

Java Servlet API

Server-Side Technologies:

Server-side technology refers to the software and frameworks used to develop and run applications on the server, handling the processing of data and delivering dynamic content to the client (such as a web browser). These technologies manage tasks such as database interaction, business logic, and user authentication. When a client (like a browser) makes a request (e.g., submitting a form), the server-side technology processes the request, communicates with the database if needed, and returns a response (usually an HTML page or data like JSON) to the client.

Different Server-Side Technologies:

- **Java (Java EE or Jakarta EE):** Java is a popular object-oriented programming language, and Java Enterprise Edition (Java EE) (now known as Jakarta EE) is its platform for developing large-scale, enterprise-level applications. It includes technologies like **Servlets**, **JSP** (JavaServer Pages), and **Enterprise JavaBeans** (EJB) to build dynamic web applications and APIs.
 - **Popular Frameworks:**
 - **Spring:** A powerful Java framework used to build Java-based web applications, including Spring MVC for web development.
 - **Struts:** A framework for developing Java-based web applications.
- **ASP.NET (C#):** ASP.NET is a server-side framework developed by Microsoft for building dynamic web applications and services using C#. It provides robust tools and libraries for creating large, scalable web applications.
 - **Popular Frameworks:**
 - **ASP.NET Core:** A cross-platform version of ASP.NET, designed for building modern, cloud-based, and internet-connected applications.
- **Node.js (JavaScript):** Node.js is a server-side runtime environment that allows you to run JavaScript on the server. It is event-driven, non-blocking, and highly efficient for building scalable, real-time applications.
 - **Popular Frameworks:**
 - **Express.js:** A minimal and flexible Node.js web framework that provides a set of robust features for web and mobile applications.
 - **NestJS:** A progressive Node.js framework for building efficient and scalable server-side applications.
- **PHP (Hypertext Preprocessor):** PHP is a server-side scripting language designed for web development. It is especially suited for creating dynamic and interactive websites and is widely used because of its simplicity and integration with databases like MySQL.
 - **Popular Frameworks:**
 - **Laravel:** A modern PHP framework that simplifies development with elegant syntax and a range of built-in features like routing, authentication, and database management.

- **Symfony:** A PHP framework for building scalable, high-performance web applications.
- **Python:** Python is a versatile, high-level programming language widely used for web development, data science, and machine learning. It is known for its readability and ease of use.
 - **Popular Frameworks:**
 - **Django:** A high-level Python web framework that encourages rapid development and clean, pragmatic design. It includes tools for authentication, database management, and URL routing.
 - **Flask:** A lightweight micro-framework for Python that is flexible and easy to extend.
- **Ruby on Rails:** Ruby on Rails (or Rails) is a server-side web application framework written in Ruby. It follows the Model-View-Controller (MVC) pattern and emphasizes convention over configuration, making development faster and more efficient.
 - **Popular Frameworks:** Rails itself is the primary framework for Ruby on Rails applications.
- **Go (Golang):** Go, or Golang, is a statically typed, compiled programming language created by Google. It is known for its performance, efficiency, and simplicity, making it well-suited for building scalable web applications and APIs.
 - **Popular Frameworks:**
 - **Gin:** A high-performance Go web framework.
 - **Echo:** A minimal and fast Go web framework for building scalable APIs.

What is Java EE or Jakarta EE?

Java EE (Java Enterprise Edition) and Jakarta EE are frameworks designed to simplify the development of enterprise-level applications in Java. They provide a standardized set of APIs and services that enable developers to build scalable, robust, and secure applications. Here's a detailed explanation of Java EE and Jakarta EE:

Java EE (Java Enterprise Edition): Java EE is a set of specifications and APIs that extend the core Java SE (Standard Edition) platform to support enterprise-level features. It is designed to simplify the development of large-scale, distributed, and transactional applications. Java EE provides a standardized approach to building enterprise applications, making it easier to develop, deploy, and manage complex systems.

The key Components of Java EE are

- **Servlet API:** For handling HTTP requests and generating dynamic web content.
- **JavaServer Pages (JSP):** For creating dynamic web pages using a combination of HTML and Java code.
- **Enterprise JavaBeans (EJB):** For building distributed, transactional, and secure business components.
- **Java Persistence API (JPA):** For object-relational mapping and database access.
- **Java Message Service (JMS):** For messaging and asynchronous communication between applications.

- **JavaServer Faces (JSF):** For building user interfaces for web applications.
- **Contexts and Dependency Injection (CDI):** For managing the lifecycle and dependencies of components.
- **Web Services:** For building and consuming web services using SOAP and REST.
- **Java Transaction API (JTA):** For managing transactions across multiple resources.
- **JavaMail API:** For sending and receiving emails.

Jakarta EE: Jakarta EE is the successor to Java EE. In 2017, Oracle transferred the stewardship of Java EE to the Eclipse Foundation, and it was rebranded as Jakarta EE. The goal was to foster innovation, increase community involvement, and accelerate the evolution of the platform. Jakarta EE builds on the foundation of Java EE and continues to provide a standardized set of APIs for enterprise application development.

Jakarta EE includes all the components of Java EE, with some enhancements and updates. The key components are:

- **Jakarta Servlet:** For handling HTTP requests and generating dynamic web content.
- **Jakarta Server Pages (JSP):** For creating dynamic web pages.
- **Jakarta Enterprise Beans (EJB):** For building distributed, transactional, and secure business components.
- **Jakarta Persistence:** For object-relational mapping and database access.
- **Jakarta Messaging:** For messaging and asynchronous communication between applications.
- **Jakarta Faces:** For building user interfaces for web applications.
- **Jakarta Contexts and Dependency Injection (CDI):** For managing the lifecycle and dependencies of components.
- **Jakarta RESTful Web Services:** For building and consuming RESTful web services.
- **Jakarta Transaction:** For managing transactions across multiple resources.
- **Jakarta Mail:** For sending and receiving emails.

Comparison between Java EE and Jakarta EE:

- **Name and Governance:** Java EE was governed by Oracle, while Jakarta EE is governed by the Eclipse Foundation.
- **Community Involvement:** Jakarta EE has a more open and community-driven development process.
- **Namespace:** Java EE uses the javax namespace, while Jakarta EE uses the jakarta namespace.
- **Innovation:** Jakarta EE aims to accelerate innovation and modernize the platform to keep up with industry trends.

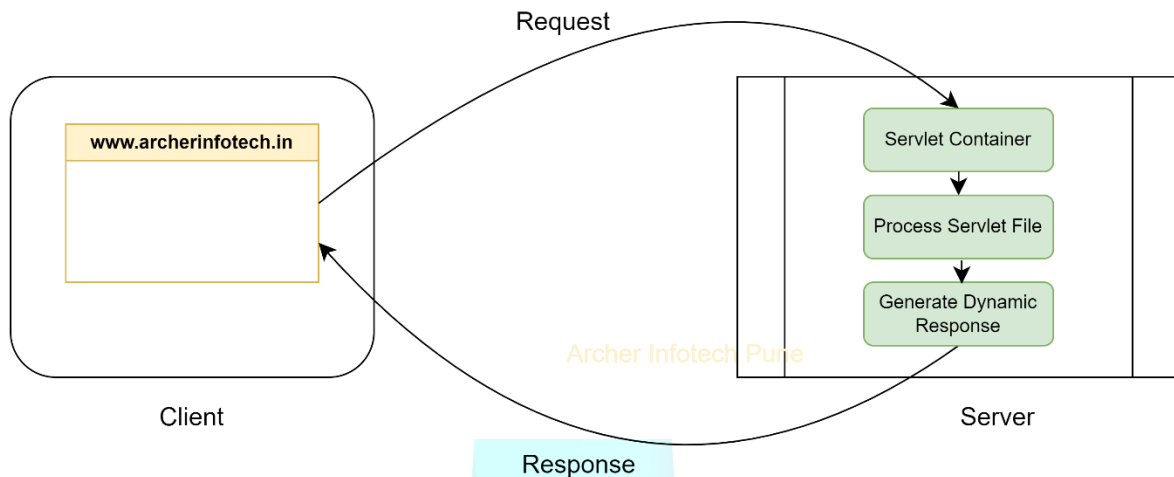
Servlet: A servlet in Java is a server-side program that extends the capabilities of a web server by generating dynamic content. Servlets are part of the Java Enterprise Edition (Java EE) and are used to **handle requests and generate responses in a web application**. They are typically used to create dynamic web content, process form data, manage user sessions, and interact with databases.

Different Versions of Servlet

The Servlet API has evolved significantly to add new features, enhance performance, and improve web functionality. Here's a list of major Servlet versions:

1. **Servlet 1.0** (1997):
 - Initial release.
 - Introduced basic functionality like handling HTTP requests and responses.
2. **Servlet 2.0** (1998):
 - Introduced as part of J2EE.
 - Added session tracking, which allows storing user information between requests.
3. **Servlet 2.1** (1999):
 - Introduced the `RequestDispatcher` interface to forward requests from one servlet to another.
 - Enhanced security with improved request and response management.
4. **Servlet 2.2** (2000):
 - Introduced web applications and `web.xml` deployment descriptor.
 - Added support for application-scoped servlets and filters.
5. **Servlet 2.3** (2001):
 - Introduced Filters, which allow request and response preprocessing.
 - Improved session handling and event listeners for handling application and session lifecycle events.
6. **Servlet 2.4** (2003):
 - Enhanced support for XML configuration and introduced web services features.
 - Allowed for more flexible request handling.
7. **Servlet 2.5** (2005):
 - Introduced annotations to simplify configuration, reducing the need for `web.xml`.
 - Added compatibility with Java EE 5.
8. **Servlet 3.0** (2009):
 - Added asynchronous processing, allowing servlets to handle long-running tasks.
 - Enhanced ease of deployment with annotations and embedded servlet containers.
9. **Servlet 3.1** (2013):
 - Introduced non-blocking I/O for improved performance.
 - Support for HTTP protocol upgrades and improved security.
10. **Servlet 4.0** (2017):
 - Introduced HTTP/2 support, improving communication efficiency and speed.
 - Enhanced support for server push and new HTTP client features.
11. **Servlet 5.0** (2020):
 - Released with Jakarta EE 9.
 - Primarily involved namespace change from `javax.servlet` to `jakarta.servlet` due to trademarking changes.
 - Continued to enhance HTTP/2 and web container features.

Official Site: <https://projects.eclipse.org/projects/ee4j.servlet>



Servlet Container: A Servlet Container is a part of a web server that interacts with the servlets. It manages the servlet lifecycle and acts as an intermediary between the servlet and the client. Some popular servlet containers include: Apache Tomcat, Jetty, GlassFish.

Servlet Lifecycle:

The servlet lifecycle is a well-defined sequence of stages that a servlet goes through from its creation to its destruction. Understanding the servlet lifecycle is crucial for developing efficient and robust web applications. Here's a detailed explanation of each stage in the servlet lifecycle:

The Servlet Life Cycle consists of five main phases:

1. Loading and Instantiation:
2. Initialization (init method)
3. Request Handling (service method)
4. Request Handling Methods (doGet, doPost, etc.)
5. Destruction (destroy method)

1. Loading and Instantiation:

- **Loading:** The servlet container (e.g., Apache Tomcat) loads the servlet class into memory. This typically happens when the servlet is first requested or when the container starts up, depending on the configuration.
- **Instantiation:** The servlet container creates an instance of the servlet class using the default constructor. This instance is used to handle all requests to that servlet.

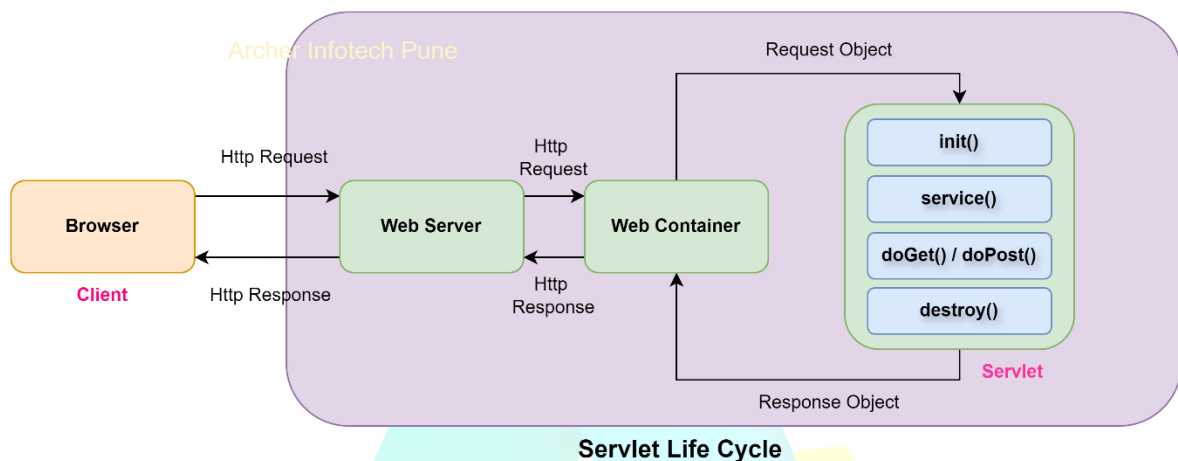
2. Initialization (init method):

The `init()` method is called once after the servlet is instantiated. It is used to perform any initialization tasks, such as setting up database connections, loading configuration files, or initializing resources.

- **Signature:** `public void init(ServletConfig config) throws ServletException`
- **Parameters:**

- **ServletConfig config:** Provides the servlet with information about its initialization parameters and the servlet context.
- **Exceptions:**
 - **ServletException:** Thrown if the servlet encounters an error during initialization.
- **Example**

```
public void init(ServletConfig config) throws ServletException {
    super.init(config);
    // Initialization code here
}
```



3. Request Handling (service method):

The service method is called for each request the servlet receives. It determines the type of request (e.g., GET, POST) and delegates the request to the appropriate method (doGet, doPost, etc.).

- **Signature:** public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
- **Parameters:**
 - **ServletRequest req:** Represents the request from the client.
 - **ServletResponse res:** Represents the response to the client.
- **Exceptions:**
 - **ServletException:** Thrown if the servlet encounters an error while handling the request.
 - **IOException:** Thrown if an I/O error occurs while handling the request.
- **Example:**

```
protected void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    String method = req.getMethod();
    if (method.equals("GET")) {
        doGet(req, res);
    } else if (method.equals("POST")) {
        doPost(req, res);
    }
}
```



```
    } else {  
        super.service(req, res);  
    }  
}
```

4. Request Handling Methods (doGet, doPost, etc.):

These methods handle specific types of HTTP requests. The most commonly used methods are doGet and doPost.

- **Signatures:**

```
protected void doGet(HttpServletRequest req, HttpServletResponse  
res) throws ServletException, IOException
```

```
protected void doPost(HttpServletRequest req, HttpServletResponse  
res) throws ServletException, IOException
```

- **Parameters:**

- **HttpServletRequest req:** Represents the HTTP request from the client.
- **HttpServletResponse res:** Represents the HTTP response to the client.

- **Exceptions:**

- **ServletException:** Thrown if the servlet encounters an error while handling the request.
- **IOException:** Thrown if an I/O error occurs while handling the request.

- **Example:**

```
protected void doGet(HttpServletRequest req, HttpServletResponse res) throws  
ServletException, IOException {  
    res.setContentType("text/html");  
    PrintWriter out = res.getWriter();  
    out.println("<html><body>");  
    out.println("<h1>Hello, World!</h1>");  
    out.println("</body></html>");  
}  
protected void doPost(HttpServletRequest req, HttpServletResponse res)  
throws ServletException, IOException {  
    // Handle POST request here  
}
```

5. Destruction (destroy method)

The destroy method is called once when the servlet is being taken out of service. This method is used to perform any cleanup tasks, such as closing database connections or releasing resources.

Signature: *public void destroy()*

Example:

```
public void destroy() {  
    // Cleanup code here  
}
```

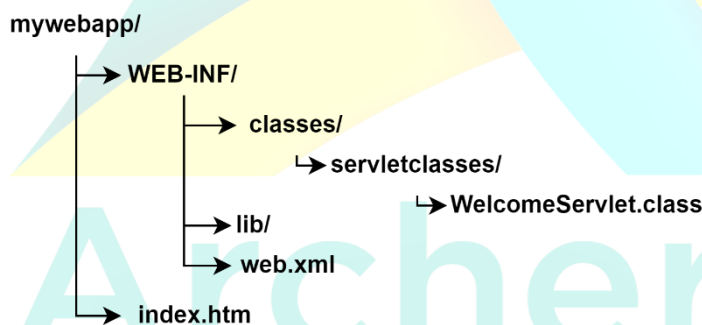
Key Features of Servlets

- **Platform Independence:** Servlets are written in Java, which makes them platform-independent. They can run on any server that supports the Java Servlet API.
- **Dynamic Content Generation:** Servlets can generate dynamic content based on user input, database queries, or other factors.
- **Session Management:** Servlets can manage user sessions, allowing for stateful interactions between the client and the server.
- **Scalability:** Servlets are designed to handle multiple requests concurrently, making them suitable for high-traffic web applications.
- **Integration with Java EE:** Servlets can easily integrate with other Java EE technologies like JavaServer Pages (JSP), Enterprise JavaBeans (EJB), and Java Message Service (JMS).

Servlet Welcome Program:

Creating a simple "Welcome" program using a servlet involves several steps, including setting up the development environment, writing the servlet code, configuring the deployment descriptor, and deploying the servlet to a servlet container. Below is a detailed procedure to create a "Welcome" program in a servlet:

- Install JDK: <https://www.oracle.com/in/java/technologies/downloads/>
 - Apache tomcat download: <https://tomcat.apache.org/download-90.cgi>
1. **Set Up the Project Structure:** Create a directory structure for your web application. The typical structure for a web application looks like as:



2. **Write the Servlet Code:** Create a Java class for your servlet. For example, create a file named 'WelcomeServlet.java' in the 'classes.servletclasses' package.


```
package servletclass;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

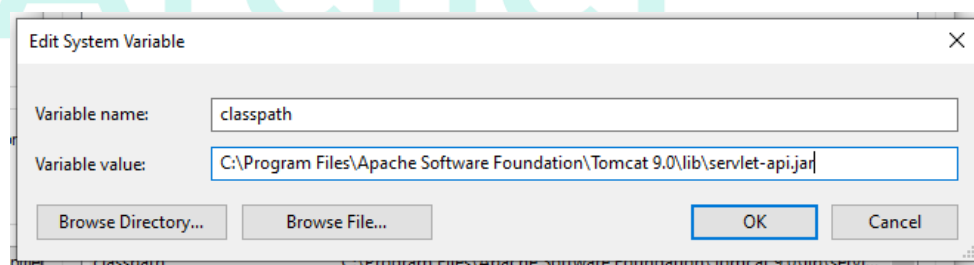
public class WelcomeServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>Welcome to the Servlet World!</h1>");
        out.println("</body></html>");
    }
}
```

- 3. Compile the Servlet:** Compile the servlet code to generate the .class file. You can use the javac command from the command line or compile it within your IDE.

For that,

- download the [servlet-api.jar](#), Place it into WEB-INF/lib, go to the classes and open cmd at same path and compile using cmd as,
C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\FirstServlet\WEB-INF\classes>javac -classpath "C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\FirstServlet\WEB-INF\lib\servlet-api.jar"; servletclass\WelcomeServlet.java
- Or
- Use the servlet-api.jar from Tomcat xx\bin as,
C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\FirstServlet\WEB-INF\classes>javac -classpath "C:\Program Files\Apache Software Foundation\Tomcat 9.0\lib\servlet-api.jar"; servletclass\WelcomeServlet.java
- Or
- Simply set classpath in Environment variable as,



- 4. Configure the Deployment Descriptor (web.xml):** Create a web.xml file in the WEB-INF directory to configure the servlet.

