Servlet

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

```
package com.example;

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

public class MyServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse respons
        // Get form data
        String name = request.getParameter("name");
        String email = request.getParameter("email");

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

        // Save data to database by calling saveToDb function
        try {
            // Assuming the ID is auto-generated by the database, you can pass a pla
            int id = 0;  // Modify this as needed based on your requirement
            saveToDb(id, name, email);
            out.println("<h1>User saved successfully!</h1>");
        } catch (Exception e) {
            out.println("<h1>Error: " + e.getMessage() + "</h1>");
        }
    }

    // Separate function to save data to the database
    private void saveToDb(int id, String name, String email) {
        try {
            // Directly specify connection details
            String url = "jdbc:mysql://localhost:3306/your_database";  // Update the c
            String user = "root";  // Database username
            String password = "password";  // Database password
            Class.forName("com.mysql.cj.jdbc.Driver");
            // Establish the connection
            Connection con = DriverManager.getConnection(url, user, password);// P
            String query = "INSERT INTO users (id, name, email) VALUES (?, ?, ?)";
            PreparedStatement preparedStatement = con.prepareStatement(query);
            preparedStatement.setInt(1, id);
            preparedStatement.setString(2, name);
            preparedStatement.setString(3, email);

            // Execute the query
            preparedStatement.executeUpdate();
            System.out.println("Data inserted")
            // Close resources
            preparedStatement.close();
            con.close();
        } catch (Exception e) {
            System.out.println("Error while saving data: " + e.getMessage());
        }
    }
}
```

Servlet to jsp

```
index.jsp

<html>
```

```
<body>
<form method = "POST" action = "login">
 Username :<input type="name" name="username"></br>
 Password:<input type="password" name="password"></br>
 <button type="submit">submit</button>
</form>
</html>
</body>
<%
     String username = request.getParameter("username");
     String password = request.getParameter("password");
 %>
 <%= username%>
 <%= password%>
```

```
Servlet file
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ForwardToJspServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse respo
        // Retrieve data from the request
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // Forward the request to the JSP page, passing the username as a query pa
        RequestDispatcher dispatcher = request.getRequestDispatcher("index.jsp")
        dispatcher.forward(request, response);
    }
}
```

```
web.xml file
<web-app>
<servlet>
    <servlet-name>ForwardToJspServlet</servlet-name>
    <servlet-class>ForwardToJspServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ForwardToJspServlet</servlet-name>
    <url-pattern>/login</url-pattern>
</servlet-mapping>

</web-app>
```

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

public class ForwardToJspServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse respons
         request.setContentType("text/html");
        // Retrieve data from the request
        String username = request.getParameter("username");
        String password = request.getParameter("password");




        // Include the JSP page in the response
        RequestDispatcher dispatcher = request.getRequestDispatcher("index.jsp")
        dispatcher.include(request, response);
    }
}
```

Servlet to Servlet

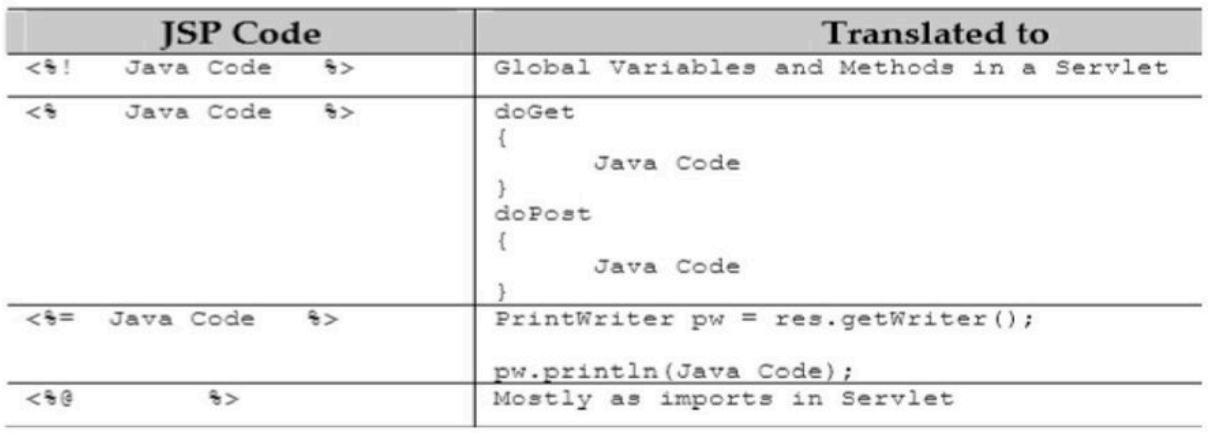
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse respons
        // Retrieve form data
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // Forward the request to SecondServlet
        RequestDispatcher dispatcher = request.getRequestDispatcher("SecondSer
        dispatcher.forward(request, response);
    }
}
```

