content without writing a lot of Java code in the servlet.

JSP files are compiled into servlets by the server at runtime. The key difference between JSP and servlets is that JSP allows you to separate the presentation logic (HTML) from the business logic (Java code), making it more maintainable and readable.

Basic JSP Syntax:

1. **Directives:** Provide global information to the JSP page, such as imports or page settings.

jsp

Copy code

```
<%@ page language="java" contentType="text/html;  
charset=ISO-8859-1" %>
```

2.

3. **Scriptlets:** Embedded Java code within the HTML (less common in modern JSP due to MVC).

jsp

Copy code

```
<% int x = 10; %>
```

4. `<p>The value of x is <%= x %></p>`

5.

6. **Expressions:** A shorthand for writing Java code to print the

value in the HTML.

jsp

Copy code

```
<%= "Hello, world!" %>
```

7.

8. **Declarations:** Declaring variables or methods within the JSP page.

jsp

Copy code

```
<%! int x = 0; %>
```

JSP Lifecycle

- **Translation Phase:** The JSP file is converted into a servlet.
- **Compilation Phase:** The generated servlet is compiled.
- **Execution Phase:** The servlet is executed to generate dynamic content.

JSP Application Design

1. **MVC Architecture (Model-View-Controller):** JSP applications often use the **Model-View-Controller (MVC)** pattern to separate the concerns of data, business logic, and presentation:
 - **Model:** The backend code that handles the data and business logic (e.g., Java classes, database access).
 - **View:** The JSP pages, which are responsible for rendering the HTML to the client and displaying dynamic data.
 - **Controller:** The servlets or frameworks (e.g., Struts, Spring MVC) that handle HTTP requests, decide which view (JSP) to display, and manage interactions between the model and view.
2. **JSP for View Layer:**
 - JSP pages typically act as the **View** in the MVC

architecture. They receive data (such as user input or database results) from the controller (Servlet) and display it on the web page.

3. **Servlet as Controller:**

- A servlet acts as the **Controller**, receiving HTTP requests, interacting with business logic (models), and forwarding the results to the appropriate JSP page.

4. **Model:**

- Java classes or beans (e.g., User class, Product class) represent the **Model**. They encapsulate the data and the business logic.

JSP Application Design Example (MVC)

1. **Model (JavaBean):**

java

Copy code

```
public class User {  
2.  private String name;  
3.  private String email;  
4.  
5.  // Getters and Setters  
6.  public String getName() { return name; }  
7.  public void setName(String name) { this.name = name; }  
8.  public String getEmail() { return email; }  
9.  public void setEmail(String email) { this.email = email; }  
10. }  
11.
```

12. **Controller (Servlet):**

java

Copy code

```
import javax.servlet.*;  
13. import javax.servlet.http.*;  
14. import java.io.*;  
15.  
16. public class UserServicelet extends HttpServlet {
```

```

17.    protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
18.        throws ServletException, IOException {
19.        // Creating User model
20.        User user = new User();
21.        user.setName("John Doe");
22.        user.setEmail("john.doe@example.com");
23.
24.        // Set user object in request scope
25.        request.setAttribute("user", user);
26.
27.        // Forward request to JSP
28.        RequestDispatcher dispatcher =
        request.getRequestDispatcher("userDetails.jsp");
29.        dispatcher.forward(request, response);
30.    }
31.}
32.

```

33. **View (JSP):**

jsp

Copy code

```

    <html>
34. <body>
35.    <h2>User Details</h2>
36.    <p>Name: <%= request.getAttribute("user").getName()
    %></p>
37.    <p>Email: <%= request.getAttribute("user").getEmail()
    %></p>
38. </body>
39. </html>
40.

```

41. **web.xml Configuration:**

xml

Copy code

```
    <web-app>
42.   <servlet>
43.       <servlet-name>UserServlet</servlet-name>
44.       <servlet-class>UserServlet</servlet-class>
45.   </servlet>
46.   <servlet-mapping>
47.       <servlet-name>UserServlet</servlet-name>
48.       <url-pattern>/user</url-pattern>
49.   </servlet-mapping>
50. </web-app>
```

Introduction to JSP, Problem with Servlet, Anatomy of JSP page, JSP Processing, JSP Application Design with MVC, Setting Up and JSP Environment:, Installing the Java Software Development Kit, Tomcat Server & Testing Tomcat.

ChatGPT said:

ChatGPT

Introduction to JSP (JavaServer Pages)

JavaServer Pages (JSP) is a technology that helps developers create dynamic, data-driven web pages. It allows embedding Java code directly into HTML using special tags. JSP pages are compiled into servlets, which makes them an integral part of Java web development. It is commonly used in combination with other technologies like Servlets, Enterprise JavaBeans (EJB), and databases to create enterprise-level applications.