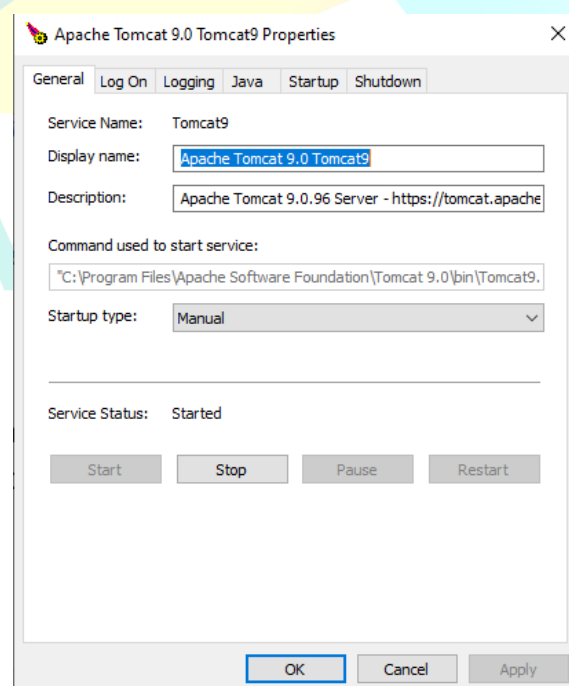
```
<web-app>
  <!-- Servlet Declaration -->
  <servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>servletclass.WelcomeServlet</servlet-class>
  </servlet>

  <!-- Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/welcome</url-pattern>
  </servlet-mapping>
</web-app>
```

5. **Create an HTML File (Optional):** Create an index.html file in the root directory of your web application to provide a link to the servlet.

```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome Page</title>
</head>
<body>
  <h1>Welcome to the Web Application@Archer Infotech Pune</h1>
  <a href="/welcome">Go to Welcome Servlet</a>
</body>
</html>
```

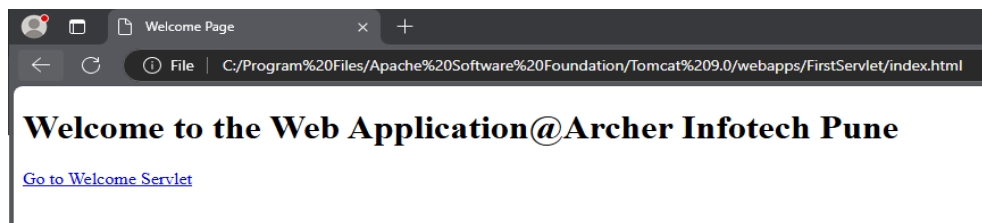
6. **Start the Servlet Container:** Start your servlet container (e.g., Apache Tomcat). You can start Tomcat using the startup.sh or startup.bat script in the bin directory of your Tomcat installation.



7. **Access the Servlet:** Open a web browser and navigate to the URL of your servlet. For example:



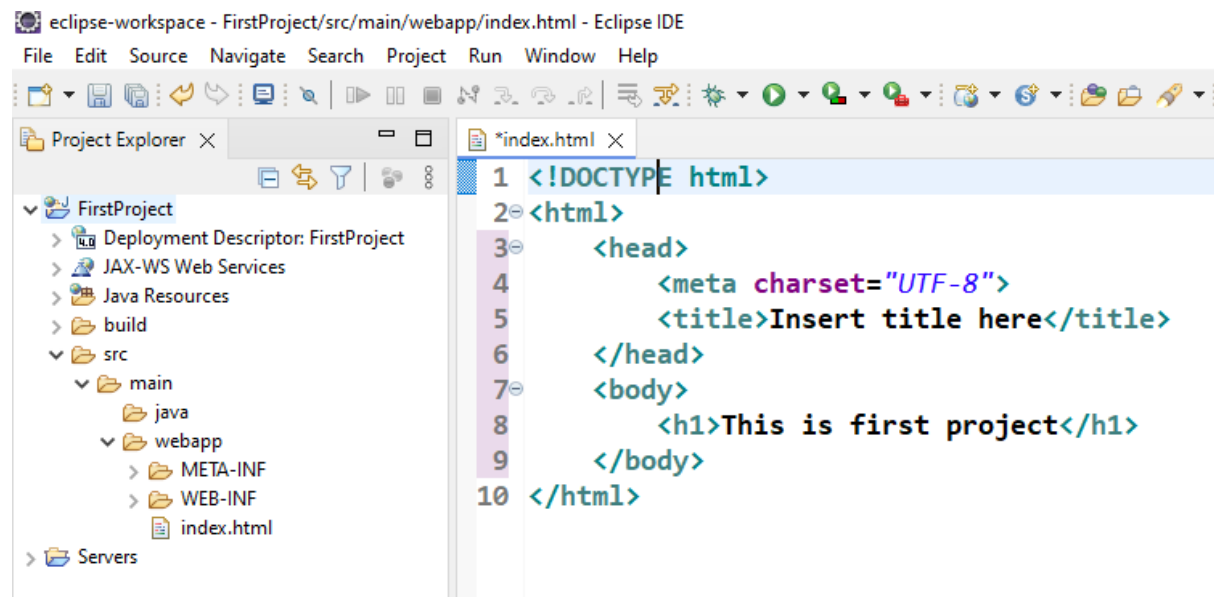
Or, Open the index.html present in the ..\webapps\FirstServlet folder



Eclipse and Apache Tomcat Configuration:

- **Download and install the Eclipse and Apache Tomcat.**
 - Eclipse IDE for Enterprise Java and Web Developers:
<https://www.eclipse.org/downloads/packages/release/2024-09/r/eclipse-ide-enterprise-java-and-web-developers>
 - Apache tomcat 9: <https://tomcat.apache.org/download-90.cgi>
- **Set Runtime Environment of Tomcat in Eclipse.**
 - In Eclipse, go to **Window > Preferences**.
 - In the Preferences dialog, navigate to **Server > Runtime Environments**.
 - Click the **Add** button.
 - In the New Server Runtime Environment dialog, select **Apache > Tomcat vX.X Server** (where X.X is the version you downloaded) and click **Next**.
 - In the Apache Tomcat dialog, click the **Browse** button and navigate to the directory where you extracted Tomcat (e.g., **C:\apache-tomcat-9.0.50**).
 - Click **Finish**.
- **Add the server.**
 - In Eclipse, go to the **Servers** tab (if it's not visible, you can open it via **Window > Show View > Servers**).
 - Right-click in the Servers tab and select **New > Server**.
 - In the New Server dialog, select **Apache > Tomcat vX.X Server** (where X.X is the version you configured) and click **Next**.
 - In the Server's host name field, you can leave the default value (**localhost**).
 - Click **Next**.

Create Dynamic project in eclipse (File → new → Dynamic Web Project → Choose Project name and finish)



Finally create the index.html in webapp, right click in editor and select Run as and then run on server. On success, installation successfully completed.

Servlet class hierarchy:

The Servlet class hierarchy in Java is designed to provide a structured and extensible framework for handling HTTP requests and generating dynamic web content. The core classes and interfaces in the Servlet API form the foundation of this hierarchy. Here's a detailed explanation of the Servlet class hierarchy:

Core Interfaces and Classes

1. javax.servlet.Servlet

- **Description:** The main **interface** that all servlets must implement. It defines the lifecycle methods (**init**, **service**, and **destroy**) that a servlet must implement.
- **Methods:**
 - **void init(ServletConfig config) throws ServletException:** Called once after the servlet is instantiated to perform initialization tasks.
 - **void service(ServletRequest req, ServletResponse res) throws ServletException, IOException:** Called for each request the servlet receives.
 - **void destroy():** Called once when the servlet is being taken out of service to perform cleanup tasks.
 - **ServletConfig getServletConfig():** Returns the **ServletConfig** object associated with the servlet.
 - **String getServletInfo():** Returns information about the servlet, such as its author, version, etc.

2. javax.servlet.GenericServlet

- **Description:** An abstract class that provides a default implementation of the **Servlet** interface. It is typically used for non-HTTP servlets.
- **Methods:**
 - **void init(ServletConfig config) throws ServletException:** Calls **init(config)** and stores the **ServletConfig** object.
 - **void service(ServletRequest req, ServletResponse res) throws ServletException, IOException:** Must be overridden by subclasses to handle requests.
 - **void destroy():** Default implementation does nothing.
 - **ServletConfig getServletConfig():** Returns the stored **ServletConfig** object.
 - **String getServletInfo():** Returns information about the servlet.

3. javax.servlet.http.HttpServlet

- **Description:** A subclass of **GenericServlet** that provides a framework for handling HTTP requests. It defines methods like **doGet**, **doPost**, **doPut**, and **doDelete**.
- **Methods:**
 - **void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException:** Handles HTTP GET requests.
 - **void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException:** Handles HTTP POST requests.
 - **void doPut(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException:** Handles HTTP PUT requests.
 - **void doDelete(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException:** Handles HTTP DELETE requests.
 - **void service(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException:** Calls the appropriate **doXXX** method based on the HTTP method.

Supporting Classes and Interfaces

4. javax.servlet.ServletConfig

- **Description:** Provides configuration information to a servlet. It is passed to the servlet's **init** method.
- **Methods:**
 - **String getServletName():** Returns the name of the servlet.
 - **ServletContext getServletContext():** Returns the **ServletContext** object for the servlet.
 - **String getInitParameter(String name):** Returns the value of the named initialization parameter.
 - **Enumeration<String> getInitParameterNames():** Returns the names of the servlet's initialization parameters.

5. javax.servlet.ServletContext

- **Description:** Provides information about the servlet's environment. It is used to access global initialization parameters and attributes.
- **Methods:**
 - **String getContextPath():** Returns the context path of the web application.
 - **String getInitParameter(String name):** Returns the value of the named context-wide initialization parameter.
 - **Enumeration<String> getInitParameterNames():** Returns the names of the context's initialization parameters.
 - **Object getAttribute(String name):** Returns the value of the named attribute.
 - **Enumeration<String> getAttributeNames():** Returns the names of the context's attributes.
 - **void setAttribute(String name, Object object):** Sets the value of the named attribute.
 - **void removeAttribute(String name):** Removes the named attribute.

6. javax.servlet.ServletRequest

- **Description:** Represents a request from a client to the server. It provides methods to retrieve request parameters, headers, and other information.
- **Methods:**
 - **String getParameter(String name):** Returns the value of the named request parameter.
 - **Enumeration<String> getParameterNames():** Returns the names of the request parameters.
 - **String[] getParameterValues(String name):** Returns the values of the named request parameter as an array.
 - **String getHeader(String name):** Returns the value of the named request header.
 - **Enumeration<String> getHeaderNames():** Returns the names of the request headers.

7. javax.servlet.ServletResponse

- **Description:** Represents a response from the server to the client. It provides methods to set response headers, status codes, and write response data.
- **Methods:**
 - **void setContentType(String type):** Sets the content type of the response.
 - **void setHeader(String name, String value):** Sets the value of the named response header.
 - **void addHeader(String name, String value):** Adds a response header with the given name and value.
 - **void setStatus(int sc):** Sets the status code of the response.
 - **PrintWriter getWriter() throws IOException:** Returns a **PrintWriter** object that can send character text to the client.

- **ServletOutputStream getOutputStream() throws IOException:** Returns a **ServletOutputStream** object that can send binary data to the client.

8. javax.servlet.http.HttpServletRequest

- **Description:** A subclass of **ServletRequest** that provides additional methods specific to HTTP requests, such as retrieving cookies, session information, and request attributes.
- **Methods:**
 - **String getMethod():** Returns the HTTP method (e.g., GET, POST) used in the request.
 - **String getRequestURI():** Returns the URI of the request.
 - **String getContextPath():** Returns the context path of the request.
 - **Cookie[] getCookies():** Returns an array of **Cookie** objects sent by the client.
 - **HttpSession getSession():** Returns the **HttpSession** object associated with the request.
 - **HttpSession getSession(boolean create):** Returns the **HttpSession** object associated with the request, optionally creating a new session if one does not exist.

9. javax.servlet.http.HttpServletResponse

- **Description:** A subclass of **ServletResponse** that provides additional methods specific to HTTP responses, such as setting cookies, redirecting requests, and setting response status codes.
- **Methods:**
 - **void addCookie(Cookie cookie):** Adds a **Cookie** to the response.
 - **void sendRedirect(String location) throws IOException:** Sends a temporary redirect response to the client.
 - **void setStatus(int sc):** Sets the status code of the response.
 - **void setHeader(String name, String value):** Sets the value of the named response header.
 - **void addHeader(String name, String value):** Adds a response header with the given name and value.

Create the Servlet:

Creating servlets in Java can be done in several ways, each with its own advantages and use cases. Here are the different methods to create servlets:

Creating Servlet by Implementing Servlet Interface:

Directly implementing the Servlet interface is generally not recommended for typical web applications due to increased complexity and reduced functionality compared to extending **HttpServlet**. Instead, it's more useful for educational purposes or specialized non-HTTP protocol implementations where the additional control can be beneficial.

Example:

Web.xml

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>arc.nov.examples.MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/myservlet</url-pattern>
  </servlet-mapping>
</web-app>
```

HTML File

```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome Page</title>
</head>
<body>
  <h1>Welcome to the Web Application</h1>
  <a href="/myservlet">Go to MyServlet</a>
</body>
</html>
```

Advantages of Implementing the Servlet Interface Directly

1. **Complete Control:** Implementing the interface gives full control over the servlet lifecycle, allowing customizations of initialization, request handling, and resource cleanup.
2. **Minimal Overhead:** There's no additional inheritance, so this can be optimal for lightweight applications where minimal structure is desired.
3. **Basic Understanding:** Good for learning the servlet lifecycle at a low level, as you must define the essential lifecycle methods explicitly.

Disadvantages of Implementing the Servlet Interface Directly

1. **More Code:** You need to implement all five methods of the Servlet interface, even if you only use init, service, and destroy, which adds boilerplate code.
2. **Limited Functionality:** Since you aren't extending HttpServlet, handling HTTP-specific actions like GET or POST requests requires extra effort. You have to manually check the request type.
3. **Not Recommended for HTTP Servlets:** In most web applications, HttpServlet provides more flexibility and is better suited to handling HTTP-specific features, so implementing the Servlet interface directly is uncommon in production.

Servlet:

```
package arc.nov.examples;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyServlet implements Servlet {
    private ServletConfig config;
    @Override
    public void init(ServletConfig config) throws ServletException {
        this.config = config;
        // Initialization code here
    }
    @Override
    public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<html><body>");
        out.println("<h1>Hello from MyServlet!</h1>");
        out.println("</body></html>");
    }
    @Override
    public void destroy() {
        // Cleanup code here
    }
    @Override
    public ServletConfig getServletConfig() {
        return config;
    }
    @Override
    public String getServletInfo() {
        return "MyServlet Information";
    }
}
```

Use Cases

Implementing the Servlet interface directly is not common in modern Java EE applications, but it is useful in specific situations:

- **Learning and Understanding Servlet Lifecycle:** For educational purposes, implementing Servlet directly can help in understanding the servlet lifecycle and internal mechanisms.
- **Non-HTTP Protocols:** If a servlet needs to handle a protocol other than HTTP, implementing Servlet directly may offer the required flexibility.

- **Low-Level Control Requirements:** If the application requires full control of request processing without the overhead of `HttpServlet`, this method can be useful.

Creating Servlet by Extending `GenericServlet`:

Creating a Java servlet by extending the `GenericServlet` class is a more streamlined way to develop servlets compared to directly implementing the `Servlet` interface. `GenericServlet` provides a base class that implements the `Servlet` and `ServletConfig` interfaces, simplifying the code by eliminating the need to implement all the methods in the `Servlet` interface.

Servlet:

```
import javax.servlet.*;
import java.io.IOException;
import java.io.PrintWriter;

public class MyGenericServlet extends GenericServlet {
    @Override
    public void service(ServletRequest request, ServletResponse response) throws ServletException,
    IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello from MyGenericServlet!</h1>");
    }
}
```

Advantages of Extending `GenericServlet`

1. **Less Boilerplate Code:** Since `GenericServlet` implements `Servlet` and `ServletConfig` interfaces, there's no need to implement methods like `init`, `destroy`, `getServletConfig`, or `getServletInfo`, unless you want to override them.
2. **Reusable Code:** `GenericServlet` provides a reusable and more organized structure, ideal for applications where the servlet isn't tied to HTTP-specific functionality (like working with generic protocols).
3. **Simplified Lifecycle Management:** The `GenericServlet` base class manages some of the lifecycle details, so you only need to override necessary methods (mainly `service`).

Disadvantages of Extending `GenericServlet`

1. **Limited HTTP Functionality:** `GenericServlet` doesn't offer built-in support for HTTP methods like `GET`, `POST`, `PUT`, or `DELETE`. For HTTP-based applications, `HttpServlet` is generally preferred.
2. **Less Control Over HTTP-Specific Features:** If the servlet needs to handle HTTP headers, cookies, or sessions, this approach requires additional handling, as `GenericServlet` is protocol-agnostic.
3. **Not Optimal for Web Applications:** Since most web applications are HTTP-based, using `GenericServlet` instead of `HttpServlet` is less common in web development.

Use Cases

Extending GenericServlet can be useful in specific scenarios:

- **Protocol-Agnostic Applications:** For applications that aren't limited to HTTP but might need to handle requests over other protocols.
- **Simple, Lightweight Servlets:** GenericServlet is suited for lightweight applications that don't need extensive HTTP functionality.
- **Learning and Experimentation:** This approach is also helpful when learning Java servlets, as it provides a balance between low-level control and simplicity without unnecessary boilerplate.

Creating Servlet by Extending HttpServlet:

Creating a Java servlet by extending the HttpServlet class is the most common approach for developing web applications in Java. HttpServlet provides a structured way to handle HTTP requests, offering built-in methods for HTTP-specific functionality like GET, POST, PUT, and DELETE.

Servlet:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;

public class MyHttpServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello from MyHttpServlet via GET!</h1>");
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello from MyHttpServlet via POST!</h1>");
    }
}
```

Advantages of Extending HttpServlet

1. **Built-in Support for HTTP Protocol:** HttpServlet provides specialized methods for handling HTTP requests (doGet, doPost, doPut, doDelete, etc.), which simplifies coding for web applications.

2. **Easy Handling of HTTP Headers and Sessions:** HttpServlet has built-in support for managing headers, cookies, sessions, and other HTTP-specific details, making it easier to build feature-rich web applications.
3. **Better Readability and Organization:** By using separate methods for each HTTP request type, code is more organized and easier to maintain. Developers can quickly identify and modify code for specific request types.
4. **Efficient and Secure:** Using HttpServlet ensures better compatibility with web standards, potentially making the application more secure and efficient.

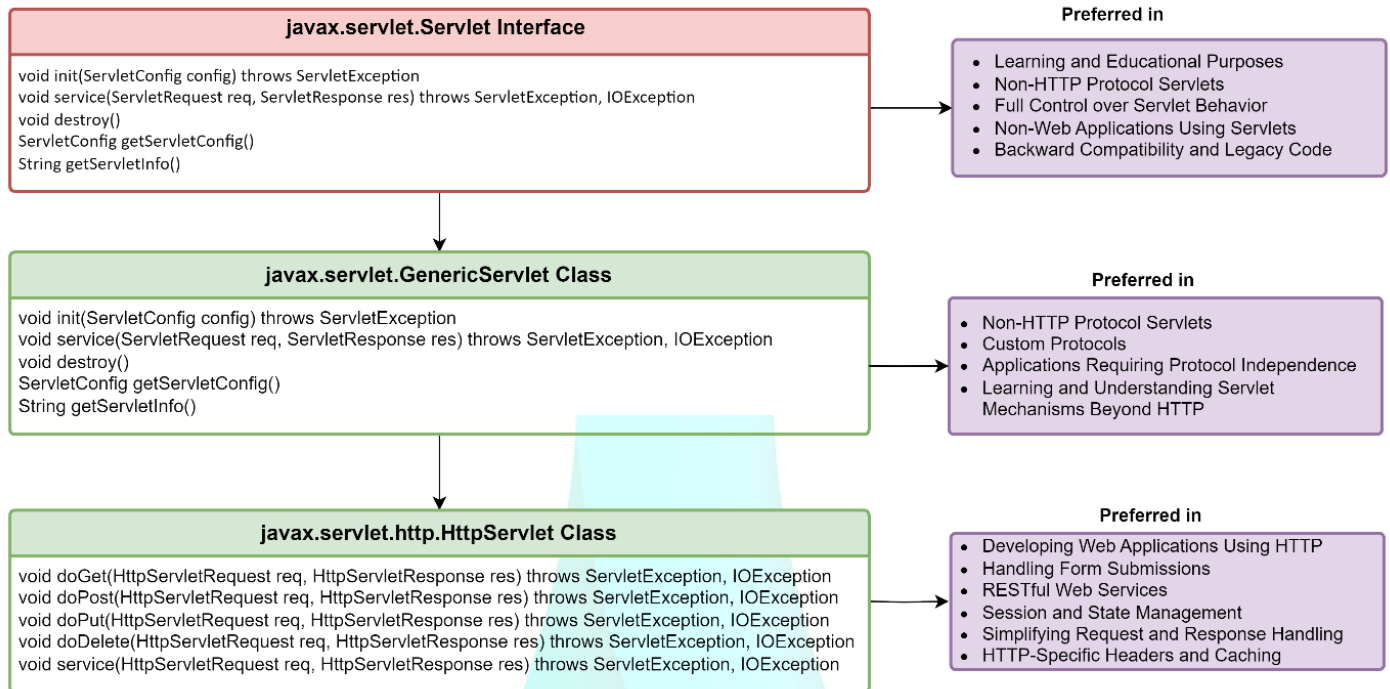
Disadvantages of Extending HttpServlet

1. **Limited to HTTP Protocol:** HttpServlet is designed exclusively for HTTP and cannot be used for other protocols without additional customization, making it unsuitable for applications that require protocol-agnostic handling.
2. **Overhead of Inheritance:** If you don't need HTTP-specific functionality, inheriting from HttpServlet may add unnecessary complexity and overhead, which could impact performance in some cases.
3. **Dependency on Web Container:** Since HttpServlet is tailored for web applications, it requires a servlet container like Apache Tomcat or Jetty to run, which may limit flexibility for other types of applications.