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

public class SecondServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse respons
        // Retrieve forwarded data from FirstServlet
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // Forward the request to a JSP for rendering the response
        RequestDispatcher dispatcher = request.getRequestDispatcher("output.jsp"
        dispatcher.forward(request, response);
    }
}
```

```
index.html
```

```
<html>
<body>
<form method = "POST" action = "login">
 Username :<input type="name" name="username"></br>
 Password:<input type="password" name="password"></br>
 <button type="submit">submit</button>
</form>
</html>
</body>
output.jsp
<%
     String username = request.getParameter("username");
     String password = request.getParameter("password");
 %>
 <%= username%>
 <%= password%>
```

JSP



JSP Code	Translated to
`<%! Java Code %>`	Global Variables and Methods in a Servlet

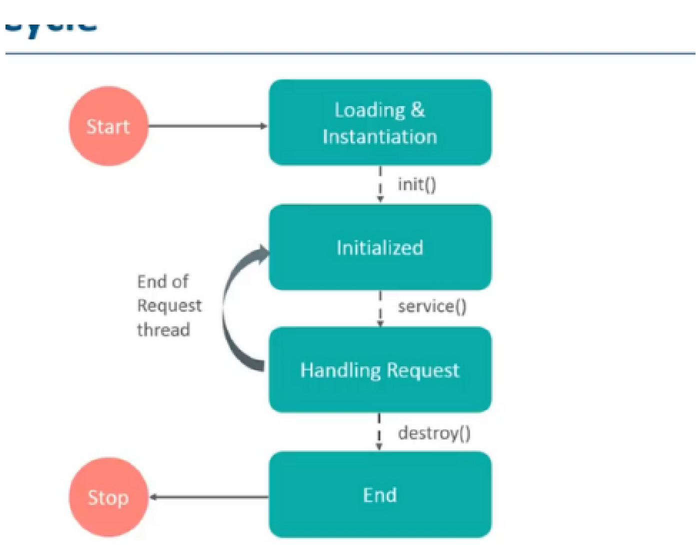
`<% Java Code %>`	doGet
	{
	Java Code
	}
	doPost
	{
	Java Code
	}

`<%= Java Code %>`	PrintWriter pw = res.getWriter();
	
	pw.println(Java Code);

`<%@ %>`	Mostly as imports in Servlet

[Servlet .pdf]

Chapter 7 Servlet.pptx



Summary:

- **Servlets** are used for handling requests and logic, while **JSP** is better for presenting dynamic content in a simpler way.

Based on the image, let me explain the Servlet lifecycle:

1. **Loading & Instantiation**

- First stage when the servlet container loads and creates an instance of the servlet

2. **Initialization**

- After instantiation, the init() method is called
- This happens only once in the servlet's lifecycle

3. **Request Handling**

- The service() method handles client requests
- This phase repeats for each new request

- The servlet remains in memory handling multiple requests

4. **End/Destruction**

- When the servlet container decides to remove the servlet
- The destroy() method is called
- Cleanup operations are performed

The cycle shows how a servlet thread continues to handle requests until it reaches the end of the request thread, at which point it returns to the "Handling Request" state for new requests, or moves to the "End" state when the servlet is being shut down.

Here's a simplified comparison between **Servlet** and **JSP**:

Feature	**Servlet**	**JSP (JavaServer Pages)**
Definition	A Java class that handles HTTP requests.	A page with embedded Java code that generates HTML.
Main Use	Processes requests and generates responses.	Displays dynamic content in web pages.
Syntax	Java code in a class.	HTML with Java code inside `<% %>` tags.
File Extension	`.java` (Java file).	`.jsp` (JSP file).
Development	More complex, requires separate HTML & Java.	Easier for mixing Java with HTML.
Separation of Concerns	Business logic and HTML are mixed.	Business logic and HTML are separate.
Performance	Typically faster for complex logic.	May have some overhead due to HTML and Java mix.
Use Case	Good for handling logic-heavy requests.	Best for displaying content dynamically in HTML.

What is a Deployment Descriptor?

A **Deployment Descriptor** is an XML file (`web.xml`) used to configure a Java web application in a servlet container (like Tomcat). It defines how servlets, filters, listeners, and other components are configured and mapped to handle specific requests.

How is it Used in a Servlet?

In servlets, the `web.xml` file is used to:

1. **Define servlets**: Specify the class name of the servlet and associate it with a URL pattern.
2. **Map URLs to servlets**: Define which URLs should be handled by which servlet.
3. **Configure initialization parameters**: Set values that the servlet can access during its initialization.
4. **Set filters, listeners, and security**: Configure additional components like filters (to process requests), listeners (for lifecycle events), and security constraints.

Example of `web.xml`:

```
<web-app>
  <!-- Define a servlet -->
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>com.example.MyServlet</servlet-class>
  </servlet>

  <!-- Map the servlet to a URL pattern -->
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/myServlet</url-pattern>
  </servlet-mapping>

  <!-- Set initialization parameters for the servlet -->
  <context-param>
```

```
    <param-name>appName</param-name>
    <param-value>MyWebApp</param-value>
  </context-param>
</web-app>
```

Key Sections:

1. `<servlet>`: Defines a servlet by providing a name and the fully qualified class name.
2. `<servlet-mapping>`: Maps a specific URL pattern (like `/myServlet`) to the servlet, so the servlet knows which requests to handle.
3. `<context-param>`: Global parameters that can be accessed by the servlet.

Use in Servlet:

- When a user accesses the URL `/myServlet`, the servlet container uses the `web.xml` file to map the request to the `MyServlet` class, which then handles the request and sends a response.
- Initialization parameters can be accessed in the servlet via `getServletConfig().getInitParameter("param-name")`.

In modern applications, **annotations** are often used instead of `web.xml`, but `web.xml` is still supported and useful for complex configurations.

A **static web page** displays the same content to every visitor and doesn't change unless manually updated. It is built using HTML and does not involve any server-side processing. For example, a simple company info page that always shows the same text is static.

A **dynamic web page**, on the other hand, changes its content based on user interaction or data from a server. It involves server-side technologies like PHP, Java, or databases to generate different content. For example, a social media feed that updates with new posts is dynamic.

In **JSP (JavaServer Pages)**, **implicit objects** are pre-defined objects provided by the JSP container. These objects are available to developers without needing to declare or instantiate them explicitly, making it easier to access common web application data like request information, session data, or application settings.

Common Implicit Objects in JSP:

1. `request`:
    - **Type**: `HttpServletRequest`
    - **Use**: Represents the client's request. It allows you to get request data such as form parameters, headers, cookies, etc.
    - **Example**: `request.getParameter("username");` retrieves the value of a form field named `username`.
2. `response`:
    - **Type**: `HttpServletResponse`
    - **Use**: Represents the response to the client. It allows you to modify the response, like setting headers, redirecting, or writing output.
    - **Example**: `response.sendRedirect("home.jsp");` redirects the user to another page.
3. `session`:
    - **Type**: `HttpSession`
    - **Use**: Represents the session between the client and the server. You can store user-related data across multiple requests.
    - **Example**: `session.setAttribute("user", "John");` stores a value in the session.
4. `application`:
    - **Type**: `ServletContext`
    - **Use**: Represents the servlet context (application-wide data). It allows you to share data across all users and servlets in the application.
    - **Example**: `application.getAttribute("appConfig");` retrieves a shared attribute.
5. `out`:
    - **Type**: `JspWriter`
    - **Use**: Used to send content (HTML, text) to the client's browser.
    - **Example**: `out.println("Hello, World!");` sends the message to the web page.
6. `config`:

- **Type**: `ServletConfig`
- **Use**: Provides servlet configuration data like initialization parameters.
- **Example**: `config.getInitParameter("adminEmail");` gets the init parameter from `web.xml`.

7. **`pageContext`**:
    - **Type**: `PageContext`
    - **Use**: Provides access to all the other implicit objects and can be used for handling session, application, and request-scoped attributes.
    - **Example**: `pageContext.getAttribute("attributeName", PageContext.REQUEST_SCOPE);`
8. **`page`**:
    - **Type**: `Object` (typically refers to the JSP page itself)
    - **Use**: Refers to the current JSP page. Mostly used internally.
    - **Example**: Rarely used explicitly, but it refers to the JSP page object.
9. **`exception`**:
    - **Type**: `Throwable`
    - **Use**: Available only in JSP error pages. It represents the exception that caused the error.
    - **Example**: `exception.getMessage();` retrieves the error message.

Example of Using Implicit Objects in JSP:

```
<%@ page language="java" contentType="text/html" %>
<html>
<body>
  <!-- Using 'request' to get a parameter from a form →
  <p>Username: <%= request.getParameter("username") %></p>

