Module – 4

Servlet Programming

What is a Servlet?

A Servlet is a Java class used to handle HTTP requests and generate responses in a web application.

It runs on a Java-enabled web server (like Tomcat).

Servlet Structure in Java

A Servlet is a Java class that handles HTTP requests and responses in a web application. It is part of the Java EE (Jakarta EE) platform and runs on a servlet container (like Apache Tomcat).

Simple Servlet Structure:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet extends HttpServlet {
 public void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {
       res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<h2>Hello from Servlet</h2>");}}
```

Servlet structure explained

extends HttpServlet: Allows the class to handle HTTP requests.

doGet(): Handles GET requests (like clicking a link).

doPost(): Handles POST requests (like form submission).

HttpServletRequest: Brings data from the client to the server.

HttpServletResponse: Sends data from the server to the client.

PrintWriter: Writes HTML content in the response

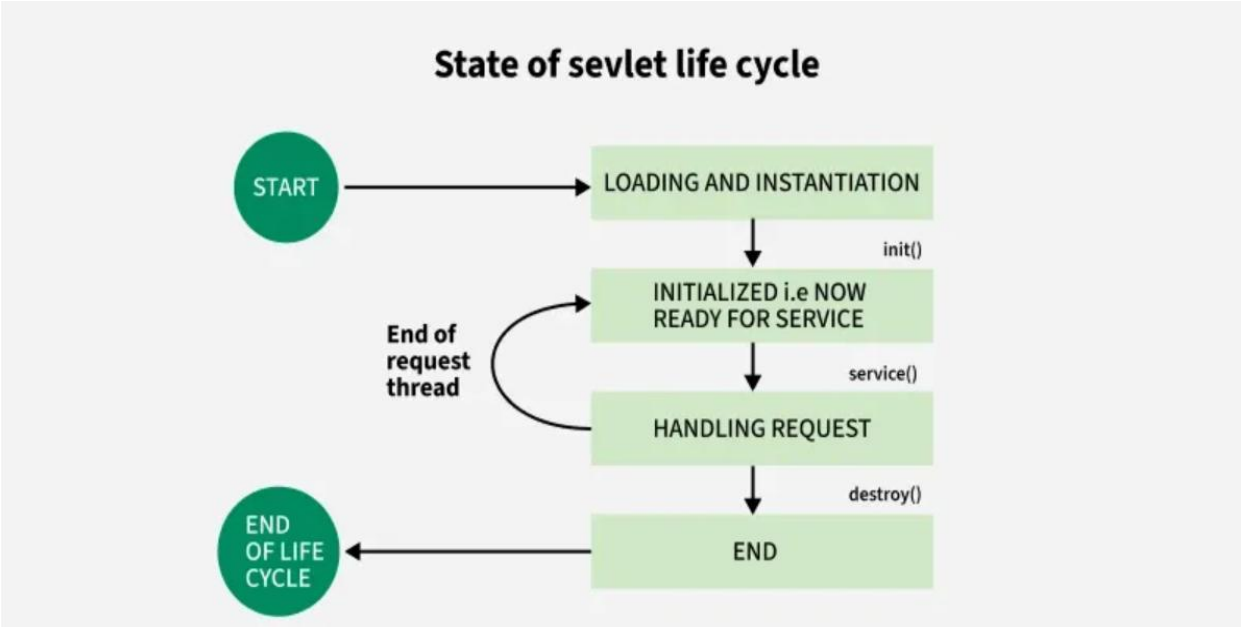
Servlet Lifecycle

The lifecycle of a Java Servlet is managed by the servlet container and involves three primary methods:

init(),

service()

destroy()



Initialization (init()):

- **When it's called:** Once when the servlet is first loaded into memory. To perform one-time setup tasks, such as establishing database connections or initializing resources.
- **EXAMPLE**
- public void init() throws ServletException {

// Initialization code here

}

Service (service()):

When it's called: Each time the servlet handles a client request.

Purpose: To process incoming requests and generate appropriate responses.

Mechanism: The service() method determines the request type GET, POST)

Example:

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {

// Request handling code here

}

Destruction (destroy()):

- **When it's called:** Once when the servlet is being removed from service, typically when the server is shutting down or the servlet is being reloaded.
- **Example:**public void destroy() {

// Cleanup code here}

HTTP Request:

An HTTP request is a message sent by the client to the server, initiating an action.

Request Line: Specifies the HTTP method (e.g., GET, POST), the resource URL, and the HTTP version.

Headers: Provide additional information about the request, such as the client's browser type, the formats it can accept, and other metadata.

Body (Optional): Contains data sent to the server, typically used with methods like POST to submit form data.

HTTP Response:

Upon receiving a request, the server processes it and returns an HTTP response, which includes:

1. **Status Line:** Indicates the HTTP version, a status code representing the outcome of the request (e.g., 200 OK, 404 Not Found), and a reason phrase.
2. **Headers:** Provide metadata about the response, such as content type, length, and caching policies.

Accessing Form Data:

When a client submits a form, the data is sent to the server as part of the HTTP request. In a servlet, you can access this data using the **getParameter()** method of the **HttpServletRequest.**

HTTP Request Headers:

HTTP request headers are key-value pairs sent by the client (usually a web browser) to the server as part of an HTTP request. They provide additional information about the request or the client itself. Common

HTTP request headers include:

User-Agent: Provides information about the client software, such as the browser type and version.

Accept: Informs the server about the media types the client can process.

Content-Type: Indicates the media type of the request body, useful when sending data to the server.

Authorization: Contains credentials for authenticating the client with the server.

Cookie: Sends previously stored cookies from the client to the server.

HTTP Status Codes:

- **1xx (Informational):** The request was received, and the process is continuing.
- **2xx (Successful):** The request was successfully received, understood, and accepted**.**
- **3xx (Redirection):** Further action is needed to complete the request.
- **4xx (Client Error):** The request contains bad syntax or cannot be fulfilled.
- **5xx (Server Error):** The server failed to fulfill a valid request.

COOKIES

cookie is a small piece of information sent by the server to any client

Types of Cookies in JSP

Persistent Cookie: Persistent cookies means the cookie data will be available even though we close the browser up to a specified time. If we want to get persistent cookies we have to use the setMaxAge() method.

Non-Persistent Cookie: Non-persistent cookies means the cookie data will be deleted when we close the browser immediately. By default, we get non-persistent cookies.

How to handle Cookie?

Handling cookie involves below three steps:

Creating a cookie object

Setting the maximum age of cookie

Sending cookie into the HTTP response headers

andling cookies in Java Servlets involves three primary steps:

Creating a Cookie Object: Instantiate a Cookie object by specifying a name and a value:

```
Cookie userCookie = new Cookie("username", "JohnDoe");
```

Setting the Maximum Age of the Cookie: Define the lifespan of the cookie using the setMaxAge(int seconds) method. For example, to set the cookie's lifespan to 30 minutes:

```
userCookie.setMaxAge(30 * 60); // 30 minutes
```

Setting the maximum age to zero deletes the cookie immediately, while a negative value indicates that the cookie is not stored persistently and will be deleted when the browser exits.

Sending the Cookie in the HTTP Response Headers: Add the cookie to the HTTP response, which sends it to the client's browser

```
response.addCookie(userCookie);
```

SESSION TRACKING

Common Techniques for Session Tracking:

1. **Cookies:**

- Cookies are small pieces of data stored on the client's browser. The server sends a cookie to the client, which the browser stores and includes in subsequent requests to the same server. This allows the server to identify returning users and maintain session information.

2. **URL Rewriting:**

- In situations where cookies are disabled, URL rewriting can be used. This involves appending a unique session identifier to the URLs of the application. For example: http://example.com/dashboard.jsp;jsessionid=123456. This ensures that the session ID is transmitted with each request**.**

3. **Hidden Form Fields:**

- Session information can be embedded within hidden fields of HTML forms. When the form is submitted, the session data is sent to the server, allowing it to maintain the user's state across requests.

4. **HttpSession:**

- In Java Servlets, the HttpSession interface provides a way to manage user sessions. It allows storing and retrieving user-specific information on the server side, maintaining state across multiple requests.

JSP PROGRAMMING

NEED OF JSP

To Create Dynamic Web Pages

JSP allows dynamic content generation using Java, unlike static HTML.

Simplifies Servlet-Based Development

Easier to write and manage than servlets where HTML is mixed with Java code.

Separation of Concerns (MVC Architecture)

JSP handles presentation; business logic is managed by Servlets/JavaBeans**.**

Easy Integration with Java Technologies

Works seamlessly with JDBC, EJB, Servlets, and other Java EE components.

Supports Reusability

Use of JavaBeans, custom tags, and tag libraries (JSTL) promotes reusable code.

Built-in Implicit Objects

Provides objects like request, response, session, etc., for easy access to data and session handling.

Automatic Compilation & Deployment

JSP pages are automatically compiled into servlets by the server, reducing development effort**.**

Platform Independent

JSP applications run on any server that supports Java (e.g., Apache Tomcat).

1. Scripting Elements (4 Tags)

These allow embedding Java code into HTML.

Tag Type	Syntax	Purpose
Scriptlet	<% ... %>	Java code block
Expression	<%= ... %>	Output of a Java expression
Declaration	<%! ... %>	Declare variables/methods
JSP Comment	<%-- ... --%>	Hidden comment in JSP

Scriptlet Tag – <% ... %>

Used to write Java code blocks.