Use Cases

1. **Web Applications and APIs:** HttpServlet is ideal for building web applications or RESTful APIs that use HTTP methods to perform operations on resources.
2. **Interactive Forms and User Inputs:** Applications with forms where data is sent through POST requests and responses need to be generated based on user inputs.
3. **Handling Sessions and Authentication:** Web applications that require session management and user authentication benefit from the built-in HTTP support in HttpServlet.

The logo for Archer Infotech, featuring a stylized 'A' composed of two overlapping triangles, one light blue and one light yellow, with the word 'Archer' in a light blue sans-serif font below it.

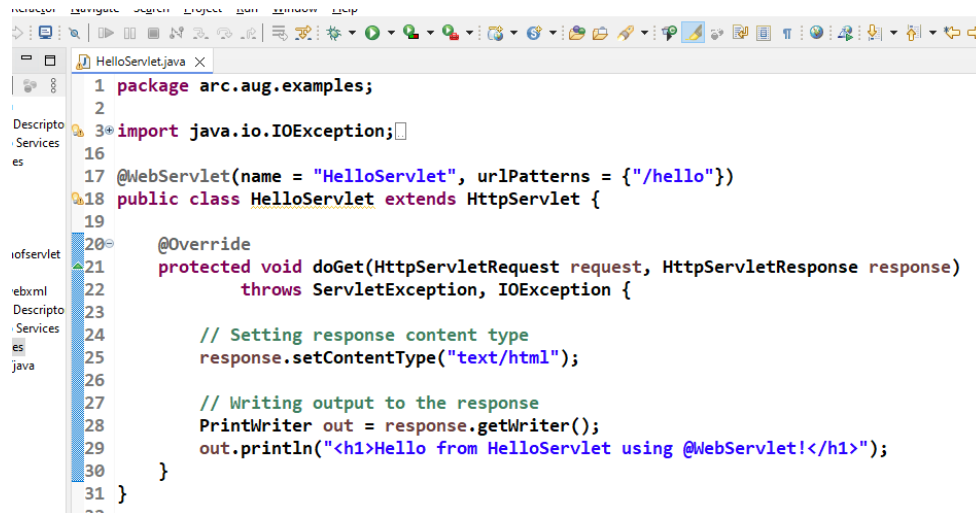


Creating a Servlet Using @WebServlet Annotation

In Java, servlets can be created and configured using annotations, specifically the `@WebServlet` annotation, which simplifies deployment by eliminating the need for web.xml configuration entries. This approach is particularly useful for developers who want a streamlined and modern way to set up servlets. The `@WebServlet` annotation is part of the `javax.servlet.annotation` package, introduced in Servlet 3.0. This annotation allows you to specify the servlet's URL pattern, initialization parameters, and more directly in the code. Here's a step-by-step example of how to create a servlet using `@WebServlet`.

Example: Basic @WebServlet Annotation

- **Import Required Packages:** First, import the necessary packages for `HttpServlet`, `HttpServletRequest`, `HttpServletResponse`, and `@WebServlet`.
- **Define the Servlet Class:** Extend the `HttpServlet` class and use the `@WebServlet` annotation above the class declaration to define the servlet's URL pattern.
- **Override HTTP Methods:** Override `doGet`, `doPost`, or any other HTTP methods needed to handle requests.



```
1 package arc.aug.examples;
2
3 import java.io.IOException;
4
5
6
7
8
9
10
11
12
13
14
15
16 @WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
17 public class HelloServlet extends HttpServlet {
18
19
20
21     @Override
22     protected void doGet(HttpServletRequest request, HttpServletResponse response)
23         throws ServletException, IOException {
24
25         // Setting response content type
26         response.setContentType("text/html");
27
28         // Writing output to the response
29         PrintWriter out = response.getWriter();
30         out.println("<h1>Hello from HelloServlet using @WebServlet!</h1>");
31     }
32 }
```

In this example:

- The `@WebServlet` annotation registers `HelloServlet` as a servlet with the name "HelloServlet", accessible via the `/hello` URL pattern.
- The `doGet` method is overridden to handle GET requests, writing a simple "Hello" message to the response.

Advantages of Using `@WebServlet`

1. **Simplifies Configuration:** Eliminates the need for `web.xml` entries, reducing configuration complexity and making the code self-contained.
2. **Improves Readability:** Configuration details are located close to the servlet code, making it easier to understand and maintain.
3. **Supports Multiple URL Patterns:** Allows registering multiple URL patterns for a single servlet, providing flexibility in how clients access it.

Use Cases for `@WebServlet`

1. **Modern Web Applications:** Using `@WebServlet` is ideal for Java EE applications that aim to be compatible with Servlet 3.0+ standards and prefer annotation-based configurations.
2. **Lightweight Configurations:** Suitable for applications where servlets don't require complex initialization, making it easy to set up.
3. **Single-Use or Simple Servlets:** Great for straightforward servlets with minimal configuration needs, such as handling single routes or endpoints.

Reading data from different form components:

Reading data from different form components in a servlet can be done using `request.getParameter()`, as each form element is sent with its name attribute in the HTTP request. Here's how to handle various types of form components:

1. **Text Fields:** Text fields are commonly used for single-line text input.

HTML Form (Text Field):

```
<form action="FormServlet" method="post">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username">
  <input type="submit" value="Submit">
</form>
```

Servlet:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String username = request.getParameter("username");
    // Process data
}
```

2. Password Fields: Password fields are similar to text fields but hide the input text.

HTML Form (Password Field)

```
<label for="password">Password:</label>
<input type="password" id="password" name="password">
```

Servlet

```
String password = request.getParameter("password");
```

3. Radio Buttons: Radio buttons allow users to select only one option from a set.

HTML Form (Radio Buttons)

```
<label>Gender:</label>
<input type="radio" name="gender" value="Male"> Male
<input type="radio" name="gender" value="Female"> Female
```

Servlet

```
String gender = request.getParameter("gender"); // Will get "Male" or "Female"
```

4. Checkboxes: Checkboxes allow multiple selections. When selected, they pass their value; otherwise, they are not sent in the request.

HTML Form (Checkboxes)

```
<label>Skills:</label>
<input type="checkbox" name="skill" value="Java"> Java
<input type="checkbox" name="skill" value="Python"> Python
<input type="checkbox" name="skill" value="JavaScript"> JavaScript
```

Servlet

```
String[] skills = request.getParameterValues("skill"); // Array of selected values
if (skills != null) {
    for (String skill : skills) {
        // Process each skill
    }
}
```

5. Drop-down (Select) Lists: Drop-down lists allow users to select one or multiple options based on the multiple attributes.

HTML Form (Single Select Drop-down):

```
<label for="country">Country:</label>
<select id="country" name="country">
  <option value="India">India</option>
  <option value="USA">USA</option>
</select>
```

Servlet:

```
String country = request.getParameter("country"); // Single selected value
```

HTML Form (Multiple Select Drop-down)

```
<label for="languages">Languages:</label>
<select id="languages" name="languages" multiple>
  <option value="Java">Java</option>
  <option value="Python">Python</option>
  <option value="JavaScript">JavaScript</option>
</select>
```

Servlet

```
String[] languages = request.getParameterValues("languages"); // Array of selected values
```

6. Hidden Fields: Hidden fields store data that is not visible to the user but sent to the server.

HTML Form (Hidden Field)

```
<input type="hidden" name="userId" value="12345">
```

Servlet

```
String userId = request.getParameter("userId");
```

7. Text Area: Text areas allow for multi-line text input.

HTML Form (Text Area)

```
<label for="comments">Comments:</label>
<textarea id="comments" name="comments"></textarea>
```

Servlet

```
String comments = request.getParameter("comments");
```

8. File Uploads: File uploads require `enctype="multipart/form-data"` in the form and need additional handling in the servlet.

HTML Form (File Upload)

```
<form method="post" action="FileUploadServlet" enctype="multipart/form-data">
    <input type="file" name="file" required/><br/><br/>
    <input type="submit" value="Upload"/>
</form>
```

Servlet

```
@WebServlet("/FileUploadServlet")
@MultipartConfig(
    fileSizeThreshold = 1024 * 1024 * 1, // 1 MB
    maxFileSize = 1024 * 1024 * 10,    // 10 MB
    maxRequestSize = 1024 * 1024 * 100 // 100 MB
)
public class FileUploadServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        // Gets the file part from the request
        Part filePart = request.getPart("file"); // "file" should match the name attribute in your form
        String fileName = getSubmittedFileName(filePart);
        // Directory where files will be saved
        String uploadPath = getServletContext().getRealPath("") + File.separator + "uploads";
        File uploadDir = new File(uploadPath);
        if (!uploadDir.exists()) {
            uploadDir.mkdir();
        }
        try { // Write the file
            filePart.write(uploadPath + File.separator + fileName);
            out.println("<html><body>");
            out.println("<h2>File " + fileName + " has been uploaded successfully!</h2>");
            out.println("</body></html>");
        } catch (Exception e) {
            out.println("<html><body>");
            out.println("<h2>Error: " + e.getMessage() + "</h2>");
            out.println("</body></html>");
        } finally {
            out.close();
        }
    }

    private String getSubmittedFileName(Part part) {
        String contentDisp = part.getHeader("content-disposition");
        String[] tokens = contentDisp.split(";");
        for (String token : tokens) {
            if (token.trim().startsWith("filename")) {
                return token.substring(token.indexOf("=") + 2, token.length()-1);
            }
        }
        return "";
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    }
}
```

9. Date Input: The date input type allows users to select a date.

HTML Form (Date Input)

```
<label for="dob">Date of Birth:</label>
<input type="date" id="dob" name="dob">
```

Servlet

```
String dob = request.getParameter("dob"); // Format: YYYY-MM-DD
```

Understanding doGet() and doPost() methods from Servlet:

In Java servlets, the doGet() and doPost() methods handle HTTP GET and POST requests, respectively. Here's an explanation of each with examples, their advantages, and when to prefer one over the other.

Understanding doGet(): The doGet() method in a servlet is used to handle HTTP GET requests. GET requests are typically used to request data from the server rather than to submit data to be processed. Here are some common use cases for doGet() in servlets:

1. Displaying a Web Page or HTML Form

- doGet() can be used to generate and display HTML content, such as a form or a webpage.
- For example, when a user visits a URL in a browser, a doGet() method might be triggered to display an HTML form or a webpage with data retrieved from the server.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><body>");
    out.println("<h1>Welcome to Our Website</h1>");
    out.println("<p>Enter your details below:</p>");
    out.println("<form action='submitForm' method='post'>");
    out.println("<input type='text' name='name' placeholder='Your Name'><br>");
    out.println("<input type='submit' value='Submit'>");
    out.println("</form>");
    out.println("</body></html>");
}
```

2. Reading Query Parameters

- GET requests often send data through URL query parameters (e.g., ?name=Yogesh&age=30). doGet() can read and process these parameters.
- Example:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String name = request.getParameter("name");
    String age = request.getParameter("age");
    out.println("<html><body>");
    out.println("<h1>Hello, " + name + "!</h1>");
    out.println("<p>Age: " + age + "</p>");
    out.println("</body></html>");
}
```


- If you accessed the servlet, you will get url in address as shown below
<http://localhost:8080/yourServlet?name=Yogesh&age=30>, and it would display Hello, Yogesh! and Age: 30.

3. Fetching Data from a Database or External Source

- doGet() can be used to retrieve data from a database or an external API and display it on the webpage.
- For example, a servlet could use doGet() to retrieve a list of products or users from a database and display them.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // Example: Fetching data from a database (pseudo-code)
    List<String> products = Database.getProductList();
    out.println("<html><body>");
    out.println("<h1>Product List</h1>");
    for (String product : products) {
        out.println("<p>" + product + "</p>");
    }
    out.println("</body></html>");
}
```

4. Implementing Search Functionality

- doGet() is often used in search forms where users submit keywords, and the servlet retrieves matching results based on the query parameter.
- For example, a search bar could send a GET request with a query parameter like ?query=java.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String query = request.getParameter("query");
    // Logic to search for results matching the query
    List<String> results = SearchService.search(query);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><body>");
    out.println("<h2>Search Results for: " + query + "</h2>");
    for (String result : results) {
        out.println("<p>" + result + "</p>");
    }
    out.println("</body></html>");
}
```

5. Returning Data in JSON or XML Format (for APIs)

- The doGet() method is also used to serve data in JSON or XML format, which can be consumed by client-side applications or other services.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("application/json");
    PrintWriter out = response.getWriter();
    String jsonResponse = "{\"name\": \"Yogesh\", \"age\": 30}";
    out.print(jsonResponse);
    out.flush();
}
```

Advantages of Using doGet():

- **Caching:** Responses can be cached, which makes repeated requests faster.
- **Bookmarkable URLs:** Users can bookmark GET requests and share URLs easily.
- **Safe and Idempotent:** GET is safe for retrieving data without side effects, so it's retry-safe.

When to Use doGet():

- When retrieving or displaying data without modifying any server state.
- When implementing search functionalities or read-only APIs.
- When working with query parameters for data filtering or sorting.

Understanding doPost(): The doPost() method in a servlet handles HTTP POST requests. POST requests are typically used to send data to the server for processing. Unlike GET requests, data sent through POST is not visible in the URL and can handle larger amounts of data, making it suitable for operations that involve sensitive or extensive input, such as login forms, file uploads, and database updates.

When to Use doPost()

- When sending sensitive or large amounts of data.
- When modifying server state (e.g., inserting or updating records).
- When handling form submissions that shouldn't be retried automatically.
- When uploading files or handling data that doesn't fit in the URL parameters.

Here are some common use cases for doPost() in servlets:

1. Processing Form Data (e.g., User Registration, Login)

- doPost() is ideal for handling form data, especially for actions that require sensitive information (like passwords) or that change server state (like submitting a registration).
- For example, a login form might use doPost() to securely send the username and password to the server.

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

    // Retrieve form data
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    // Process login (simplified for illustration)
    if ("admin".equals(username) && "12345".equals(password)) {
        out.println("<h1>Welcome, " + username + "!</h1>");
    } else {
        out.println("<h1>Invalid login credentials</h1>");
    }
}
```

2. Handling File Uploads

- When uploading files, doPost() is required because POST requests support the multipart/form-data encoding type necessary for file upload.
- Example:

```
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.http.Part;
@MultipartConfig
public class FileUploadServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // Retrieving the uploaded file part
        Part filePart = request.getPart("file");
        String fileName = filePart.getSubmittedFileName();
        // Process the file upload
        filePart.write("uploads/" + fileName); // Saves file to server
        out.println("<h2>File uploaded successfully: " + fileName + "</h2>");
    }
}
```

3. Inserting Data into a Database

- doPost() is often used when inserting data into a database, such as saving new user information or submitting feedback forms.
- For example, handling form data and saving it to a database:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String name = request.getParameter("name");
    String email = request.getParameter("email");
    // Database connection and insert logic (simplified)
    Database.saveUser(name, email); // Hypothetical method
    response.getWriter().println("<h1>Thank you, " + name + ", for registering!</h1>");
}
```

4. Updating Data (e.g., Profile Updates)

- doPost() is used for updating existing records, such as modifying user profile information.
- Example:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String userId = request.getParameter("userId");
    String newEmail = request.getParameter("newEmail");
    // Logic to update user email in database
    Database.updateUserEmail(userId, newEmail);
    response.getWriter().println("<h1>Email updated successfully!</h1>");
}
```

5. Processing Sensitive Transactions (e.g., Payment Processing)

- Since POST requests are not stored in browser history and parameters are not visible in the URL, they are suitable for handling sensitive transactions like payment data.

- Example:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String cardNumber = request.getParameter("cardNumber");
    String amount = request.getParameter("amount");
    // Payment processing logic (hypothetical)
    PaymentService.processPayment(cardNumber, amount);
    response.getWriter().println("<h1>Payment Successful!</h1>");
}
```

Advantages of Using doPost()

Advantage	Description
Larger Data Capacity	Supports larger amounts of data, suitable for forms and file uploads.
Security	Data isn't exposed in the URL, making it safer for sensitive information like passwords or payments.
Server State Modification	Suitable for actions that modify server state (e.g., adding, updating, or deleting records).
Non-Idempotent	Suitable for operations that should not be retried automatically by the browser, like transactions.

Program to add two numbers using Servlet

This servlet application:

- Displays a form with two input fields for numbers
- When submitted, processes the numbers in the servlet
- Calculates their sum
- Displays the result in a formatted HTML page
- Provides a link to try another addition

HTML File:

```
<!DOCTYPE html>
<html>
<head>
    <title>Add Two Numbers</title>
</head>
<body>
    <h2>Add Two Numbers</h2>
    <form action="addNumbers" method="post">
        Number 1: <input type="number" name="num1" required><br><br>
        Number 2: <input type="number" name="num2" required><br><br>
        <input type="submit" value="Add Numbers">
    </form>
</body>
</html>
```

Servlet:

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/addNumbers")
public class AddNumbersServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        int num1 = Integer.parseInt(request.getParameter("num1"));
        int num2 = Integer.parseInt(request.getParameter("num2"));
        int sum = num1 + num2;

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Addition Result</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>Addition Result</h2>");
        out.println("<p>" + num1 + " + " + num2 + " = " + sum + "</p>");
        out.println("<br><a href='add-numbers.html'>Try Another Addition</a>");
        out.println("</body>");
        out.println("</html>");
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // Redirect to the HTML form if someone tries to access the servlet directly
        response.sendRedirect("add-numbers.html");
    }
}
```



Create servlet to demonstrate Password Fields, Radio Buttons, Checkboxes, Drop-down (Select) Lists, Hidden Fields, Text Area and Date Input.

This is a web-based registration form application using Java Servlets that demonstrates various HTML form input types and their handling on the server side.

Components Breakdown:

HTML Form (registration-form.html): The form includes multiple input types:

- Hidden Field: Invisible to user, stores form ID
- Text Input: For username
- Password Field: For secure password entry
- Radio Buttons: For gender selection (male/female/other)
- Checkboxes: For multiple interests selection
- Dropdown List: For country selection
- Text Area: For "About Me" longer text input
- Date Input: For date of birth

```
<!DOCTYPE html>
<html>
<head>
  <title>User Registration Form</title>
</head>
<body>
  <h2>User Registration Form</h2>
  <form action="processForm" method="post">
    <!-- Hidden Field -->
    <input type="hidden" name="formId" value="registration_123">

    <!-- Text and Password Fields -->
    <label>Username:</label>
    <input type="text" name="username" required><br><br>

    <label>Password:</label>
    <input type="password" name="password" required><br><br>

    <!-- Radio Buttons -->
    <label>Gender:</label><br>
    <input type="radio" name="gender" value="male" required> Male
    <input type="radio" name="gender" value="female"> Female
    <input type="radio" name="gender" value="other"> Other<br><br>

    <!-- Checkboxes -->
    <label>Interests:</label><br>
    <input type="checkbox" name="interests" value="sports"> Sports
    <input type="checkbox" name="interests" value="music"> Music
    <input type="checkbox" name="interests" value="reading"> Reading
    <input type="checkbox" name="interests" value="travel"> Travel<br><br>

    <!-- Dropdown List -->
    <label>Country:</label>
    <select name="country">
      <option value="">Select a country</option>
      <option value="USA">United States</option>
      <option value="UK">United Kingdom</option>
      <option value="Canada">Canada</option>
      <option value="Australia">Australia</option>
      <option value="India">India</option>
    </select><br><br>

    <!-- Text Area -->
    <label>About Me:</label><br>
    <textarea name="aboutMe" rows="4" cols="50"></textarea><br><br>

    <!-- Date Input -->
    <label>Date of Birth:</label>
    <input type="date" name="dob" required><br><br>

    <input type="submit" value="Register">
  </form>
</body>
</html>
```



```
package arc.nov.examples;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/processForm")
public class RegistrationServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Get all form parameters
        String formId = request.getParameter("formId");
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String gender = request.getParameter("gender");
        String[] interests = request.getParameterValues("interests");
        String country = request.getParameter("country");
        String aboutMe = request.getParameter("aboutMe");
        String dob = request.getParameter("dob");

        // Set response content type
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Generate response HTML
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Registration Details</title>");
        out.println("<style>");
        out.println("table { border-collapse: collapse; width: 60%; margin: 20px 0; }");
        out.println("th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }");
        out.println("th { background-color: #f2f2f2; }");
        out.println("</style>");
        out.println("</head>");
        out.println("<body>");

        out.println("<h2>Registration Details</h2>");
        out.println("<table>");

        // Hidden Field
        out.println("<tr><th>Form ID</th><td>" + formId + "</td></tr>");