  <!-- Using 'session' to store user information →
  <%
    session.setAttribute("user", "John Doe");
  %>
  <p>Welcome, <%= session.getAttribute("user") %>!</p>

  <!-- Using 'application' to get a shared attribute →
  <p>Application Name: <%= application.getAttribute("appName") %></p>

  <!-- Using 'out' to write output to the response →
  <p><%= out.println("This message is from the out object.") %></p>
</body>
</html>
```

Summary of Implicit Object Use:

- `request`: Get request data like form parameters.
- `response`: Modify the response, send redirects, etc.
- `session`: Store data between user requests.
- `application`: Share data across the entire application.
- `out`: Send content to the client's browser.
- `config`: Get initialization parameters.
- `pageContext`: Access all other implicit objects.
- `page`: Refers to the current page (rarely used).
- `exception`: Used in error pages to handle exceptions.

These objects help simplify common web application tasks like handling requests, managing sessions, or sharing data.

```
<!-- studentForm.jsp →
<html>
<head>
   <title>Student Attendance Form</title>
</head>
<body>
   <form method="post" action="form" >
      StudentID:<input type="text" name="studentId" ><br/><br/>
```

```
        StudentAttendance(%):<input type="number" name="attendance" ><br/><
        <button type="submit"> submit </button>
    </form>
</body>
</html>
```

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

public class MyServlet extends HttpServlet {


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

        // Get the student ID and attendance from the request
        String studentId = request.getParameter("studentId");
        int attendance = Integer.parseInt(request.getParameter("attendance"));

        // Eligibility criteria: attendance must be 75% or higher
        if (attendance >= 75) {
            // Call the method to save to the database
            boolean success = saveToDb(studentId, out);
            if (success) {
                out.println("<h3>Student with ID " + studentId + " is now eligible for the
            } else {
                out.println("<h3>Student ID not found or database error.</h3>");
            }
        } else {
            out.println("<h3>Student with ID " + studentId + " is NOT eligible. Minimu
        }
    }

    // Method to handle database connection and update the student's eligibility
    private boolean saveToDb(String studentId, PrintWriter out) {
        boolean updated = false;
        try {
                String url = "jdbc:mysql://localhost:3306/your_database";
                String username = "root";
                String password = "password";
                Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url, username, passwor

            // Update query to mark the student as eligible for the exam
            String query = "UPDATE students SET eligible = 1 WHERE student_id = ?";
            PreparedStatement pstmt = conn.prepareStatement(query);
            pstmt.setString(1, studentId);

           int rowsUpdated = pstmt.executeUpdate();
            if (rowsUpdated > 0) {
                updated = true;
            }
            pstmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
            out.println("<h3>Error: " + e.getMessage() + "</h3>");
        }
        return updated;
    }
}
```

```
<web-app>
    <servlet>
```

```
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>MyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/form</url-pattern>
    </servlet-mapping>
</web-app>
```

Here's the updated example using **two JSPs only**, and excluding the `@import` directive as requested. I'll keep the logic simple and show the request forwarding from one JSP to another using `RequestDispatcher`.

1. First JSP (sender.jsp): This is where the user enters their username.

```
<html>
<body>
    <form method="post" action="processData.jsp">
        Enter your name: <input type="text" name="username"><br>
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

2. Second JSP (processData.jsp): This is where the data is processed, and the request is forwarded to another JSP (`welcome.jsp`).

```
<%
    // Retrieve the username parameter from the request
    String username = request.getParameter("username");

    // Set the attribute to be passed to the next JSP
    request.setAttribute("username", username);

    // Forward the request to the next JSP page (welcome.jsp)
    RequestDispatcher dispatcher = request.getRequestDispatcher("welcome.j
sp");
    dispatcher.forward(request, response);
%>
```

3. Third JSP (welcome.jsp): This is where the forwarded request is received and the username is displayed.

```
<html>
<body>
    <h1>Welcome!</h1>
    <p>Hello, <%= request.getAttribute("username") %>!</p>
</body>
</html>
```

Flow:

1. **`sender.jsp`**: A simple form where the user can enter their username and submit the form. The form sends the data to `processData.jsp`.
2. **`processData.jsp`**: This page retrieves the `username` from the request, sets it as an attribute, and forwards the request to `welcome.jsp`.
3. **`welcome.jsp`**: The forwarded request from `processData.jsp` receives the username via `request.getAttribute()` and displays it.

Key Points:

- The `@import` directive is omitted.
- The `RequestDispatcher` forwards the request to another JSP (`welcome.jsp`) from `processData.jsp`.
- This example shows how to use two JSPs for handling form data and forwarding requests in a simple manner.