Problems with Servlets

While Servlets are a powerful tool for web development, they can have certain drawbacks:

1. **Complexity of HTML generation:** Servlets require developers to manually write the HTML code within Java classes, which can lead to complex and hard-to-maintain code.
2. **Separation of concerns:** Servlets mix logic (Java code) and presentation (HTML) together, violating the **separation of**

concerns principle.

3. **UI Maintenance:** When the UI needs to be changed, developers have to go back and modify the Java code, making the maintenance process more cumbersome.

JSP was designed to address these issues by allowing HTML and Java code to be separated, making web development more manageable and maintainable.

Anatomy of a JSP Page

A JSP page consists of several components:

1. **Directives:** These provide global information about the JSP page, such as the page language or error handling.

- Example: `<%@ page language="java" contentType="text/html; charset=ISO-8859-1" %>`

2. **Declarations:** These allow developers to declare variables and methods that are accessible to the entire page.

- Example: `jsp`
Copy code

```
<%! int counter = 0; %>
```

-

3. **Scripting Elements:** These are used to embed Java code within the JSP page.

- Expressions: Used for generating output.jsp
Copy code

```
<%= "Hello, World!" %>
```

-

- Scriptlets: Used for embedding Java code that is executed during page processing.jsp
Copy code

```
<%
```

- `int counter = 0;`

- counter++;
- %>
-

- **Declarations:** Used to declare methods or variables at the page level.

4. **Standard Actions:** These are predefined JSP tags that perform specific tasks.

- <jsp:include>: Includes content from another page.
- <jsp:forward>: Forwards the request to another resource.
- <jsp:useBean>: Defines a JavaBean for use in the page.

5. **Expression Language (EL):** A simplified way to access data stored in JavaBeans, request parameters, or session attributes.

- Example: \${user.name} would access the name property of a user bean.

6. **HTML:** JSP pages contain standard HTML content interspersed with Java code.

JSP Processing

1. **Request:** When a client (browser) sends a request, the JSP engine receives it.
2. **Compilation:** The JSP engine compiles the JSP file into a Servlet. This is done automatically the first time the JSP is requested or when the page changes.
3. **Execution:** The servlet executes the Java code and generates the resulting HTML. The servlet is then sent back to the client.

The primary benefit of JSP is its ability to separate content generation from business logic by letting you embed Java directly into HTML, which allows for easier maintenance and cleaner code.

JSP Application Design with MVC (Model-View-Controller)

JSP works very well with the **MVC architecture**, which separates an application into three components:

1. **Model:** The business logic and data of the application, often represented as JavaBeans or POJOs (Plain Old Java Objects).
2. **View:** The presentation layer (JSP pages) that displays data to the user.

3. **Controller:** The part of the application that handles user input and updates the model or view accordingly. This is typically implemented using Servlets.

In a typical MVC flow:

- The **Controller (Servlet)** receives a request from the user, processes the data (business logic), and sets attributes in the **Model**.
- The **Model** (JavaBeans) stores the data and may interact with a database.
- The **View (JSP)** displays the data received from the model to the user.

Setting Up and JSP Environment

To work with JSP, you need to set up the following components:

1. Installing the Java Software Development Kit (JDK)

- **Download the JDK** from Oracle's official site or from other OpenJDK distributions.
- **Install the JDK** following the platform-specific instructions.
- Verify the installation by running `java -version` and `javac -version` in the command line to check if Java is properly installed.

2. Installing Tomcat Server

Tomcat is one of the most popular web servers and servlet containers that supports JSP and Servlets.

- **Download Tomcat:** You can download the latest version of Tomcat from the [Apache Tomcat website](#).
- **Install Tomcat:** Extract the downloaded zip/tar file to a location on your system.
- **Configure Tomcat:**
 - Set the `CATALINA_HOME` environment variable to point to your Tomcat installation directory.
 - Update the `bin/setenv.sh` (or `setenv.bat` for Windows) script if needed to configure the JVM or memory settings.

3. Testing Tomcat

After installing Tomcat:

- Start Tomcat by running the `bin/startup.sh` (or `startup.bat` on Windows).

- Open a browser and go to <http://localhost:8080>. You should see the Tomcat welcome page.
- To test JSP functionality, deploy a simple .jsp file under the webapps directory in Tomcat and access it via a browser.

Conclusion

JSP simplifies web development by separating Java code from the presentation layer. When used with Servlets, JSP enables developers to create dynamic web applications using the **MVC architecture**, improving maintainability and scalability. Setting up the JSP environment involves installing the Java Development Kit (JDK) and a servlet container like Apache Tomcat, and configuring them correctly. Once the environment is set up, you can start building and testing JSP-based applications.

4o mini