        // Username
        out.println("<tr><th>Username</th><td>" + username + "</td></tr>");
```

```
// Password (showing masked)
out.println("<tr><th>Password</th><td>*****</td></tr>");

// Gender
out.println("<tr><th>Gender</th><td>" + gender + "</td></tr>");

// Interests
out.println("<tr><th>Interests</th><td>");
if (interests != null) {
    out.println(String.join(" ", interests));
} else {
    out.println("None selected");
}
out.println("</td></tr>");

// Country
out.println("<tr><th>Country</th><td>" + (country.isEmpty() ? "Not selected" : country) + "</td></tr>");

// About Me
out.println("<tr><th>About Me</th><td>" + (aboutMe.isEmpty() ? "Not provided" : aboutMe) +
"</td></tr>");

// Date of Birth
out.println("<tr><th>Date of Birth</th><td>" + dob + "</td></tr>");

out.println("</table>");

// Add a back button
out.println("<br><a href='index.html'>Back to Registration Form</a>");

out.println("</body>");
out.println("</html>");
}

protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.sendRedirect("index.html");
}
}
```

The screenshot displays a web application with two panels. The left panel, titled "User Registration Form", contains input fields for Username (Archer Infotech), Password (masked with asterisks), Gender (Male selected), Interests (Sports, Music, Travel selected), Country (India), About Me (Archer infotech Pune, We make IT Easy), and Date of Birth (30-12-2008). A "Register" button is at the bottom. The right panel, titled "Registration Details", shows a table with the following data:

Registration Details	
Form ID	registration_123
Username	Archer Infotech
Password	*****
Gender	male
Interests	sports, music, travel
Country	India
About Me	Archer infotech Pune, We make IT Easy
Date of Birth	2008-12-30

A blue arrow points from the "Register" button in the form to the "Registration Details" table. A "Back to Registration Form" link is located below the table.

Deployment Descriptor (web.xml) and Servlet Annotations:

Deployment Descriptor (web.xml): The web.xml file is a configuration file in a Java EE web application that acts as a deployment descriptor. It resides in the WEB-INF directory of the application and provides a way to configure servlets, filters, listeners, and other web components.

Purpose of web.xml:

- **Servlet Configuration:** Define servlets and map them to specific URLs.
- **Filter Configuration:** Set filters for processing requests and responses.
- **Listener Registration:** Specify lifecycle event listeners.
- **Error Handling:** Define custom error pages for specific HTTP error codes.
- **Security Settings:** Configure authentication, authorization, and access constraints.
- **Initialization Parameters:** Pass initialization parameters to servlets or the application.

Here's a basic structure:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="3.1">
```

```
<!-- Servlet Definition -->
```

```
<servlet>
```

```
  <servlet-name>HelloServlet</servlet-name>
```

```
  <servlet-class>com.example.HelloServlet</servlet-class>
```

```
</servlet>
```

```
<!-- Servlet Mapping -->
```

```
<servlet-mapping>
```

```
  <servlet-name>HelloServlet</servlet-name>
```

```
  <url-pattern>/hello</url-pattern>
```

```
</servlet-mapping>
```

```
<!-- Initialization Parameter -->
```

```
<context-param>
```

```
  <param-name>configParam</param-name>
```

```
  <param-value>configValue</param-value>
```

```
</context-param>
```

```
<!-- Error Page -->
```

```
<error-page>
```

```
  <error-code>404</error-code>
```

```
  <location>/error.html</location>
```

```
</error-page>
```

```
<!-- Security Constraint -->
```

```
<security-constraint>
```

```
  <web-resource-collection>
```

```
    <web-resource-name>SecureArea</web-resource-name>
```

```
    <url-pattern>/secure/*</url-pattern>
```

```
  </web-resource-collection>
```

```
  <auth-constraint>
```

```
    <role-name>admin</role-name>
```

```
  </auth-constraint>
```

```
</security-constraint>
```

```
</web-app>
```

The provided web.xml file is a well-structured deployment descriptor for a Java web application. Here's a breakdown of its components:

1. **Servlet Definition and Mapping:**

- A servlet named HelloServlet is defined, pointing to the Java class com.example.HelloServlet.
- It is mapped to the URL pattern /hello, meaning requests to /hello will be handled by this servlet.

2. **Initialization Parameter:**

- A context-wide initialization parameter configParam is set with the value configValue. This is useful for global configuration accessible to all servlets.

3. **Error Page:**

- If a 404 error (Page Not Found) occurs, users will be redirected to /error.html.

4. **Security Constraint:**

- The URL pattern /secure/* is restricted.
- Only users with the role admin can access resources under this pattern.

Servlet Annotations:

Annotations are metadata tags added directly to Java code to configure components without using web.xml. Starting from Servlet 3.0 (Java EE 6), annotations provide an easier and more readable alternative.

Common Annotations:

@WebServlet: The **@WebServlet** annotation is a part of the Java Servlet API, specifically introduced in the Servlet 3.0 specification. It is used to declare a class as a servlet and configure its mapping to specific URLs, replacing the need for XML-based configuration in the web.xml deployment descriptor.

The **@WebServlet** annotation is placed above a class definition to declare that the class is a servlet. The basic syntax is:

```
@WebServlet("/urlPattern")  
public class MyServlet extends HttpServlet {  
    // Servlet methods  
}
```

Attributes: The **@WebServlet** annotation supports several attributes that allow you to configure various aspects of the servlet. Here are the key attributes:

name: Specifies the name of the servlet. This is optional and is used to refer to the servlet in other parts of the application.

```
@WebServlet(name = "myServlet", urlPatterns = "/urlPattern")
```

urlPatterns: Specifies the URL patterns that the servlet will handle. This is a mandatory attribute if value is not used.

```
@WebServlet(urlPatterns = {"/urlPattern1", "/urlPattern2"})
```

value: A shorthand for **urlPatterns**. You can use this attribute to specify a single URL pattern.

```
@WebServlet("/urlPattern")
```

loadOnStartup: Specifies the load-on-startup order of the servlet. A positive integer indicates the load order, with lower values indicating earlier loading. A negative integer indicates that the servlet should be loaded on demand.

@WebServlet(urlPatterns = "/urlPattern", loadOnStartup = 1)

initParams: Specifies initialization parameters for the servlet. These parameters can be accessed using the **ServletConfig** object.

```
@WebServlet(urlPatterns = "/urlPattern", initParams = {
    @WebInitParam(name = "param1", value = "value1"),
    @WebInitParam(name = "param2", value = "value2")
})
```

asyncSupported: Indicates whether the servlet supports asynchronous processing.

@WebServlet(urlPatterns = "/urlPattern", asyncSupported = true)

description: Provides a description of the servlet. This is optional and is used for documentation purposes.

@WebServlet(urlPatterns = "/urlPattern", description = "This is my servlet")

displayName: Provides a display name for the servlet. This is optional and is used for documentation purposes.

@WebServlet(urlPatterns = "/urlPattern", displayName = "My Servlet")

Example:

```
@WebServlet(
    name = "ConfigServlet",
    urlPatterns = {"/config"},
    initParams = {
        @WebInitParam(name = "appName", value = "MyApp"),
        @WebInitParam(name = "version", value = "1.0")
    }
)
public class ConfigServlet extends HttpServlet {
    @Override
    public void init() throws ServletException {
        String appName = getServletConfig().getInitParameter("appName");
        String version = getServletConfig().getInitParameter("version");
        System.out.println("App Name: " + appName + ", Version: " + version);
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.getWriter().write("Configuration loaded.");
    }
}
```

// Employee Registration Application

// index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Employee Registration</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f4;
      margin: 20px;
      padding: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      min-height: 100vh;
    }
    .container {
      width: 50%;
      background: #fff;
      padding: 20px;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      margin: 20px;
    }
    h2 {
      text-align: center;
      color: #333;
      margin-bottom: 20px;
    }
    fieldset {
      border: 2px solid #ddd;
      padding: 15px;
      border-radius: 8px;
      margin-bottom: 20px;
    }
    legend {
      font-weight: bold;
      color: #555;
    }
    label {
```

```
        display: block;
        font-weight: bold;
        margin-top: 10px;
    }
    input, select {
        width: 100%;
        padding: 8px;
        margin-top: 5px;
        margin-bottom: 10px;
        border: 1px solid #ccc;
        border-radius: 5px;
        font-size: 14px;
    }
    button {
        width: 100%;
        padding: 10px;
        background-color: #28a745;
        border: none;
        color: white;
        font-size: 16px;
        cursor: pointer;
        border-radius: 5px;
        margin-top: 10px;
    }
    button:hover {
        background-color: #218838;
    }
</style>
</head>
<body>
<div class="container">
    <h2>Employee Registration Form</h2>
    <form action="EmployeeServlet" method="post">
        <fieldset>
            <legend>Personal Details</legend>
            <label for="name">Full Name:</label>
            <input type="text" id="name" name="name" required>

            <label for="dob">Date of Birth:</label>
            <input type="date" id="dob" name="dob" required>

            <label for="gender">Gender:</label>
            <select id="gender" name="gender" required>
                <option value="Male">Male</option>
```



```
<option value="Female">Female</option>
<option value="Other">Other</option>
</select>

<label for="email">Email:</label>
<input type="email" id="email" name="email" required>

<label for="phone">Phone Number:</label>
<input type="tel" id="phone" name="phone" required>
</fieldset>

<fieldset>
  <legend>Educational Details</legend>
  <label for="qualification">Highest Qualification:</label>
  <input type="text" id="qualification" name="qualification" required>

  <label for="university">University/Institute:</label>
  <input type="text" id="university" name="university" required>

  <label for="passingYear">Year of Passing:</label>
  <input type="number" id="passingYear" name="passingYear" required>
</fieldset>

<fieldset>
  <legend>Employment Details</legend>
  <label for="company">Company Name:</label>
  <input type="text" id="company" name="company">

  <label for="designation">Designation:</label>
  <input type="text" id="designation" name="designation">

  <label for="experience">Years of Experience:</label>
  <input type="number" id="experience" name="experience" min="0">
</fieldset>

<button type="submit">Register</button>
</form>
</div>
</body>
</html>
```

// Registration servlet

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/EmployeeServlet")
public class EmployeeServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Retrieve form data
        String name = request.getParameter("name");
        String dob = request.getParameter("dob");
        String gender = request.getParameter("gender");
        String email = request.getParameter("email");
        String phone = request.getParameter("phone");
        String qualification = request.getParameter("qualification");
        String university = request.getParameter("university");
        String passingYear = request.getParameter("passingYear");
        String company = request.getParameter("company");
        String designation = request.getParameter("designation");
        String experience = request.getParameter("experience");

        // HTML response
        out.println("<html><head><title>Registration Successful</title>");
        out.println("<style>");
        out.println("body { font-family: Arial, sans-serif; background-color: #f4f4f4; text-align:");
        out.println("center; padding: 50px; }");
        out.println(".container { background: #fff; padding: 20px; border-radius: 10px; display:");
        out.println("inline-block; box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); }");
        out.println("h2 { color: #28a745; }");
        out.println("</style>");
        out.println("</head><body>");
        out.println("<div class='container'>");
        out.println("<h2>Registration Successful</h2>");
        out.println("<p>Name: " + name + "</p>");
```

```

        out.println("<p>Date of Birth: " + dob + "</p>");
        out.println("<p>Gender: " + gender + "</p>");
        out.println("<p>Email: " + email + "</p>");
        out.println("<p>Phone: " + phone + "</p>");
        out.println("<p>Qualification: " + qualification + "</p>");
        out.println("<p>University: " + university + "</p>");
        out.println("<p>Passing Year: " + passingYear + "</p>");
        out.println("<p>Company: " + (company.isEmpty() ? "N/A" : company) + "</p>");
        out.println("<p>Designation: " + (designation.isEmpty() ? "N/A" : designation) + "</p>");
        out.println("<p>Experience: " + (experience.isEmpty() ? "0" : experience) + " years</p>");
        out.println("</div>");
        out.println("</body></html>");
    }
}

```

```
//=====
```

```
// CRUD for above (Only Insert)
```

```
-- Create database
```

```
CREATE DATABASE EmployeeDB;
```

```
-- Use the database
```

```
USE EmployeeDB;
```

```
-- Create employees table
```

```
CREATE TABLE employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    dob DATE NOT NULL,
    gender ENUM('Male', 'Female', 'Other') NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15) NOT NULL,
    qualification VARCHAR(100) NOT NULL,
    university VARCHAR(150) NOT NULL,
    passingYear INT NOT NULL,
    company VARCHAR(150),
    designation VARCHAR(100),
    experience INT DEFAULT 0
);
```

```
mysql> desc employees;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment

name	varchar(100)	NO		NULL		
dob	date	NO		NULL		
gender	enum('Male','Female','Other')	NO		NULL		
email	varchar(100)	NO	UNI	NULL		
phone	varchar(15)	NO		NULL		
qualification	varchar(100)	NO		NULL		
university	varchar(150)	NO		NULL		
passingYear	int	NO		NULL		
company	varchar(150)	YES		NULL		
designation	varchar(100)	YES		NULL		
experience	int	YES		0		
+-----+-----+-----+-----+-----+						

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Employee Registration</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f4;
      margin: 20px;
      padding: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      min-height: 100vh;
    }
    .container {
      width: 50%;
      background: #fff;
      padding: 20px;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      margin: 20px;
    }
    h2 {
      text-align: center;
      color: #333;
      margin-bottom: 20px;
    }
```

```
}
fieldset {
    border: 2px solid #ddd;
    padding: 15px;
    border-radius: 8px;
    margin-bottom: 20px;
}
legend {
    font-weight: bold;
    color: #555;
}
label {
    display: block;
    font-weight: bold;
    margin-top: 10px;
}
input, select {
    width: 100%;
    padding: 8px;
    margin-top: 5px;
    margin-bottom: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 14px;
}
button {
    width: 100%;
    padding: 10px;
    background-color: #28a745;
    border: none;
    color: white;
    font-size: 16px;
    cursor: pointer;
    border-radius: 5px;
    margin-top: 10px;
}
button:hover {
    background-color: #218838;
}
</style>
</head>
<body>
    <div class="container">
        <h2>Employee Registration Form</h2>
```

```
<form action="EmployeeServlet" method="post">
  <fieldset>
    <legend>Personal Details</legend>
    <label for="name">Full Name:</label>
    <input type="text" id="name" name="name" required>

    <label for="dob">Date of Birth:</label>
    <input type="date" id="dob" name="dob" required>

    <label for="gender">Gender:</label>
    <select id="gender" name="gender" required>
      <option value="Male">Male</option>
      <option value="Female">Female</option>
      <option value="Other">Other</option>
    </select>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>

    <label for="phone">Phone Number:</label>
    <input type="tel" id="phone" name="phone" required>
  </fieldset>

  <fieldset>
    <legend>Educational Details</legend>
    <label for="qualification">Highest Qualification:</label>
    <input type="text" id="qualification" name="qualification" required>

    <label for="university">University/Institute:</label>
    <input type="text" id="university" name="university" required>

    <label for="passingYear">Year of Passing:</label>
    <input type="number" id="passingYear" name="passingYear" required>
  </fieldset>

  <fieldset>
    <legend>Employment Details</legend>
    <label for="company">Company Name:</label>
    <input type="text" id="company" name="company">

    <label for="designation">Designation:</label>
    <input type="text" id="designation" name="designation">

    <label for="experience">Years of Experience:</label>
```

```
<input type="number" id="experience" name="experience" min="0">
</fieldset>

<button type="submit">Register</button>
</form>
</div>
</body>
</html>

//-----

// EmployeeServlet.java

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/EmployeeServlet")
public class EmployeeServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Retrieve form data
        String name = request.getParameter("name");
        String dob = request.getParameter("dob");
        String gender = request.getParameter("gender");
        String email = request.getParameter("email");
        String phone = request.getParameter("phone");
        String qualification = request.getParameter("qualification");
        String university = request.getParameter("university");
        String passingYear = request.getParameter("passingYear");
        String company = request.getParameter("company");
        String designation = request.getParameter("designation");
        String experience = request.getParameter("experience");
```



```
// Save data to database
EmployeeDAO employeeDAO = new EmployeeDAO();
boolean isSaved = employeeDAO.saveEmployee(name, dob, gender, email, phone,
qualification, university, passingYear, company, designation, experience);

// HTML response
out.println("<html><head><title>Registration Status</title>");
out.println("<style>");
out.println("body { font-family: Arial, sans-serif; background-color: #f4f4f4; text-align:
center; padding: 50px; }");
out.println(".container { background: #fff; padding: 20px; border-radius: 10px; display:
inline-block; box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); }");
out.println("h2 { color: #28a745; }");
out.println("</style>");
out.println("</head><body>");
out.println("<div class='container'>");
if (isSaved) {
    out.println("<h2>Registration Successful</h2>");
} else {
    out.println("<h2>Registration Failed</h2>");
}
out.println("</div>");
out.println("</body></html>");
}
}
//-----
```

EmployeeDAO.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class EmployeeDAO {
    private static final String URL = "jdbc:mysql://localhost:3306/EmployeeDB";
    private static final String USER = "root";
    private static final String PASSWORD = "Archer@12345";

    public boolean saveEmployee(String name, String dob, String gender, String email, String
phone, String qualification, String university, String passingYear, String company, String
designation, String experience) {
        boolean status = false;
```

```
String query = "INSERT INTO employees (name, dob, gender, email, phone, qualification, university, passingYear, company, designation, experience) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
```

```
try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);  
    PreparedStatement stmt = conn.prepareStatement(query)) {
```

```
    stmt.setString(1, name);  
    stmt.setString(2, dob);  
    stmt.setString(3, gender);  
    stmt.setString(4, email);  
    stmt.setString(5, phone);  
    stmt.setString(6, qualification);  
    stmt.setString(7, university);  
    stmt.setString(8, passingYear);  
    stmt.setString(9, company);  
    stmt.setString(10, designation);  
    stmt.setString(11, experience);
```

```
    int rowsInserted = stmt.executeUpdate();  
    if (rowsInserted > 0) {  
        status = true;  
    }  
} catch (SQLException e) {  
    e.printStackTrace();  
}  
return status;  
}  
}
```

```
//=====
```

// CRUD for above (Entire)

```
// index.html
```

```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="UTF-8">  
    <title>Employee Registration</title>  
</head>  
<body>  
    <h2>Employee Registration Form</h2>
```

```
<form action="EmployeeServlet" method="post">
  <input type="hidden" name="action" value="register">
  <label for="name">Name:</label><br>
  <input type="text" id="name" name="name" required><br><br>

  <label for="dob">Date of Birth:</label><br>
  <input type="date" id="dob" name="dob" required><br><br>

  <label for="gender">Gender:</label><br>
  <select id="gender" name="gender" required>
    <option value="Male">Male</option>
    <option value="Female">Female</option>
    <option value="Other">Other</option>
  </select><br><br>

  <label for="email">Email:</label><br>
  <input type="email" id="email" name="email" required><br><br>

  <label for="phone">Phone:</label><br>
  <input type="text" id="phone" name="phone" required><br><br>

  <label for="qualification">Qualification:</label><br>
  <input type="text" id="qualification" name="qualification" required><br><br>

  <label for="university">University:</label><br>
  <input type="text" id="university" name="university" required><br><br>

  <label for="passingYear">Passing Year:</label><br>
  <input type="number" id="passingYear" name="passingYear" required><br><br>

  <label for="company">Company:</label><br>
  <input type="text" id="company" name="company"><br><br>

  <label for="designation">Designation:</label><br>
  <input type="text" id="designation" name="designation"><br><br>

  <label for="experience">Experience (years):</label><br>
  <input type="number" id="experience" name="experience" value="0"><br><br>