```
<%
  int a = 5;
  int b = 10;
  int sum = a + b;
%>
```

Expression Tag – <%= ... %>

Outputs the result of a Java expression directly to the client.

```
<p>Sum is: <%= sum %></p>
```

Declaration Tag – <%! ... %>

Used to declare variables or methods at the class level (shared across requests).

```
<%! int counter = 0; %>
<%! String greet(String name) { return "Hi, " + name; } %>
```

Directive Tag – <%@ ... %>

Provides global settings or includes.

<%@ page language="java" contentType="text/html" %>

<%@ include file="header.jsp" %>

JSP Comment – <%-- ... --%>

JSP comment not visible in browser.

<%-- This comment won't appear in the HTML output --%>

✅ **Full Example**

```
<%@ page contentType="text/html" language="java" %>
<%! int count = 0; %>
<%
  count++;
  String user = "John";
%>
<html>
<body>
   <h2>Hello <%= user %>, you're visitor number <%= count %></h2>
</body>
</html>
```

What is a JSP Expression?

A **JSP Expression** is used to insert the **result of a Java expression directly into the output (HTML page)** sent to the client.

Syntax:

<%= expression %>

EXAMPLE:

```
<%
  String name = "Ravi";
  int x = 5, y = 10;
%>
<h2>Hello <%= name %></h2>
<p>Sum of x and y is: <%= x + y %></p>
```

What are JSP Directives?

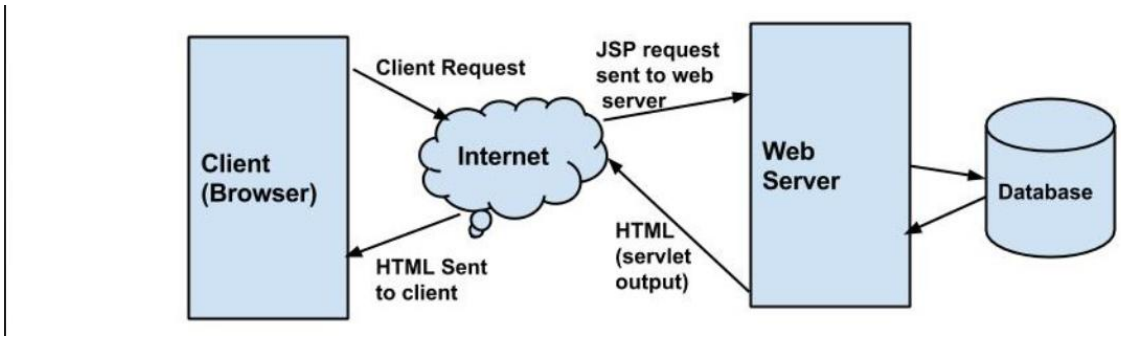
- JSP directives provide **global instructions** to the JSP engine.
- They affect the **entire JSP page** (not just specific parts).
- **Syntax:**

```
<%@ directive attribute="value" %>
<%@ page attribute="value" %>
<%@ include file="filename.jsp" %>
<%@ taglib prefix="prefix" uri="URI" %
```

Directive	Purpose	Syntax Example
page	Defines page settings	<%@ page contentType="text/html" %>
include	Includes another JSP file (static)	<%@ include file="header.jsp" %>
taglib	Declares tag library for custom tags	<%@ taglib prefix="c" uri="..." %>

SAME FOR JSP ATTRIBUTES

ARCHITECTURE FOR REFERENCES



JSP Lifecycle Methods

Method	Purpose
jspInit()	Called once when JSP is initialized
jspService()	Called for every client request
jspDestroy()	Called once before JSP is destroyed