  <input type="submit" value="Register">
</form>
<hr>
<!-- Employee list will be shown by the servlet on successful registration or after CRUD
operations -->
```

</body>

</html>

//-----

EmployeeServlet.java

import java.io.IOException;

import java.io.PrintWriter;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

// Map the servlet to /EmployeeServlet

@WebServlet("/EmployeeServlet")

public class EmployeeServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

private EmployeeDAO employeeDAO;

public void init() {

employeeDAO = new EmployeeDAO();

}

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Decide action based on parameter. If not specified, show list.

String action = request.getParameter("action");

if ("delete".equals(action)) {

int id = Integer.parseInt(request.getParameter("id"));

employeeDAO.deleteEmployee(id);

response.sendRedirect("EmployeeServlet");

} else if ("edit".equals(action)) {

int id = Integer.parseInt(request.getParameter("id"));

Employee emp = employeeDAO.getEmployeeById(id);

showEditForm(response, emp);

} else {

// Default: display all employees

listEmployees(response);

}

}

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// Determine if it is a registration or update
String action = request.getParameter("action");
if ("update".equals(action)) {
// Update an existing employee
Employee emp = new Employee();
emp.setId(Integer.parseInt(request.getParameter("id")));
emp.setName(request.getParameter("name"));
emp.setDob(request.getParameter("dob"));
emp.setGender(request.getParameter("gender"));
emp.setEmail(request.getParameter("email"));
emp.setPhone(request.getParameter("phone"));
emp.setQualification(request.getParameter("qualification"));
emp.setUniversity(request.getParameter("university"));
emp.setPassingYear(Integer.parseInt(request.getParameter("passingYear")));
emp.setCompany(request.getParameter("company"));
emp.setDesignation(request.getParameter("designation"));
emp.setExperience(Integer.parseInt(request.getParameter("experience")));

employeeDAO.updateEmployee(emp);
response.sendRedirect("EmployeeServlet");
} else {
// Registration of a new employee
Employee emp = new Employee();
emp.setName(request.getParameter("name"));
emp.setDob(request.getParameter("dob"));
emp.setGender(request.getParameter("gender"));
emp.setEmail(request.getParameter("email"));
emp.setPhone(request.getParameter("phone"));
emp.setQualification(request.getParameter("qualification"));
emp.setUniversity(request.getParameter("university"));
emp.setPassingYear(Integer.parseInt(request.getParameter("passingYear")));
emp.setCompany(request.getParameter("company"));
emp.setDesignation(request.getParameter("designation"));
emp.setExperience(Integer.parseInt(request.getParameter("experience")));

employeeDAO.insertEmployee(emp);
response.sendRedirect("EmployeeServlet");
}
}

// Method to display all employees in an HTML table
private void listEmployees(HttpServletResponse response) throws IOException {
```

```
List<Employee> list = employeeDAO.getAllEmployees();
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html><head><title>Employee List</title></head><body>");
out.println("<h2>Employee List</h2>");
out.println("<table border='1' cellpadding='5'>");
out.println("<tr>" + "<th>ID</th>" + "<th>Name</th>" + "<th>DOB</th>" +
"<th>Gender</th>" + "<th>Email</th>"
+ "<th>Phone</th>" + "<th>Qualification</th>" + "<th>University</th>" + "<th>Passing
Year</th>"
+ "<th>Company</th>" + "<th>Designation</th>" + "<th>Experience</th>" +
"<th>Actions</th>" + "</tr>");
```

```
for (Employee emp : list) {
out.println("<tr>");
out.println("<td>" + emp.getId() + "</td>");
out.println("<td>" + emp.getName() + "</td>");
out.println("<td>" + emp.getDob() + "</td>");
out.println("<td>" + emp.getGender() + "</td>");
out.println("<td>" + emp.getEmail() + "</td>");
out.println("<td>" + emp.getPhone() + "</td>");
out.println("<td>" + emp.getQualification() + "</td>");
out.println("<td>" + emp.getUniversity() + "</td>");
out.println("<td>" + emp.getPassingYear() + "</td>");
out.println("<td>" + emp.getCompany() + "</td>");
out.println("<td>" + emp.getDesignation() + "</td>");
out.println("<td>" + emp.getExperience() + "</td>");
out.println("<td>" + "<a href='EmployeeServlet?action=edit&id=" + emp.getId() +
"'>Edit</a> | "
+ "<a href='EmployeeServlet?action=delete&id=" + emp.getId()
+ "' onclick='return confirm(\"Are you sure?\");'>Delete</a>" + "</td>");
out.println("</tr>");
}
out.println("</table>");
out.println("<br><a href='index.html'>Register New Employee</a>");
out.println("</body></html>");
}
```

```
// Method to show the edit form pre-filled with employee data
private void showEditForm(HttpServletResponse response, Employee emp) throws
IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html><head><title>Edit Employee</title></head><body>");
```

```

out.println("<h2>Edit Employee</h2>");
out.println("<form action='EmployeeServlet' method='post'>");
out.println("<input type='hidden' name='action' value='update'>");
out.println("<input type='hidden' name='id' value='" + emp.getId() + "'>");
out.println("Name: <input type='text' name='name' value='" + emp.getName() + "'");
out.println("required><br><br>");
out.println("Date of Birth: <input type='date' name='dob' value='" + emp.getDob() + "'");
out.println("required><br><br>");
out.println("Gender: <select name='gender' required>");
out.println("<option value='Male'" + ("Male".equals(emp.getGender()) ? " selected" : "") +");
out.println(">Male</option>");
out.println("<option value='Female'" + ("Female".equals(emp.getGender()) ? " selected" : "") +");
out.println(">Female</option>");
out.println("<option value='Other'" + ("Other".equals(emp.getGender()) ? " selected" : "") +");
out.println(">Other</option>");
out.println("</select><br><br>");
out.println("Email: <input type='email' name='email' value='" + emp.getEmail() + "'");
out.println("required><br><br>");
out.println("Phone: <input type='text' name='phone' value='" + emp.getPhone() + "'");
out.println("required><br><br>");
out.println("Qualification: <input type='text' name='qualification' value='" +");
out.println("emp.getQualification()");
out.println("+ "' required><br><br>");
out.println("University: <input type='text' name='university' value='" + emp.getUniversity()");
out.println("+ "' required><br><br>");
out.println("Passing Year: <input type='number' name='passingYear' value='" +");
out.println("emp.getPassingYear()");
out.println("+ "' required><br><br>");
out.println("Company: <input type='text' name='company' value='" + emp.getCompany() +");
out.println("><br><br>");
out.println("Designation: <input type='text' name='designation' value='" +");
out.println("emp.getDesignation() + "'><br><br>");
out.println("Experience: <input type='number' name='experience' value='" +");
out.println("emp.getExperience() + "'><br><br>");
out.println("<input type='submit' value='Update'>");
out.println("</form>");
out.println("<br><a href='EmployeeServlet'>Back to Employee List</a>");
out.println("</body></html>");
}
}
///-----

```

```
// EmployeeDAO.java
```



```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class EmployeeDAO {

    // Update these connection details to match your database configuration
    private final String jdbcURL = "jdbc:mysql://localhost:3306/EmployeeDB";
    private final String jdbcUsername = "root";
    private final String jdbcPassword = "Archer@12345";

    public EmployeeDAO() {
        try {
            // Load the MySQL JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    // Utility method to get a DB connection
    private Connection getConnection() throws SQLException {
        return DriverManager.getConnection(jdbcURL, jdbcUsername, jdbcPassword);
    }

    // Insert a new employee record
    public void insertEmployee(Employee emp) {
        String sql = "INSERT INTO employees (name, dob, gender, email, phone, qualification, university, passingYear, company, designation, experience) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
        try (Connection connection = getConnection();
            PreparedStatement ps = connection.prepareStatement(sql)) {
            ps.setString(1, emp.getName());
            ps.setDate(2, Date.valueOf(emp.getDob())); // expecting yyyy-MM-dd
            ps.setString(3, emp.getGender());
            ps.setString(4, emp.getEmail());
            ps.setString(5, emp.getPhone());
            ps.setString(6, emp.getQualification());
            ps.setString(7, emp.getUniversity());
            ps.setInt(8, emp.getPassingYear());
            ps.setString(9, emp.getCompany());
            ps.setString(10, emp.getDesignation());
            ps.setInt(11, emp.getExperience());
        }
    }
}
```

```
        ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

// Update an existing employee record

```
public void updateEmployee(Employee emp) {
    String sql = "UPDATE employees SET name=?, dob=?, gender=?, email=?, phone=?,
    qualification=?, university=?, passingYear=?, company=?, designation=?, experience=?
    WHERE id=?";
    try (Connection connection = getConnection();
        PreparedStatement ps = connection.prepareStatement(sql)) {
        ps.setString(1, emp.getName());
        ps.setDate(2, Date.valueOf(emp.getDob()));
        ps.setString(3, emp.getGender());
        ps.setString(4, emp.getEmail());
        ps.setString(5, emp.getPhone());
        ps.setString(6, emp.getQualification());
        ps.setString(7, emp.getUniversity());
        ps.setInt(8, emp.getPassingYear());
        ps.setString(9, emp.getCompany());
        ps.setString(10, emp.getDesignation());
        ps.setInt(11, emp.getExperience());
        ps.setInt(12, emp.getId());
        ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

// Delete an employee record by id

```
public void deleteEmployee(int id) {
    String sql = "DELETE FROM employees WHERE id=?";
    try (Connection connection = getConnection();
        PreparedStatement ps = connection.prepareStatement(sql)) {
        ps.setInt(1, id);
        ps.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

// Retrieve a single employee by id

```
public Employee getEmployeeById(int id) {
    Employee emp = null;
    String sql = "SELECT * FROM employees WHERE id=?";
    try (Connection connection = getConnection();
        PreparedStatement ps = connection.prepareStatement(sql)) {
        ps.setInt(1, id);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            emp = new Employee();
            emp.setId(rs.getInt("id"));
            emp.setName(rs.getString("name"));
            emp.setDob(rs.getDate("dob").toString());
            emp.setGender(rs.getString("gender"));
            emp.setEmail(rs.getString("email"));
            emp.setPhone(rs.getString("phone"));
            emp.setQualification(rs.getString("qualification"));
            emp.setUniversity(rs.getString("university"));
            emp.setPassingYear(rs.getInt("passingYear"));
            emp.setCompany(rs.getString("company"));
            emp.setDesignation(rs.getString("designation"));
            emp.setExperience(rs.getInt("experience"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return emp;
}
```

```
// Retrieve all employee records
public List<Employee> getAllEmployees() {
    List<Employee> list = new ArrayList<>();
    String sql = "SELECT * FROM employees";
    try (Connection connection = getConnection();
        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {
        while (rs.next()) {
            Employee emp = new Employee();
            emp.setId(rs.getInt("id"));
            emp.setName(rs.getString("name"));
            emp.setDob(rs.getDate("dob").toString());
            emp.setGender(rs.getString("gender"));
            emp.setEmail(rs.getString("email"));
            emp.setPhone(rs.getString("phone"));
            emp.setQualification(rs.getString("qualification"));
        }
    }
}
```

```
        emp.setUniversity(rs.getString("university"));
        emp.setPassingYear(rs.getInt("passingYear"));
        emp.setCompany(rs.getString("company"));
        emp.setDesignation(rs.getString("designation"));
        emp.setExperience(rs.getInt("experience"));
        list.add(emp);
    }
} catch (SQLException e) {
    e.printStackTrace();
}
return list;
}
}
```

//-----

// Employee.java

```
public class Employee {
    private int id;
    private String name;
    private String dob;
    private String gender;
    private String email;
    private String phone;
    private String qualification;
    private String university;
    private int passingYear;
    private String company;
    private String designation;
    private int experience;

    // Getters and setters

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
```

```
        this.name = name;
    }
    public String getDob() {
        return dob;
    }
    public void setDob(String dob) {
        this.dob = dob;
    }
    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public String getQualification() {
        return qualification;
    }
    public void setQualification(String qualification) {
        this.qualification = qualification;
    }
    public String getUniversity() {
        return university;
    }
    public void setUniversity(String university) {
        this.university = university;
    }
    public int getPassingYear() {
        return passingYear;
    }
    public void setPassingYear(int passingYear) {
        this.passingYear = passingYear;
    }
}
```

```
public String getCompany() {  
    return company;  
}  
public void setCompany(String company) {  
    this.company = company;  
}  
public String getDesignation() {  
    return designation;  
}  
public void setDesignation(String designation) {  
    this.designation = designation;  
}  
public int getExperience() {  
    return experience;  
}  
public void setExperience(int experience) {  
    this.experience = experience;  
}  
}  
  
//-----
```

// Using Servlet to fetch data from External API Data:

What is External API Data?

External API Data refers to the data retrieved from an external (third-party) API over the internet. An API (Application Programming Interface) allows different applications to communicate and exchange data in a structured format, typically JSON or XML.

When is External API Data Used?

External API data is used when an application needs real-time or regularly updated information from another system without manually storing or managing it.

Common Use Cases:

1. Weather Information – Fetching weather details from APIs like OpenWeatherMap.
2. Financial Data – Stock market prices, currency exchange rates from APIs like Alpha Vantage.
3. Social Media Integration – Fetching user posts, likes, and comments from Facebook, Twitter APIs.
4. Maps & Location Services – Google Maps API for getting locations and routes.
5. Authentication – Using OAuth APIs like Google or Facebook for login.
6. E-commerce – Retrieving product details, prices, or reviews from marketplaces like Amazon.
7. Machine Learning & AI – Using APIs like OpenAI (ChatGPT), Google Vision, etc., for text/image processing.

How to Use External API Data?

To fetch data from an external API, a client (like a Java servlet, frontend app, or another system) makes an HTTP request (GET, POST, etc.) and receives a response.

Basic Steps to Use an External API:

1. Find the API Endpoint – Identify the URL that provides the required data (e.g., <https://api.example.com/data>).
2. Send an HTTP Request – Use methods like GET (to retrieve) or POST (to send data).
3. Receive API Response – The response is typically in JSON format.
4. Parse and Process Data – Convert JSON/XML data into usable Java objects.
5. Display or Store Data – Use it in a UI (JSP, HTML) or save it in a database.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.json.JSONArray;
import org.json.JSONObject;

@WebServlet("/fetchData")
public class ApiService extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String apiUrl = "https://jsonplaceholder.typicode.com/users";
        String jsonResponse = fetchApiData(apiUrl);

        response.setContentType("text/html");
        response.getWriter().println(generateHtmlTable(jsonResponse));
    }

    private String fetchApiData(String apiUrl) throws IOException {
        URL url = new URL(apiUrl);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Accept", "application/json");
```



```
if (conn.getResponseCode() != 200) {
    throw new RuntimeException("Failed: HTTP error code " + conn.getResponseCode());
}

BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
StringBuilder response = new StringBuilder();
String output;
while ((output = br.readLine()) != null) {
    response.append(output);
}
br.close();
conn.disconnect();

return response.toString();
}

private String generateHtmlTable(String jsonData) {
    StringBuilder html = new StringBuilder();
    html.append("<html><head><title>API Data</title>");
    html.append("<style>table{width:50%;border-collapse:collapse;}th,td{border:1px solid black;padding:8px;text-align:left;}th{background-color:#f2f2f2;}</style>");
    html.append("</head><body>");
    html.append("<h2>Fetched Data from External API</h2>");
    html.append("<table>");

    html.append("<tr><th>ID</th><th>Name</th><th>Email</th><th>Phone</th><th>Company</th></tr>");

    JSONArray jsonArray = new JSONArray(jsonData);
    for (int i = 0; i < jsonArray.length(); i++) {
        JSONObject obj = jsonArray.getJSONObject(i);
        JSONObject company = obj.getJSONObject("company");

        html.append("<tr>");
        html.append("<td>").append(obj.getInt("id")).append("</td>");
        html.append("<td>").append(obj.getString("name")).append("</td>");
        html.append("<td>").append(obj.getString("email")).append("</td>");
        html.append("<td>").append(obj.getString("phone")).append("</td>");
        html.append("<td>").append(company.getString("name")).append("</td>");
        html.append("</tr>");
    }

    html.append("</table></body></html>");
    return html.toString();
}
}
```

Example: Integrating the web templet with Servlet application:

(Download templet from: <https://www.codingnepalweb.com/create-website-login-registration-form-html/>)

What is a Session?

A session is a way to store and maintain user-specific data across multiple requests in a web application. Since HTTP is stateless, a session helps retain user information (like login details, cart items, or preferences) between different page requests.

What is HttpSession?

HttpSession is an interface in Java Servlet API that allows web applications to create, manage, and track user sessions. It stores session-specific data on the server side and assigns a unique Session ID to the client.

How HttpSession Works

1. User makes a request to the server.
2. Server creates a session using `request.getSession()`.
3. Session data is stored on the server, and a JSESSIONID cookie is sent to the client.
4. Client sends back JSESSIONID in subsequent requests, allowing the server to identify the user.
5. Session ends when the user logs out or the session times out.

Basic Methods in HttpSession

Method	Description
<code>getSession()</code>	Returns the current session or creates a new one.
<code>setAttribute(String name, Object value)</code>	Stores an attribute in the session.
<code>getAttribute(String name)</code>	Retrieves an attribute from the session.
<code>removeAttribute(String name)</code>	Removes an attribute from the session.
<code>invalidate()</code>	Destroys the session and removes all attributes.
<code>getId()</code>	Returns the unique session ID.
<code>getCreationTime()</code>	Returns the time when the session was created.
<code>getLastAccessedTime()</code>	Returns the last time the session was accessed.

What is a Cookie?

A **cookie** is a small piece of data stored on the client's browser by the server. It helps in identifying users and maintaining their session across multiple requests.

Types of Cookies:

1. **Session Cookies** – Temporary, deleted when the browser is closed.
2. **Persistent Cookies** – Stored for a specific duration, even after the browser is closed.
3. **Secure Cookies** – Transmitted only over HTTPS for security.
4. **HttpOnly Cookies** – Not accessible via JavaScript (prevents XSS attacks).

Index. html

```
<!DOCTYPE html>
<!-- Coding By CodingNepal - www.codingnepalweb.com -->
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Online Feedback System</title>
<link rel="stylesheet"
```

```

        href="https://fonts.googleapis.com/css2?family=Material+Symbols+Rounded:opsz,wght,FILL,GRAD@48,400,0,0">
<link rel="stylesheet" href="css/style.css">
<script src="js/script.js" defer></script>
</head>
<body>
    <header>
        <nav class="navbar">
            <span class="hamburger-btn material-symbols-rounded">menu</span> <a
                href="#" class="logo"> 
                <h2>Online Feedback System</h2>
            </a>
            <ul class="links">
                <span class="close-btn material-symbols-rounded">close</span>
                <li><a href="#">Home</a></li>
                <li><a href="#">Portfolio</a></li>
                <li><a href="#">Courses</a></li>
                <li><a href="#">About us</a></li>
                <li><a href="#">Contact us</a></li> -->
            </ul>
            <button class="login-btn">LOG IN</button>
        </nav>
        <div class="justified-text">
            <p>
                <strong> Project Overview: Online Feedback System </strong> </br> The
                "Online Feedback System" is a web-based application designed to
                streamline feedback collection and analysis. The system starts with
                an introduction page that provides an overview and prompts users to
                submit feedback or admins to log in. Users can register with their
                name, email, and password, log in, and answer feedback questions set
                by the admin. Admins can log in to add, update, or delete questions,
                assign subjects and marks to questions, view feedback results, and
                calculate an overall rating based on user responses. The application
                leverages HTML and CSS for the frontend, Java Servlets for backend
                logic, and MySQL with JDBC for persistent data storage.
            </p>
        </div>
    </header>

    <div class="blur-bg-overlay"></div>
    <div class="form-popup">
        <span class="close-btn material-symbols-rounded">close</span>
        <div class="form-box login">
            <div class="form-details">
                <h2>Welcome Back</h2>
                <p>Please log in using your personal information to stay
                    connected with us.</p>
            </div>
            <div class="form-content">
                <h2>LOGIN</h2>
                <form action="LoginServlet" method="post">
                    <div class="input-field">
                        <input type="text" required name="email"> <label>Email</label>
                    </div>
                    <div class="input-field">
                        <input type="password" required name="password">
                        <label>Password</label>
                    </div>
                    <a href="#" class="forgot-pass-link">Forgot password?</a>
                    <button type="submit">Log In</button>
                </form>
                <div class="bottom-link">
                    Don't have an account? <a href="#" id="signup-link">Signup</a>
                </div>
            </div>
        </div>
    </div>
    <div class="form-box signup">

```

```

        <div class="form-details">
            <h2>Create Account</h2>
            <p>To become a part of our community, please sign up using your
                personal information.</p>
        </div>
        <div class="form-content">
            <h2>SIGNUP</h2>
            <form action="SignupServlet" method="post">
                <div class="input-field">
                    <input type="text" required name="name"> <label>Name</label>
                </div>
                <div class="input-field">
                    <input type="text" required name="email"> <label>Enter your
email</label>
                </div>
                <div class="input-field">
                    <input type="password" required name="password"> <label>Create
password</label>
                </div>
                <div class="policy-text">
                    <input type="checkbox" id="policy"> <label for="policy">
                        I agree the <a href="#" class="option">Terms &
Conditions</a>
                    </label>
                </div>
                <div>
                    <button type="submit">Sign Up</button>
                </div>
            </form>
            <div class="bottom-link">
                Already have an account? <a href="#" id="login-link">Login</a>
            </div>
        </div>
    </div>
</body>
</html>

```

```

// Take the CSS and JS from downloaded folder
// -----
// Login Servlet

```

```
package controller;
```

```
import java.io.*;
import java.sql.SQLException;
```

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

```
import dao.UserDAO;
import model.User;
```

```
@WebServlet("/LoginServlet")
```

```
public class LoginServlet extends HttpServlet {
    private UserDAO userDAO = new UserDAO();
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String email = request.getParameter("email");
        String password = request.getParameter("password");
        System.out.println(email+" - "+password);
```

```
        try {
            User user = userDAO.login(email, password);
            if (user != null) {
                HttpSession session = request.getSession();
```

```
        session.setAttribute("name", user.getName());
        session.setAttribute("role", user.getRole());
        response.sendRedirect("dashboard");
    } else {
        response.getWriter().write("Invalid credentials");
    }
} catch (SQLException e) {
    e.printStackTrace();
    response.getWriter().write("Login Failed: " + e.getMessage());
}
}
}

//-----

// UserDAO. Java

package dao;

import java.sql.*;

import model.User;

public class UserDAO {
    private static final String URL = "jdbc:mysql://localhost:3306/feedback_db";
    private static final String USER = "root";
    private static final String PASSWORD = "Archer@12345";

    private Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

    public void signup(User user) throws SQLException {
        String sql = "INSERT INTO users(name, email, password) VALUES(?, ?, ?)";
        try (Connection con = getConnection();
            PreparedStatement ps = con.prepareStatement(sql)) {
            ps.setString(1, user.getName());
            ps.setString(2, user.getEmail());
            ps.setString(3, user.getPassword());
            ps.executeUpdate();
        }
    }

    public User login(String email, String password) throws SQLException {
        String sql = "SELECT * FROM users WHERE email=? AND password=?";
        try (Connection con = getConnection();
            PreparedStatement ps = con.prepareStatement(sql)) {
            ps.setString(1, email);
            ps.setString(2, password);
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                User user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setEmail(rs.getString("email"));
                user.setPassword(rs.getString("password"));
                user.setRole(rs.getString("role"));
                return user;
            }
            return null;
        }
    }

    public void createAdmin(User user) throws SQLException {
        String sql = "INSERT INTO users(name, email, password, role) VALUES(?, ?, ?, 'admin')";
        try (Connection con = getConnection();
```

```
        PreparedStatement ps = con.prepareStatement(sql) {
            ps.setString(1, user.getName());
            ps.setString(2, user.getEmail());
            ps.setString(3, user.getPassword());
            ps.executeUpdate();
        }
    }
}
```

//-----

// User.java

package model;

```
public class User {
    private int id;
    private String name;
    private String email;
    private String password;
    private String role;

    // Constructors
    public User() {}
    public User(String name, String email, String password, String role) {
        this.name = name;
        this.email = email;
        this.password = password;
        this.role = role;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }
}
```

//-----

//Dashboard.Servlet.java

package controller;

```
import java.io.IOException;
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
```

@WebServlet("/dashboard")

```
public class DashboardServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        HttpSession session = request.getSession();
        String name = (String)session.getAttribute("name");
    }
}
```

Java Programming – Java Servlet API

```
String role = (String)session.getAttribute("role");

response.setContentType("text/html");
PrintWriter out = response.getWriter();
System.out.println(name+" - "+role);

out.println("<html><body>");
out.println("<h2>Welcome, " + name + "!</h2>");

if("admin".equals(role)) {
    out.println("<h3>Admin Dashboard</h3>");
    out.println("<a href='createAdmin.html'>Create New Admin</a><br>");
    out.println("<a href='questions.html'>Fill Questions</a><br>");
    out.println("<a href='viewFeedback'>View Feedbacks</a>");
} else {
    out.println("<h3>User Dashboard</h3>");
    out.println("<a href='feedback.html'>Fill Feedback</a>");
}
out.println("<br><a href='logout'>Logout</a>");
out.println("</body></html>");
}
}

//-----

// SignUpServlet.java

package controller;

import java.io.*;
import dao.UserDAO;
import java.sql.SQLException;

import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

import model.User;

@WebServlet("/SignUpServlet")
public class SignUpServlet extends HttpServlet {

    private UserDAO userDAO = new UserDAO();

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String name = request.getParameter("name");
        String email = request.getParameter("email");
        String password = request.getParameter("password");

        User user = new User(name, email, password, "user");
        try {
            userDAO.signup(user);
            response.sendRedirect("index.html");
        } catch (SQLException e) {
            e.printStackTrace();
            response.getWriter().write("Signup Failed: " + e.getMessage());
        }
    }
}
```


Database table query:

```
+-----+  
| users | CREATE TABLE `users` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `name` varchar(100) NOT NULL,  
  `email` varchar(100) NOT NULL,  
  `password` varchar(100) NOT NULL,  
  `role` enum('user','admin') DEFAULT 'user',  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `email` (`email`)  
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |  
+-----+
```

//-----



JavaServer Pages (JSP):

Meaning of Dynamic Web Content:

Dynamic web content refers to web pages or elements that are generated or modified in real-time based on user interactions, database queries, or server-side logic. Unlike static content (which remains unchanged and is the same for every user), dynamic content is tailored and personalized for each user or context.

Here are some examples

- **Login Pages:** Displaying user-specific data after authentication.
- **E-commerce Sites:** Showing personalized recommendations or cart contents.
- **Social Media:** Updating news feeds or showing real-time interactions.
- **Search Engines:** Dynamically generating search results based on user queries.

Dynamic content is created on the server side using logic (e.g., querying a database, processing user input) and then sent to the client (browser) for rendering.

web developers primarily relied on Common Gateway Interface (CGI) and technologies like Servlets for creating dynamic web content. While effective, these approaches had several limitations.

- The CGI having some performance issues, It is more complex and less portable.
- Servlets required developers to embed HTML directly within Java code, which made the code difficult to read, maintain, and debug. This approach blended business logic with the presentation layer. and having Scalability Challenges while Managing large-scale applications. Also Web designers and Java developers often worked on the same files, creating conflicts during development.

These limitations highlighted the need for a more efficient, maintainable, and scalable solution for dynamic web content generation. This paved the way for JavaServer Pages (JSP).

Introduction of JSP

JSP was introduced by Sun Microsystems to simplify the development of web-based applications. It allowed developers to create dynamic web pages by embedding Java code within HTML using special tags, thus separating the presentation layer from the business logic.

History of JSP

- **1999:** JSP 1.0 was released as part of the Java EE platform by Sun Microsystems to address the complexity of servlets and improve the web application development process.
- **2001:** JSP 1.1 and JSP 1.2 introduced features like XML-compliant syntax and better integration with JavaBeans and tag libraries.
- **2003:** JSP 2.0 was a significant release, introducing the Expression Language (EL), custom tag

libraries, and enhancements for simplifying page development.

- **2006:** JSP 2.1 became part of Java EE 5, with better support for JavaServer Faces (JSF) and improved integration with other Java EE technologies.

- **2013 onwards:** JSP 2.3, part of Java EE 7, introduced features like enhanced security, better annotations, and improved integration with Servlet 3.1.

Advantages of JSP Over Existing Technology

- **Separation of Concerns:** Clear distinction between presentation (HTML) and business logic (Java).

- **Ease of Use:** Simplified syntax and integration with JavaBeans and tag libraries.

- **Scalability:** JSPs compile into servlets, retaining the performance benefits of servlets while providing better maintainability.

- **Reusable Components:** Use of custom tags and JavaBeans enhances code reusability.

Thus, JSP solved many of the issues faced with earlier technologies, offering a more modern, scalable, and developer-friendly approach to web development.

Role of JSP in Real-Time Application Development Today:

JavaServer Pages (JSP) remains a foundational technology for building dynamic web applications. While newer frameworks and technologies like Spring Boot, Angular, React, and Vue.js have emerged, JSP continues to have relevance in specific use cases, especially in applications that rely heavily on the Java ecosystem.

While JSP is still relevant, it has been partially replaced in many modern web development contexts due to some limitations:

- **Modern Frontend Frameworks:** Tools like React, Angular, and Vue.js are preferred for dynamic and interactive user interfaces due to their rich capabilities and separation from backend technologies.

- **Server-Side Rendering Alternatives:** Technologies like Thymeleaf, Freemarker, or even JavaScript server-side frameworks (e.g., Next.js) are often preferred for templating in Java-based projects.

- **Complexity in Large Applications:** In complex web apps with a lot of user interaction, using JSP's traditional method of mixing logic with views can get messy and hard to manage.

While JSP is no longer the default choice for modern, large-scale, or highly interactive web applications, it continues to play a vital role in Java-based ecosystems.

JSP Life Cycle

The JSP life cycle defines the process by which a JSP page is created, executed, and destroyed within a web server. This cycle involves several stages, transforming a JSP page into a servlet, executing it, and finally cleaning up resources. Understanding this lifecycle is crucial for optimizing the performance of JSP-based applications.

Stages in the JSP Life Cycle

Translation Phase

- The JSP engine translates the JSP file into a corresponding servlet file.
- This phase occurs only once unless the JSP file is modified.

Compilation Phase

- The generated servlet file is compiled into a .class file.

Loading and Initialization

- The servlet container loads the .class file and creates an instance of the servlet.
- The `jspInit()` method is called for initialization (e.g., setting up resources).

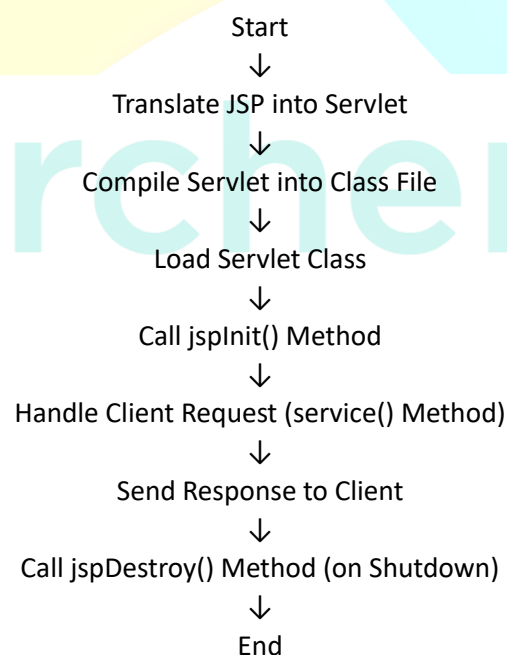
Request Processing

- For every client request, the `service()` method is executed.
- The servlet container processes the request and generates a response (HTML, XML, etc.).

Destruction

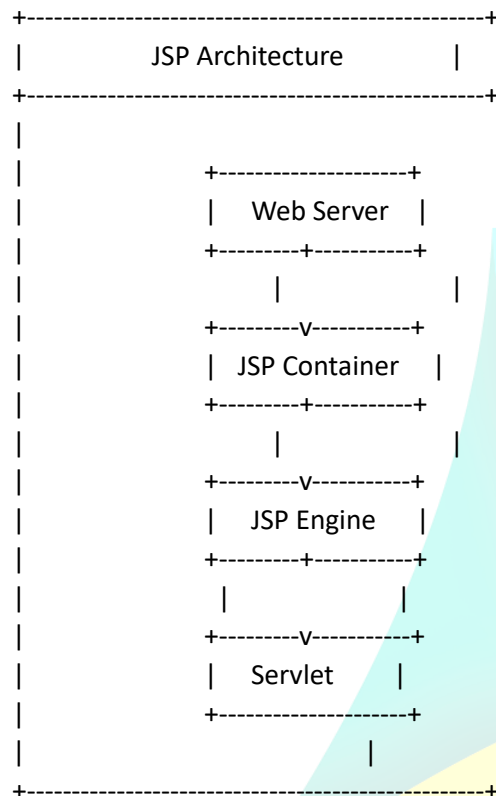
- When the JSP is no longer needed or the application shuts down, the `jspDestroy()` method is called.
- This phase releases resources used by the JSP.

Flowchart for JSP Life Cycle



JSP Architecture

JavaServer Pages (JSP) is a server-side technology that enables the creation of dynamic web content. Understanding the architecture of JSP is crucial for developing efficient and robust web applications. The JSP architecture is designed to separate the presentation layer from the business logic, making it easier to manage and maintain web applications. Here's a detailed explanation of the JSP architecture:



Here's a step-by-step flow of how a JSP page is processed in the JSP architecture:

- Client Request:

- * A client (e.g., a web browser) sends an HTTP request to the web server.

- Web Server:

- * The web server receives the request and forwards it to the JSP container.

- JSP Container:

- * The JSP container receives the request and determines if the requested JSP page exists.
- * If the JSP page exists, the JSP container checks if the servlet corresponding to the JSP page is already loaded.

- JSP Engine:

- * If the servlet is not loaded, the JSP engine translates the JSP page into a servlet source file.
- * The JSP engine compiles the servlet source file into bytecode.
- * The JSP engine loads the servlet class into the JVM.
- * The JSP engine creates an instance of the servlet class.

- Servlet Lifecycle:

- * The JSP container calls the `jspInit()` method to initialize the servlet instance.
- * The JSP container calls the `_jspService()` method to handle the request.
- * The `_jspService()` method generates the dynamic content and sends the response to the client.
- * The JSP container calls the `jspDestroy()` method to clean up resources before the servlet instance is destroyed.

- Response:

- * The JSP container sends the generated response back to the client through the web server.

//-----

JSP Tags:

JavaServer Pages (JSP) provides a variety of tags that are used to embed Java code within HTML pages, enabling the creation of dynamic web content. These tags are divided into different groups based on their functionality.

1. JSP Directives Tags: JSP directives provide global information about the JSP page. They are used to set page-level instructions and configurations.

- **@page:** Defines page-dependent attributes, such as scripting language, error page, and buffer size.
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

- **@include:** Includes a file during the translation phase.
<%@ include file="header.jsp" %>

- **@taglib:** Declares a tag library that is used in the JSP page.
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

2. JSP Scripting Elements: JSP scripting elements allow you to embed Java code within the JSP page. They are used to perform dynamic operations and generate content.

* **<% %>:** Scriptlets are used to embed Java code that is executed during the request processing phase.
<%
String message = "Hello, World!";
out.println(message);
%>

* **<%= %>:** Expressions are used to output the value of a Java expression directly into the page.
<%= message %>

* **<%! %>:** Declarations are used to declare variables and methods that are available throughout the JSP page.
<%! String message = "Hello, World!";%>

3. JSP Actions: JSP actions are used to control the behavior of the JSP engine. They are predefined tags that perform specific tasks.

- * **<jsp:useBean>:** Declares a JavaBean and makes it available for use in the JSP page.
`<jsp:useBean id="user" class="com.example.User" scope="session" />`
- * **<jsp:setProperty>:** Sets the properties of a JavaBean.
`<jsp:setProperty name="user" property="name" value="John Doe" />`
- * **<jsp:getProperty>:** Gets the properties of a JavaBean.
`<jsp:getProperty name="user" property="name" />`
- * **<jsp:include>:** Includes a file during the request processing phase.
`<jsp:include page="footer.jsp" />`
- * **<jsp:forward>:** Forwards the request to another resource.
`<jsp:forward page="nextPage.jsp" />`
- * **<jsp:param>:** Passes parameters to the included or forwarded page.
`<jsp:include page="footer.jsp">`
`<jsp:param name="paramName" value="paramValue" />`
`</jsp:include>`
- * **<jsp:plugin>:** Embeds a Java applet or JavaBean in the JSP page.
`<jsp:plugin type="applet" code="MyApplet.class" codebase="." width="300" height="300">`
`<jsp:params>`
`<jsp:param name="paramName" value="paramValue" />`
`</jsp:params>`
`</jsp:plugin>`
- * **<jsp:fallback>:** Provides alternative content if the browser does not support the <jsp:plugin> tag.
`<jsp:plugin type="applet" code="MyApplet.class" codebase="." width="300" height="300">`
`<jsp:fallback>`
`<p>Your browser does not support Java applets.</p>`
`</jsp:fallback>`
`</jsp:plugin>`

4. JSP Comments: JSP comments are used to add comments to the JSP page. They are not included in the generated HTML output.

* **<%-- --%>:** JSP comments are used to add comments that are not included in the generated HTML output.

`<%-- This is a JSP comment --%>`

`//=====`

// Using JSP Directives Tags.

1. @page Directive: The @page directive is used to define page-dependent attributes, such as scripting language, error page, and buffer size.

// Example: Setting Content Type and Character Encoding

//index-page.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
  <title>Page Directive Example</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <p>This page uses the @page directive to set the content type and character encoding.</p>
</body>
</html>
```

//-----

// Example: Setting an Error Page

//index-error.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"
  errorPage="error.jsp" %>
<!DOCTYPE html>
<html>
<head>
  <title>Error Page Directive Example</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <p>This page uses the @page directive to set an error page.</p>
  <%
    // Simulate an error
    int result = 10 / 0;
  %>
</body>
</html>
```

//error.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Insert title here</title>
</head>
<body>
  <h1> This is Error Page</h1>
</body>
</html>
```

///-----

// Example: Setting Buffer Size

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" buffer="8kb"
autoFlush="true" %>
<!DOCTYPE html>
<html>
<head>
  <title>Buffer Size Directive Example</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <p>This page uses the @page directive to set the buffer size and auto-flush behavior.</p>
</body>
</html>
//-----
```

2. @include Directive: The @include directive is used to include a file during the translation phase. This is useful for including common headers, footers, or other reusable components.

// Example: Including a Header File

// header.jsp:

```
<!DOCTYPE html>
<html>
<head>
  <title>Included Header</title>
</head>
<body>
  <h1>Welcome to My Website</h1>
</body>
</html>
```

// main.jsp:

```
<%@ include file="header.jsp" %>
<!DOCTYPE html>
<html>
<head>
  <title>Include Directive Example</title>
</head>
<body>
  <h2>This is the main content.</h2>
  <p>The header is included using the @include directive.</p>
</body>
</html>
```

//-----

3. @taglib Directive: The @taglib directive is used to declare a tag library that is used in the JSP page. This is useful for using custom tags or JSTL (JavaServer Pages Standard Tag Library).

// Example: Using JSTL Core Tags(need to include jstl.jar)

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

```
<!DOCTYPE html>
<html>
<head>
  <title>JSTL Loop Example</title>
</head>
<body>
  <h1>List of Items</h1>

  <!-- Initialize the list using scriptlet -->
  <%
    java.util.List<String> itemList = java.util.Arrays.asList("Item 1", "Item 2", "Item 3", "Item 4");
    request.setAttribute("itemList", itemList);
  %>

  <!-- Iterate over the list and display items -->
  <c:forEach var="item" items="${itemList}">
    <p>${item}</p>
  </c:forEach>
</body>
</html>

//-----
```

// Using JSP Scripting Elements:

JSP scripting elements allow you to embed Java code within the JSP page. They are used to perform dynamic operations and generate content. JSP (JavaServer Pages) scripting elements allow you to embed Java code within HTML pages, enabling dynamic content generation. There are three main types of JSP scripting elements: scriptlets, expressions, and declarations. Each serves a specific purpose in embedding Java code within JSP pages.

1. Scriptlets: <% %>: Scriptlets are used to embed Java code that is executed during the request processing phase. They are enclosed within <% %> tags. The code is placed inside the servlet's service() method.

2. Expressions: <%= %>: Expressions are used to output the value of a Java expression directly into the page. They are enclosed within <%= %> tags.

3. Declarations: <%! %>: Declarations are used to declare variables and methods that are available throughout the JSP page. They are enclosed within <%! %> tags. The declared content becomes part of the servlet's class definition and is shared across multiple requests.

4. Comments: Comments in JSP can be either HTML comments (visible in the browser's source code) or JSP comments (invisible to the client and not processed by the server).

- **HTML Comment:** Visible in the browser's "View Source".

```
<!-- This is an HTML comment -->
```

- **JSP Comment:** Invisible in the browser's "View Source".

```
<%-- This is a JSP comment --%>
```

- | | | |
|---------------|---------------|--|
| - Declaration | <%! ... %> | Declare variables or methods |
| - Expression | <%= ... %> | Output the result of an expression |
| - Scriptlet | <% ... %> | Write Java code inside the JSP |
| - JSP Comment | <%-- ... --%> | Add comments that are not visible to clients |

Example:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <title>JSP Scriptlet, Expression, and Declaration</title>
</head>
<body>
  <h2>Demonstration of JSP Scripting Elements</h2>
  <%-- Declaration: Variable declared here is available throughout the JSP page --%>
  <%!
    String globalMessage = "This is a declared message!";

    // Declaring a method
    public String greetUser(String name) {
      return "Hello, " + name + "!";
    }
  %>

  <%-- Scriptlet: Executing Java code during request processing --%>
  <%
    String localMessage = "Hello from Scriptlet!";
    out.println("<p>Scriptlet Message: " + localMessage + "</p>");
  %>

  <%-- Expression: Directly outputting a Java expression --%>
  <p>Expression Message: <%= globalMessage %></p>

  <%-- Calling a method declared in Declaration block --%>
  <p>Greet User: <%= greetUser("John") %></p>

</body>
</html>
```

//-----

//Example :

```
<%@ page language="java" %>
<%!
  // Declaration: A method to calculate factorial
  public int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
      result *= i;
    }
  }
}
```

```
        return result;
    }
}%>

<%
    // Scriptlet: Initialize a number
    int number = 5;
}%>

<h1>Factorial Calculation</h1>

<!-- Expression: Display the result -->
<p>The factorial of <%= number %> is <%= factorial(number) %>.</p>
```

```
//=====
```

// Reading the database table using the JSP Scripting tags

```
<%@ page import="java.sql.*" %>
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Employee Records</title>
    <style>
        table {
            width: 60%;
            border-collapse: collapse;
            margin: 20px 0;
        }
        th, td {
            border: 1px solid black;
            padding: 10px;
            text-align: left;
        }
        th {
            background-color: #f2f2f2;
        }
    </style>
</head>
<body>
    <h2>Employee List</h2>
    <table>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Salary</th>
        </tr>
    <%
        // Database connection details
        String url = "jdbc:mysql://localhost:3306/empdb"; // Database URL
        String user = "root"; // Change if necessary
```

```

String password = "Archer@12345"; // Change if necessary

Connection con = null;
Statement stmt = null;
ResultSet rs = null;

try {
    // Load MySQL JDBC Driver
    Class.forName("com.mysql.cj.jdbc.Driver");

    // Establish connection
    con = DriverManager.getConnection(url, user, password);

    // Create SQL statement
    stmt = con.createStatement();
    String query = "SELECT * FROM employees";
    rs = stmt.executeQuery(query);

    // Loop through the result set and display in table rows
    while (rs.next()) {
        %>
        <tr>
            <td><%= rs.getInt("id") %></td>
            <td><%= rs.getString("name") %></td>
            <td><%= rs.getDouble("salary") %></td>
        </tr>
        <%
    }
} catch (Exception e) {
    out.println("<p>Error: " + e.getMessage() + "</p>");
} finally {
    // Close resources
    if (rs != null) try { rs.close(); } catch (SQLException ignored) {}
    if (stmt != null) try { stmt.close(); } catch (SQLException ignored) {}
    if (con != null) try { con.close(); } catch (SQLException ignored) {}
}
%>

</table>
</body>
</html>

```

//=====

// Using JSP Actions:

JSP actions are used to control the behavior of the JSP engine. They are predefined tags that perform specific tasks.

* **<jsp:useBean>**: Declares a JavaBean and makes it available for use in the JSP page.

```
<jsp:useBean id="user" class="com.example.User" scope="session" />
```

* **<jsp:setProperty>**: Sets the properties of a JavaBean.

```
<jsp:setProperty name="user" property="name" value="John Doe" />
```

* **<jsp:getProperty>**: Gets the properties of a JavaBean.

```
<jsp:getProperty name="user" property="name" />
```

* **<jsp:include>**: Includes a file during the request processing phase.

```
<jsp:include page="footer.jsp" />
```

* **<jsp:forward>**: Forwards the request to another resource.

```
<jsp:forward page="nextPage.jsp" />
```

* **<jsp:param>**: Passes parameters to the included or forwarded page.

```
<jsp:include page="footer.jsp">
<jsp:param name="paramName" value="paramValue" />
</jsp:include>
```

* **<jsp:plugin>**: Embeds a Java applet or JavaBean in the JSP page.

```
<jsp:plugin type="applet" code="MyApplet.class" codebase="." width="300" height="300">
<jsp:params>
<jsp:param name="paramName" value="paramValue" />
</jsp:params>
</jsp:plugin>
```

* **<jsp:fallback>**: Provides alternative content if the browser does not support the <jsp:plugin> tag.

```
<jsp:plugin type="applet" code="MyApplet.class" codebase="." width="300" height="300">
<jsp:fallback>
<p>Your browser does not support Java applets.</p>
</jsp:fallback>
</jsp:plugin>
```

// CRUD Application using JSP Tages

// Employee. Java

package classes;

import java.io.Serializable;

```
public class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private double salary;

    public Employee() {}
    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
}
```



```
public int getId() { return id; }
public void setId(int id) { this.id = id; }

public String getName() { return name; }
public void setName(String name) { this.name = name; }

public double getSalary() { return salary; }
public void setSalary(double salary) { this.salary = salary; }
}
//-----
```

// EmployeeDAO.java

```
package classes;
import java.sql.*;
import java.util.*;

public class EmployeeDAO {
    private static final String URL = "jdbc:mysql://localhost:3306/empdb";
    private static final String USER = "root";
    private static final String PASSWORD = "Archer@12345";
    static {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static List<Employee> getAllEmployees() throws SQLException {
        List<Employee> list = new ArrayList<>();
        try (Connection con = DriverManager.getConnection(URL, USER, PASSWORD);
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM employees")) {
            while (rs.next()) {
                list.add(new Employee(rs.getInt("id"), rs.getString("name"), rs.getDouble("salary")));
            }
        }
        return list;
    }

    public static void addEmployee(Employee emp) throws SQLException {
        String sql = "INSERT INTO employees (name, salary) VALUES (?, ?)";
        try (Connection con = DriverManager.getConnection(URL, USER, PASSWORD);
            PreparedStatement stmt = con.prepareStatement(sql)) {
            stmt.setString(1, emp.getName());
            stmt.setDouble(2, emp.getSalary());
            stmt.executeUpdate();
        }
    }

    public static void updateEmployee(Employee emp) throws SQLException {
        String sql = "UPDATE employees SET name=?, salary=? WHERE id=?";
        try (Connection con = DriverManager.getConnection(URL, USER, PASSWORD);
            PreparedStatement stmt = con.prepareStatement(sql)) {
            stmt.setString(1, emp.getName());
        }
    }
}
```

```
        stmt.setDouble(2, emp.getSalary());
        stmt.setInt(3, emp.getId());
        stmt.executeUpdate();
    }
}

public static void deleteEmployee(int id) throws SQLException {
    String sql = "DELETE FROM employees WHERE id=?";
    try (Connection con = DriverManager.getConnection(URL, USER, PASSWORD);
        PreparedStatement stmt = con.prepareStatement(sql)) {
        stmt.setInt(1, id);
        stmt.executeUpdate();
    }
}
}
//-----
```

// Index.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Employee Management</title>
</head>
<body>
    <h2>Employee Management System</h2>
    <ul>
        <li><a href="listEmployees.jsp">View Employees</a></li>
        <li><a href="addEmployee.jsp">Add Employee</a></li>
    </ul>
    <!-- <jsp:include page="footer.jsp" /> --%>
</body>
</html>
//-----
```

// addEmployee.jsp

```
<jsp:useBean id="newEmployee" class="classes.Employee" scope="request"/>
<jsp:setProperty name="newEmployee" property="*" />
<%@ page import="classes.*" %>

<form action="addEmployee.jsp" method="post">
    Name: <input type="text" name="name"><br>
    Salary: <input type="text" name="salary"><br>
    <input type="submit" value="Add Employee">
</form>
<%
    if ("POST".equalsIgnoreCase(request.getMethod())) {
        EmployeeDAO.addEmployee(newEmployee);
        response.sendRedirect("listEmployees.jsp");
    }
%>
//-----
```

// updateEmployee.jsp

```
<jsp:useBean id="updateEmployee" class="classes.Employee" scope="request"/>
<jsp:setProperty name="updateEmployee" property="*" />
<%@ page import="classes.*" %>
<%
    int id = Integer.parseInt(request.getParameter("id"));
    for (Employee e : EmployeeDAO.getAllEmployees()) {
        if (e.getId() == id) {
            request.setAttribute("updateEmployee", e);
            break;
        }
    }
%>
<form action="updateEmployee.jsp" method="post">
    <input type="hidden" name="id" value="<jsp:getProperty name='updateEmployee' property='id' />" />
    Name: <input type="text" name="name" value="<jsp:getProperty name='updateEmployee' property='name' />"><br>
    Salary: <input type="text" name="salary" value="<jsp:getProperty name='updateEmployee' property='salary' />"><br>
    <input type="submit" value="Update">
</form>

<%
    if ("POST".equalsIgnoreCase(request.getMethod())) {
        EmployeeDAO.updateEmployee(updateEmployee);
        response.sendRedirect("listEmployees.jsp");
    }
%>
//-----
```

// deleteEmployee.jsp

```
<%@ page import="classes.EmployeeDAO, java.sql.SQLException" %>
<%
    // Get the employee ID from request parameter
    String idParam = request.getParameter("id");
    if (idParam != null && !idParam.isEmpty()) {
        try {
            int id = Integer.parseInt(idParam);
            // Delete employee record
            EmployeeDAO.deleteEmployee(id);
            // Redirect to the employee list page
            response.sendRedirect("listEmployees.jsp");
        } catch (NumberFormatException e) {
            out.println("<p style='color:red;'>Invalid Employee ID format.</p>");
        } catch (SQLException e) {
            out.println("<p style='color:red;'>Error deleting employee: " + e.getMessage() + "</p>");
        }
    } else {
        out.println("<p style='color:red;'>Employee ID is required for deletion.</p>");
    }
%>
//-----
```

```
// listEmployee.jsp
```

```
<%@ page import="java.util. *, java.sql. *" %>
```

```
<%@ page import="classes. *" %>
```

```
<jsp:useBean id="employees" class="java.util.ArrayList" scope="request" />
```

```
<jsp:setProperty name="employees" property="*" />
```

```
<%
```

```
List<Employee> employeeList = EmployeeDAO.getAllEmployees();
```

```
request.setAttribute("employees", employeeList);
```

```
%>
```

```
<h2>Employee List</h2>
```

```
<table border="1">
```

```
<tr>
```

```
    <th>ID</th>
```

```
    <th>Name</th>
```

```
    <th>Salary</th>
```

```
    <th>Actions</th>
```

```
</tr>
```

```
<%
```

```
for (Employee e : employeeList) {
```

```
%>
```

```
    <tr>
```

```
        <td><%=e.getId()%></td>
```

```
        <td><%=e.getName()%></td>
```

```
        <td><%=e.getSalary()%></td>
```

```
        <td><a href="updateEmployee.jsp?id=<%=e.getId()%>">Edit</a> | <a
```

```
href="deleteEmployee.jsp?id=<%=e.getId()%>" onclick="return confirm('Are you sure you want to delete this employee?');">Delete </a></td>
```

```
    </tr>
```

```
<%
```

```
}
```

```
%>
```

```
</table>
```

```
//-----
```

Demonstrating Session Management with <jsp:useBean> in JSP

To demonstrate **session management** using <jsp:useBean>, we will store an **Employee** object in the session scope. This allows the bean to persist across multiple requests during a user's session.

```
// index.jsp
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
```

```
    pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

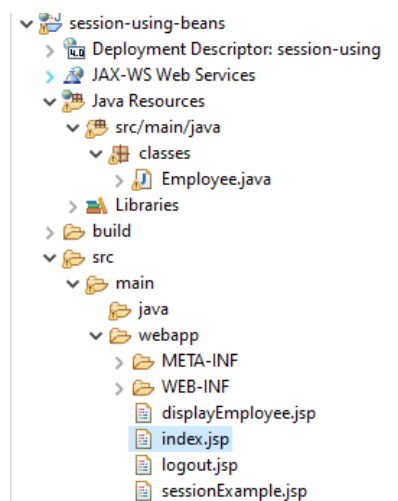
```
<meta charset="UTF-8">
```

```
<title>Insert title here</title>
```

```
</head>
```

```
<body>
```

```
    <form action="sessionExample.jsp" method="post">
```



```
Name: <input type="text" name="name"><br>
Age: <input type="text" name="age"><br>
<input type="submit" value="Submit">

</form>
</body>
</html>
//-----

// sessionExample.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<jsp:useBean id="emp" class="classes.Employee" scope="session"/>
<jsp:setProperty name="emp" property="*" />

<!DOCTYPE html>
<html>
<head>
<title>Session Example</title>
</head>
<body>
<h2>Employee Details Stored in Session</h2>
Name: <%= emp.getName() %><br>
Age: <%= emp.getAge() %><br>

<a href="displayEmployee.jsp">Go to Display Page</a>
</body>
</html>
//-----
displayEmployee.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<jsp:useBean id="emp" class="classes.Employee" scope="session"/>

<!DOCTYPE html>
<html>
<head>
<title>Display Employee</title>
</head>
<body>
<h2>Retrieved Employee Details from Session</h2>
Name: <%= emp.getName() %><br>
Age: <%= emp.getAge() %><br>

<a href="logout.jsp">Logout</a>
</body>
</html>
//-----
Logout.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<jsp:useBean id="emp" class="classes.Employee" scope="session"/>
<%
    session.invalidate(); // Destroy session
%>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Logout</title>
</head>
<body>
    <h2><%= emp.getName() %> Session Cleared</h2>
    <a href="index.jsp">Go Back to Form</a>
</body>
</html>
//-----
Employee.java
package classes;
import java.io.Serializable;

public class Employee implements Serializable {
    private String name;
    private int age;

    public Employee() {} // Required no-arg constructor

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
}

//-----
```

JSP Implicit Objects

JSP provides several **implicit objects** that are automatically available in JSP pages without requiring explicit declaration. These objects simplify web development by allowing easy access to request data, response handling, session management, and application configurations.

JSP Implicit Objects:

1. **Request Object (request)**
 - Represents HttpServletRequest.
 - Used to retrieve request parameters, headers, attributes, and session details.
2. **Response Object (response)**
 - Represents HttpServletResponse.
 - Used to send responses to the client, including setting headers, cookies, and redirections.
3. **Out Object (out)**
 - Represents JspWriter.
 - Used for sending output to the client (browser), supporting buffering.
4. **Session Object (session)**
 - Represents HttpSession.
 - Used to manage user sessions, store and retrieve session attributes.
5. **Application Object (application)**
 - Represents ServletContext.
 - Used for sharing data across the entire application, accessing global parameters.
6. **Config Object (config)**
 - Represents ServletConfig.

- Used to fetch configuration parameters defined for the servlet.
- 7. **Exception Object (exception)**
 - Represents Throwable.
 - Used in JSP error pages to handle exceptions.

Each of these objects plays a crucial role in handling web requests efficiently in JSP-based applications.

//=====

Expression Language (EL)

Expression Language (EL) is used in JSP to simplify the process of accessing **JavaBeans properties, request attributes, session attributes, and more** without writing Java code inside JSP.

Why Use EL?

- Eliminates the need for `<%= ... %>` scriptlets.
- Provides a **more readable** and **cleaner** JSP page.
- Can **access objects** from different scopes (request, session, application, etc.).
- Supports **arithmetic, relational, and logical operations**.

Example: Without EL (Traditional JSP)

```
<jsp:useBean id="emp" class="classes.Employee" scope="session"/>
Employee Name: <%= emp.getName() %>
```

Example: Using EL

```
Employee Name: ${emp.name}
```

//-----

EL Operators

- **Arithmetic Operators** (+, -, *, /, %)
- **Relational Operators** (==, !=, <, >, <=, >=)
- **Logical Operators** (&&, ||, !)
- **Empty Operator** (empty to check if a value is null or empty)
- **Conditional Operator** (? : for ternary operations)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Insert title here</title>
    </head>
    <body>
        <p>Sum: ${10 + 5}</p>
        <p>Is 10 greater than 5? ${10 > 5}</p>
        <p>Logical AND (true && false): ${true && false}</p>
        <p>Is variable 'name' empty? ${empty name}</p>
        <p>Result: ${10 > 5} ? 'Yes' : 'No'</p>
    </body>
</html>
```


Implicit Objects in EL

- **param** → Access request parameters (param.name)
- **paramValues** → Retrieve multiple values of a request parameter
- **header** → Access request headers (header.User-Agent)
- **headerValues** → Retrieve multiple values from headers
- **cookie** → Access cookies (cookie.username.value)
- **initParam** → Access context initialization parameters
- **pageScope, requestScope, sessionScope, applicationScope** → Access attributes stored in different scopes

Moderate-level JSP CRUD (Create, Read, Update, Delete) application demonstrating the use of EL implicit objects:

Overview

- **Create:** Add a new user via form submission.
- **Read:** Display the list of users.
- **Update:** Update an existing user's name.
- **Delete:** Remove a user from the list.
- **Demonstrates EL Implicit Objects:**
 - param: Access form data.
 - paramValues: Retrieve multiple values.
 - header & headerValues: Access request headers.
 - cookie: Access stored cookies.
 - initParam: Access context parameters.
 - pageScope, requestScope, sessionScope, applicationScope: Store and retrieve data.

index.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page import="java.util.List, java.util.ArrayList" %>

<!-- Retrieve or initialize users list in session scope -->
<%
    List<String> users = (List<String>) session.getAttribute("users");
    if (users == null) {
        users = new ArrayList<>();
        session.setAttribute("users", users);
    }
%>
<!--
<c:if test="${empty sessionScope.users}">
    <c:set var="users" value="${pageContext.request.setAttribute('users', [])}" scope="session"/>
</c:if> --%>

<html>
<head>
    <title>${initParam.appName}</title>
```

```
</head>
<body>
  <h2>Welcome to ${initParam.appName}</h2>

  <!-- Display User-Agent from headers --%>
  <p>Your browser: ${header["User-Agent"]}</p>

  <!-- Form to Add Users --%>
  <h3>Add User</h3>
  <form action="process.jsp" method="post">
    Name: <input type="text" name="name">
    <input type="submit" name="action" value="Add">
  </form>

  <!-- Display Users List --%>
  <h3>User List</h3>
  <ul>
    <c:forEach var="user" items="${sessionScope.users}">
      <li>${user}
        <a href="process.jsp?action=Delete&name=${user}">Delete</a>
      </li>
    </c:forEach>
  </ul>

  <!-- Form to Update User --%>
  <h3>Update User</h3>
  <form action="process.jsp" method="post">
    Old Name: <input type="text" name="oldName">
    New Name: <input type="text" name="newName">
    <input type="submit" name="action" value="Update">
  </form>

  <!-- Retrieve and display cookies --%>
  <h3>Cookies:</h3>
  <p>Username stored in cookie: ${cookie.username.value}</p>
</body>
</html>
//-----
```

// process.jsp

```
<%@ page import="java.util.List" %>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%
  List<String> users = (List<String>) session.getAttribute("users");
  if (users == null) {
    response.sendRedirect("index.jsp");
  }
%>
```

```
        return;
    }

    String action = request.getParameter("action");

    if ("Add".equals(action)) {
        String name = request.getParameter("name");
        if (name != null && !name.trim().isEmpty()) {
            users.add(name);
        }
    } else if ("Delete".equals(action)) {
        String name = request.getParameter("name");
        users.remove(name);
    } else if ("Update".equals(action)) {
        String oldName = request.getParameter("oldName");
        String newName = request.getParameter("newName");
        if (users.contains(oldName) && newName != null && !newName.trim().isEmpty()) {
            users.set(users.indexOf(oldName), newName);
        }
    }
}

session.setAttribute("users", users);

// Store username in cookie
if (request.getParameter("name") != null) {
    javax.servlet.http.Cookie cookie = new javax.servlet.http.Cookie("username",
request.getParameter("name"));
    response.addCookie(cookie);
}

response.sendRedirect("index.jsp");
%>
//-----

//web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <context-param>
        <param-name>appName</param-name>
        <param-value>EL CRUD Demo</param-value>
    </context-param>
</web-app>
```

Accessing JavaBeans & Collections Using EL:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.example.User"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page import="java.util.Map, java.util.HashMap"%>
    <%@ page import="java.util.List, java.util.ArrayList"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%
        User user = new User("Pankaj", 20);
        session.setAttribute("user", user);
    %>

    <%
        List<String> names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");
        session.setAttribute("names", names);
    %>

    <%
        Map<String, String> countries = new HashMap<>();
        countries.put("US", "United States");
        countries.put("IN", "India");
        countries.put("UK", "United Kingdom");
        session.setAttribute("countries", countries);
    %>

    <a href="showcase.jsp">Show the details</a>
</body>
</html>
//-----
Showcase.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.example.User"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page import="java.util.Map, java.util.HashMap"%>
    <%@ page import="java.util.List, java.util.ArrayList"%>
<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>

    <h2>User Details</h2>
    <p>Name: ${sessionScope.user.name}</p>
    <!-- Calls user.getName() -->
    <p>Age: ${sessionScope.user.age}</p>
    <!-- Calls user.getAge() -->

    <h2>Names List</h2>
    <p>First Name: ${sessionScope.names[0]}</p>
    <!-- Alice -->
    <p>Second Name: ${sessionScope.names[1]}</p>
    <!-- Bob -->

    <h3>Iterating Over List</h3>
    <ul>
        <c:forEach var="name" items="${sessionScope.names}">
            <li>${name}</li>
        </c:forEach>
    </ul>

    <h2>Country Codes</h2>
    <p>US: ${sessionScope.countries.US}</p>
    <!-- United States -->
    <p>IN: ${sessionScope.countries["IN"]}</p>
    <!-- India -->
    <p>UK: ${sessionScope.countries['UK']}</p>
    <!-- United Kingdom -->

    <h3>Iterating Over Map</h3>
    <ul>
        <c:forEach var="entry" items="${sessionScope.countries}">
            <li>${entry.key}- ${entry.value}</li>
        </c:forEach>
    </ul>
</body>
</html>
```

JSTL (JavaServer Pages Standard Tag Library)

JSTL (JavaServer Pages Standard Tag Library) is a collection of predefined JSP tags that simplify the development of JSP pages by reducing the need for Java code (scriptlets). It provides reusable, standard tags for common tasks such as conditional processing, loops, internationalization, and database interactions.

List of JSTL Tag Categories & Their Tags: JSTL is divided into five major categories:

Core Tags (c prefix - <http://java.sun.com/jsp/jstl/core>)

- `<c:out>` → Prints values (like `System.out.println()`).
- `<c:set>` → Sets a variable in a specific scope.
- `<c:remove>` → Removes a variable from a scope.
- `<c:if>` → Conditional statement (like `if` in Java).
- `<c:choose>`, `<c:when>`, `<c:otherwise>` → Switch-case equivalent.
- `<c:forEach>` → Iterates over collections or arrays (like `for` loop).
- `<c:forTokens>` → Iterates over tokens in a delimited string.
- `<c:import>` → Imports content from another resource (JSP, HTML, etc.).
- `<c:redirect>` → Redirects to another page.
- `<c:url>` → Constructs a URL with parameters.

Formatting Tags (fmt prefix - <http://java.sun.com/jsp/jstl/fmt>)

- `<fmt:formatNumber>` → Formats numbers (currency, percentage, etc.).
- `<fmt:parseNumber>` → Parses a formatted number.
- `<fmt:formatDate>` → Formats dates and times.
- `<fmt:parseDate>` → Parses a date string into a `Date` object.
- `<fmt:setLocale>` → Sets the locale for internationalization.
- `<fmt:bundle>` → Loads a resource bundle for localization.
- `<fmt:message>` → Retrieves messages from a resource bundle.

SQL Tags (sql prefix - <http://java.sun.com/jsp/jstl/sql>)

- `<sql:setDataSource>` → Defines a database connection.
- `<sql:query>` → Executes an SQL `SELECT` query.
- `<sql:update>` → Executes SQL `INSERT`, `UPDATE`, or `DELETE` statements.
- `<sql:param>` → Sets query parameters.
- `<sql:dateParam>` → Sets date parameters in SQL queries.

XML Tags (x prefix - <http://java.sun.com/jsp/jstl/xml>)

- `<x:parse>` → Parses XML data.
- `<x:out>` → Extracts and prints XML data.
- `<x:forEach>` → Iterates over XML nodes.
- `<x:if>` → Conditional processing in XML.

Functions (fn prefix - <http://java.sun.com/jsp/jstl/functions>)

- `<fn:contains>` → Checks if a string contains a substring.
- `<fn:startsWith>` / `<fn:endsWith>` → Checks if a string starts/ends with another string.
- `<fn:substring>` → Extracts a substring.
- `<fn:length>` → Finds the length of a string/collection.
- `<fn:toUpperCase>` / `<fn:toLowerCase>` → Converts case.
- `<fn:split>` → Splits a string into an array.

Why Use JSTL?

- ✓ Reduces Java code in JSP → Makes pages cleaner.
- ✓ Improves Readability & Maintainability → No scriptlets needed.
- ✓ Provides Standardized Tags → Reusable across multiple JSPs.
- ✓ Enhances Performance → Uses expression evaluation instead of Java code execution.

// Using JSTL Core Tags:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
  <title>JSTL Core Tags Demo</title>
</head>
<body>
  <h2>1. <!--c:set--> - Set Variables</h2>
  <c:set var="username" value="Yogesh Patil" scope="session"/>
  <c:set var="number" value="10"/>

  <h2>2. <!--c:out--> - Output Variables</h2>
  Hello, <c:out value="{username}" /> !<br>
  Number = <c:out value="{number}" />

  <h2>3. <!--c:if--> - Conditional Statement</h2>
  <c:if test="{number > 5}">
    Number is greater than 5<br>
  </c:if>

  <h2>4. <!--c:choose-->, <!--c:when-->, <!--c:otherwise--> - Switch Case</h2>
  <c:choose>
    <c:when test="{number == 5}">
      Number is five
    </c:when>
    <c:when test="{number == 10}">
      Number is ten
    </c:when>
    <c:otherwise>
      Number is unknown
    </c:otherwise>
  </c:choose>

  <h2>5. <!--c:forEach--> - Loop Over a List</h2>
  <c:set var="items" value="{['Java', 'Python', 'C++']}" />
  <ul>
    <c:forEach var="lang" items="{items}">
      <li><c:out value="{lang}" /></li>
    </c:forEach>
  </ul>
```



```
<h2>6. &lt;c:forTokens> - Split String</h2>
<c:set var="csv" value="Mango,Banana,Apple"/>
<ul>
  <c:forTokens var="fruit" items="{csv}" delims=",">
    <li><c:out value="{fruit}"/></li>
  </c:forTokens>
</ul>
```

7. <c:remove> - Remove a Variable

```
<c:remove var="number"/>
```

Number after remove: <c:out value="{number}" default="Removed"/>

8. <import> - Import External Resource

```
<import url="imported.jsp"/>
```

<url> - Create a URL with Parameters

```
<c:url var="newUrl" value="redirect.jsp">  
  <c:param name="user" value="Yogesh"/>  
</c:url>  
<a href="{newUrl}">Go to Redirect Page</a>
```

```
<h2>10. <:redirect> - Redirect Example (Auto redirect)</h2>
<!-- Uncomment this line to auto-redirect -->
<!-- <c:redirect url="redirect.jsp?user=Yogesh"/> --%>
</body>
</html>
```

//-----

// Formatting Tags:

```
<%@ page contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ page import="java.util.Date"%>
```

```
<html>
<head>
<title>JSTL Formatting Tags Example</title>
</head>
<body>
```

1. <fmt:setLocale> - Set Locale</h2> <fmt:setLocale value="en_US" />

2. <fmt:formatNumber> - Format Number</h2> <fmt:formatNumber value="123456.789" type="number" />
 <fmt:formatNumber value="123456.789" type="currency" />


```
<fmt:formatNumber value="0.85" type="percent" />
<br>
```

```
<h2>3. <fmt:parseNumber> - Parse Formatted Number</h2>
<fmt:parseNumber var="parsedNum" type="number" value="123,456.78" />
Parsed Number: ${parsedNum}
<br>
```

```
<h2>4. <fmt:formatDate> - Format Date</h2>
<jsp:useBean id="today" class="java.util.Date" />
Default Format:
<fmt:formatDate value="${today}" />
<br> Short Format:
<fmt:formatDate value="${today}" type="date" dateStyle="short" />
<br> Time Only:
<fmt:formatDate value="${today}" type="time" timeStyle="medium" />
<br>
```

```
<h2>5. <fmt:parseDate> - Parse Date</h2>
<fmt:parseDate var="parsedDate" value="09-Apr-2025"
    pattern="dd-MMM-yyyy" />
Parsed Date:
<fmt:formatDate value="${parsedDate}" type="both" dateStyle="medium"
    timeStyle="short" />
```

```
<h2>6. <fmt:bundle> and <fmt:message> - Localization</h2>
<fmt:bundle basename="messages">
    <p>
        <fmt:message key="welcome" />
    </p>
    <p>
        <fmt:message key="greeting" />
    </p>
</fmt:bundle>
```

```
</body>
</html>
```

//-----

// Using JSTL SQL Tags:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page import="java.util.Date" %>
```

```
<html>
<head>
    <title>JSTL SQL Tags Demo</title>
```

```
</head>
<body>
<h2>1. <sql:setDataSource> - Set Database Connection</h2>
<sql:setDataSource var="db" driver="com.mysql.cj.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/studentdb"
    user="root" password="Archer@12345" />
<h2>2. <sql:update> - Insert Student Record</h2>
<sql:update dataSource="{db}">
    INSERT INTO students (name, dob) VALUES (?, ?)
    <sql:param value="Yogesh Patil"/>
    <sql:dateParam value="<%= new java.util.Date() %>"/>
</sql:update>

<h2>3. <sql:query> - Select All Students</h2>
<sql:query dataSource="{db}" var="result">
    SELECT * FROM students
</sql:query>
<table border="1">
    <tr>
        <th>ID</th>
        <th>Name</th>
        <th>DOB</th>
    </tr>
    <c:forEach var="row" items="{result.rows}">
        <tr>
            <td><c:out value="{row.id}"/></td>
            <td><c:out value="{row.name}"/></td>
            <td><c:out value="{row.dob}"/></td>
        </tr>
    </c:forEach>
</table>

<h2>4. <sql:update> - Update Name Where ID=1</h2>
<sql:update dataSource="{db}">
    UPDATE students SET name = ? WHERE id = ?
    <sql:param value="Updated Name"/>
    <sql:param value="1"/>
</sql:update>

<h2>5. <sql:update> - Delete Where ID = 2</h2>
<sql:update dataSource="{db}">
    DELETE FROM students WHERE id = ?
    <sql:param value="2"/>
</sql:update>
</body>
</html>
//-----
```

// Using JSTL Functions:

```
<%@ page contentType="text/html; charset=UTF-8" language="java"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<title>JSTL Functions Demo</title>
</head>
<body>
    <h2>JSTL Functions (fn prefix) Demo</h2>
    <c:set var="text" value="Welcome to JSTL Functions in Java!" />
    <c:set var="nameList" value="Yogesh,Arjun,Neha,Ravi" />

    <h3>1. &lt;fn:contains&gt; - Contains 'JSTL'</h3>
    <c:if test="{fn:contains(text, 'JSTL')}">
        Yes, the string contains 'JSTL'.
    </c:if>

    <h3>2. &lt;fn:startsWith&gt; / &lt;fn:endsWith&gt;</h3>
    Starts with 'Welcome'?
    <c:out value="{fn:startsWith(text, 'Welcome')}" />
    <br> Ends with 'Java'?
    <c:out value="{fn:endsWith(text, 'Java!')}" />

    <h3>3. &lt;fn:substring&gt; - Substring from index 11 to 16</h3>
    <c:out value="{fn:substring(text, 11, 16)}" />
    <!-- Output: JSTL -->

    <h3>4. &lt;fn:length&gt; - Length of text and nameList</h3>
    Text length:
    <c:out value="{fn:length(text)}" />
    <br>
    <c:set var="namesArray" value="{fn:split(nameList, ',')}" />
    Name list count:
    <c:out value="{fn:length(namesArray)}" />

    <h3>5. &lt;fn:toUpperCase&gt; / &lt;fn:toLowerCase&gt;</h3>
    Upper:
    <c:out value="{fn:toUpperCase(text)}" />
    <br> Lower:
    <c:out value="{fn:toLowerCase(text)}" />

    <h3>6. &lt;fn:split&gt; - Iterate over names</h3>
    <ul>
        <c:forEach var="n" items="{namesArray}">
            <li><c:out value="{n}" /></li>
        </c:forEach>
    </ul>
</body>
</html>
```

Using the RequestDispatcher:

In Servlet-JSP, the RequestDispatcher is a key interface used to forward a request from one resource to another or to include the content of another resource in the response. The resources can be Servlets, JSPs, or HTML files on the same server.

What is RequestDispatcher?

RequestDispatcher is an interface provided by **javax.servlet** package.

It is used to:

1. **Forward a request** to another resource (like another servlet or JSP).
2. **Include content** from another resource in the response.

When to Use RequestDispatcher?

You use RequestDispatcher when:

- You want to **break your application logic** into multiple components.
- You want to **reuse** a part of a page or a servlet logic.
- You want to **transfer control** from one servlet to another **on the server side** (without client's involvement).
- You want to **include** a header/footer or common data in multiple pages (like a menu bar).

Types of Dispatching

1. Forwarding a Request:

- Transfers control to another resource.
- The **original request and response** are passed along.
- The client **does not know** the change; URL remains the same.

```
RequestDispatcher rd = request.getRequestDispatcher("nextPage.jsp");  
rd.forward(request, response);
```

2. Including a Resource:

- Includes the content of another resource **in the response**.
- Useful for including headers, footers, menus.

```
RequestDispatcher rd = request.getRequestDispatcher("header.jsp");  
rd.include(request, response);
```

Example

Example 1: Forwarding from Servlet to JSP

```
// Inside a servlet  
String username = request.getParameter("username");  
request.setAttribute("user", username);  
  
RequestDispatcher rd = request.getRequestDispatcher("welcome.jsp");  
rd.forward(request, response);
```

In welcome.jsp:

```
Hello, ${user}!
```

Example: Student CRUD using RequestDispatcher, EL and JSTL with MYSQL

Welcome to Student Management System

Add StudentView Students

Edit Student

Name

Email

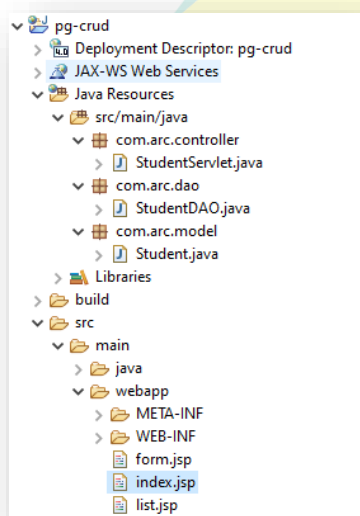
SaveCancel

Student Records

+ Add Student

ID	Name	Email	Actions
2	amar	am@gmail.com	<button>Edit</button> <button>Delete</button>
3	ram	ram@gmail.com	<button>Edit</button> <button>Delete</button>
4	sham	sham@gmail.com	<button>Edit</button> <button>Delete</button>
5	amar	am@gmail.com	<button>Edit</button> <button>Delete</button>

Project Structure:



// Index.jsp

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Student Management</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin-top: 100px;
    }
    a {
      display: inline-block;
      margin: 20px;
      padding: 10px 25px;
      font-size: 18px;
      text-decoration: none;
      background-color: #4CAF50;
      color: white;
      border-radius: 5px;
    }
    a:hover {
      background-color: #45a049;
    }
  </style>
</head>
<body>

  <h1>Welcome to Student Management System</h1>

  <a href="form.jsp">Add Student</a>
  <a href="list.jsp">View Students</a>

</body>
</html>
```

// form.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
  <title>Student Form</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
```

```
<body class="container mt-5">
  <div class="card p-4 shadow-sm">
    <h2 class="mb-4">
      <c:choose>
        <c:when test="\${student.id == 0}">Add New Student</c:when>
        <c:otherwise>Edit Student</c:otherwise>
      </c:choose>
    </h2>
    <form action="student" method="post">
      <input type="hidden" name="action" value="\${student.id == 0 ? 'insert' : 'update'}"/>
      <input type="hidden" name="id" value="\${student.id}"/>

      <div class="mb-3">
        <label class="form-label">Name</label>
        <input class="form-control" type="text" name="name" value="\${student.name}" required/>
      </div>

      <div class="mb-3">
        <label class="form-label">Email</label>
        <input class="form-control" type="email" name="email" value="\${student.email}"
required/>
      </div>

      <button type="submit" class="btn btn-primary">Save</button>
      <a href="student" class="btn btn-secondary">Cancel</a>
    </form>
  </div>
</body>
</html>
```

// list.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page import="java.util.*, com.arc.model.Student, com.arc.dao.StudentDAO" %>

<%
  // Load students from database on page load
  StudentDAO studentDAO = new StudentDAO();
  List<Student> studentList = studentDAO.getAll();

  // Set the list in request scope
  request.setAttribute("studentList", studentList);
%>
<!DOCTYPE html>
<html>
<head>
  <title>Student List</title>
```



```

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body class="container mt-5">
  <div class="d-flex justify-content-between align-items-center mb-4">
    <h2>Student Records</h2>
    <a href="student?action=new" class="btn btn-success">+ Add Student</a>
  </div>

  <c:choose>
    <c:when test="${not empty studentList}">
      <table class="table table-bordered table-hover">
        <thead class="table-dark">
          <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Email</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <c:forEach var="s" items="${studentList}">
            <tr>
              <td>${s.id}</td>
              <td>${s.name}</td>
              <td>${s.email}</td>
              <td>
                <a href="student?action=edit&id=${s.id}" class="btn btn-sm btn-warning">Edit</a>
                <a href="student?action=delete&id=${s.id}" class="btn btn-sm btn-danger"
                  onclick="return confirm('Are you sure you want to delete this
student?');">Delete</a>
              </td>
            </tr>
          </c:forEach>
        </tbody>
      </table>
    </c:when>
    <c:otherwise>
      <div class="alert alert-info">No students found. Please add some!</div>
    </c:otherwise>
  </c:choose>
</body>
</html>

```

// Student.java

```
package com.arc.model;
```

```
public class Student {
    private int id;
    private String name;
    private String email;

    // Constructors
    public Student() {}

    public Student(int id, String name, String email) {
        this.id = id; this.name = name; this.email = email;
    }

    public Student(String name, String email) {
        this.name = name; this.email = email;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}

// StudentServlet .java

package com.arc.controller;

import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

import com.arc.dao.StudentDAO;
import com.arc.model.Student;

import java.io.IOException;
@WebServlet("/student")
public class StudentServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private StudentDAO dao;

    public void init() {
        dao = new StudentDAO();
    }
}
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
IOException, ServletException {  
    doGet(request, response);  
}
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
IOException, ServletException {  
    String action = request.getParameter("action");  
    System.out.println("Get: "+action);
```

```
    if (action == null) action = "list";
```

```
    switch (action) {  
        case "new":  
            showForm(request, response);  
            break;  
        case "insert":  
            insertStudent(request, response);  
            break;  
        case "edit":  
            showEditForm(request, response);  
            break;  
        case "update":  
            updateStudent(request, response);  
            break;  
        case "delete":  
            deleteStudent(request, response);  
            break;  
        default:  
            listStudents(request, response);  
    }  
}
```

```
private void listStudents(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {  
    request.setAttribute("studentList", dao.getAll());  
    RequestDispatcher dispatcher = request.getRequestDispatcher("list.jsp");  
    dispatcher.forward(request, response);  
}
```

```
private void showForm(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {  
    request.setAttribute("student", new Student());  
    RequestDispatcher dispatcher = request.getRequestDispatcher("form.jsp");  
    dispatcher.forward(request, response);  
}
```

```
private void showEditForm(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    int id = Integer.parseInt(request.getParameter("id"));
    Student student = dao.get(id);
    request.setAttribute("student", student);
    RequestDispatcher dispatcher = request.getRequestDispatcher("form.jsp");
    dispatcher.forward(request, response);
}
```

```
private void insertStudent(HttpServletRequest request, HttpServletResponse response) throws
IOException {
    String name = request.getParameter("name");
    String email = request.getParameter("email");
    dao.insert(new Student(name, email));
    response.sendRedirect("student");
}
```

```
private void updateStudent(HttpServletRequest request, HttpServletResponse response) throws
IOException {
    int id = Integer.parseInt(request.getParameter("id"));
    String name = request.getParameter("name");
    String email = request.getParameter("email");
    dao.update(new Student(id, name, email));
    response.sendRedirect("student");
}
```

```
private void deleteStudent(HttpServletRequest request, HttpServletResponse response) throws
IOException {
    int id = Integer.parseInt(request.getParameter("id"));
    dao.delete(id);
    response.sendRedirect("student");
}
}
```

// StudentDAO.java

```
package com.arc.dao;
```

```
import java.sql.*;
import java.util.*;
```

```
import com.arc.model.Student;
```

```
public class StudentDAO {
    private final String JDBC_URL = "jdbc:postgresql://localhost:5432/studentdb";
    private final String USER = "postgres";
    private final String PASSWORD = "Archer@12345";

    private Connection connect() throws SQLException {
```

```
        return DriverManager.getConnection(JDBC_URL, USER, PASSWORD);
    }

    public List<Student> getAll() {
        List<Student> list = new ArrayList<>();
        String sql = "SELECT * FROM student ORDER BY id";

        try (Connection conn = connect();
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                list.add(new Student(rs.getInt("id"), rs.getString("name"), rs.getString("email")));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return list;
    }

    public void insert(Student student) {
        String sql = "INSERT INTO student(name, email) VALUES (?, ?)";
        try (Connection conn = connect();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, student.getName());
            pstmt.setString(2, student.getEmail());
            pstmt.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void update(Student student) {
        String sql = "UPDATE student SET name = ?, email = ? WHERE id = ?";
        try (Connection conn = connect();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, student.getName());
            pstmt.setString(2, student.getEmail());
            pstmt.setInt(3, student.getId());
            pstmt.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void delete(int id) {
        String sql = "DELETE FROM student WHERE id = ?";
        try (Connection conn = connect();
```

```
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            pstmt.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public Student get(int id) {
        Student student = null;
        String sql = "SELECT * FROM student WHERE id = ?";
        try (Connection conn = connect();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                student = new Student(rs.getInt("id"), rs.getString("name"), rs.getString("email"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return student;
    }
}
```

The Archer logo is a stylized graphic consisting of two overlapping, curved shapes. The shape on the left is a light teal color, and the shape on the right is a light yellow color. They overlap in the center, creating a darker teal area. Below this graphic, the word "Archer" is written in a large, light teal, sans-serif font.

Archer