

# Advanced Computer Programming

## Java Servlets

### Web Development in Java

By:

Zohair Ahmed



 [www.youtube.com/@zohairahmed007](https://www.youtube.com/@zohairahmed007)

**Subscribe**

 [www.begindiscovery.com](http://www.begindiscovery.com)

# Web Request Cycle

---



# Web Request Cycle

---

- The web request cycle (or HTTP request cycle) is the sequence of steps that occur whenever a client (like a web browser or mobile app) makes a request to a web server. This cycle happens for every interaction between the client and the server and is central to how web applications work, regardless of the underlying technologies used (Servlets, frameworks, or APIs).

## 1. Client Sends a Request

- **Action:** The client (e.g., a browser or mobile app) sends an HTTP request to the server. This request can be triggered by various user actions, like entering a URL in the browser, clicking a link, submitting a form, or loading an app.
- **Request Type:** The HTTP request may be a GET, POST, PUT, DELETE, etc., depending on the action.
- **For example:**
  - GET: Requesting a resource (such as an HTML page, image, etc.).
  - POST: Sending data to the server (like form submission).
- **Example Request:**
  - A user clicks a link to **`https://example.com/page`**.
  - The browser sends an HTTP GET request to the server.



# Web Request Cycle

---

## 2. DNS Resolution and TCP Connection

- Before the server can process the request, the domain name system (DNS) resolves the domain (e.g., example.com) to an IP address of the web server.
- Once the IP address is obtained, a TCP connection is established between the client and the server.

## 3. Server Receives the Request

- The server that hosts the website or application receives the incoming HTTP request.
- The server checks the requested resource (e.g., a file or endpoint) to decide how to handle the request. This depends on the server configuration (**e.g., Apache, Nginx, or an application server like Node.js or Django**).

# Web Request Cycle

---

## 4. Request Processing by the Server

- The server processes the request in one of several ways:
- If the request is for static content (like an HTML file, image, or CSS), the server retrieves and serves the static resource.
- If the request is for dynamic content, the server might call a backend application, a database, or other services to generate the response.
- This step can involve complex logic, such as:
  - Fetching data from a database.
  - Running server-side scripts or business logic.
  - Interacting with other services or APIs.

## 5. Server Generates Response

- After processing, the server prepares the HTTP response. This consists of:
- Status Code: A numerical code that indicates the outcome of the request (e.g., 200 OK for success, 404 Not Found for an error).
- Headers: Metadata about the response, like the content type (e.g., Content-Type: text/html) and cache control.
- Body: The actual content that will be sent to the client, like HTML, JSON, images, or any other data requested by the client.



# Web Request Cycle

---

## 6. Server Sends Response to the Client

- The server sends the HTTP response back to the client over the established TCP connection.
- The response includes:
  - A status code to indicate if the request was successful (e.g., 200 OK) or there was an error (e.g., 404 Not Found).
  - The headers with meta-information.
  - The body with the actual data (HTML, JSON, etc.).

## 7. Client Processes the Response

- If the response contains HTML, the browser will render the page.
- If the response contains JavaScript or CSS, the browser will execute or apply it accordingly.
- If the response contains data (like JSON for an API), the client may use it to update the UI or perform other tasks.
- If the content requires further resources (such as images, CSS, or additional JavaScript files), the browser will make additional requests to the server for those resources.





# Web Request Cycle

---

## 8. Browser Renders the Content

- The browser displays the content (HTML page, images, styles) to the user.
- If the page includes client-side JavaScript, it may perform additional actions, like fetching data from an API or updating parts of the page dynamically (AJAX).

**Example:** If the page includes an image, the browser will make an additional request for the image file. Once it gets the image, it displays it on the webpage.

## 9. Client Interaction and Additional Requests

- After the page is displayed, further interactions (e.g., clicking buttons, submitting forms, or navigating between pages) may result in additional HTTP requests.
- These interactions are typically asynchronous, allowing the user to interact with the application while new data is being fetched or processed in the background.



# Summary of the Web Request Cycle:

---

- **Client Sends Request:** The client (browser or app) sends an HTTP request (e.g., GET, POST).
- **DNS Resolution:** The domain is resolved to an IP address, and a TCP connection is established.
- **Server Receives Request:** The server receives the request and processes it.
- **Request Processing:** The server processes the request (fetches data, runs logic).
- **Server Generates Response:** The server generates an HTTP response, including status code, headers, and body.
- **Server Sends Response:** The response is sent back to the client.
- **Client Processes Response:** The client processes and renders the content.
- **Rendering:** The client renders the page and may request additional resources (images, scripts).
- **Further Requests:** As needed, the client makes additional requests to load more resources.
- **Client Interaction:** The client may trigger more requests based on user interactions (e.g., form submissions).



# Servlet

---

# Servlet

---

- A **Servlet** in Java is a server-side program that handles client requests and generates dynamic responses, typically in the form of web pages.
- It is part of the Java EE (Enterprise Edition) specification, specifically designed to handle HTTP requests and produce dynamic content, usually for web applications.
  - Key points about Servlets:
- **Server-side execution:** Servlets are executed on the server, rather than on the client (like JavaScript or HTML). This makes them a key component in creating dynamic web applications.
- **Java API:** The servlet is part of the javax.servlet package and runs within a Servlet Container (such as Apache Tomcat, Jetty, or GlassFish). The servlet container is responsible for managing the lifecycle of a servlet.
- **HTTP Servlet:** Most commonly, a servlet extends the HttpServlet class, which provides methods such as doGet() and doPost() to handle HTTP GET and POST requests, respectively.
- **Request-Response Model:** Servlets follow a request-response model where the client sends an HTTP request to the server, and the servlet processes it, usually generating dynamic content (such as HTML) in response.



# Servlet

---

- The javax.servlet package is not part of the standard JDK (Java Development Kit). Instead, it is part of the Servlet API, which is a separate specification provided by Java EE (Enterprise Edition) or Jakarta EE.
- To work with servlets, you need a Servlet Container or a Java EE server (now part of Jakarta EE), such as:
  - Apache Tomcat
  - GlassFish
  - WildFly (formerly JBoss)
  - Jetty

These servers provide the necessary environment for running servlets and implementing the Servlet API.

- JDK itself does not include the javax.servlet package.
- To use servlets, you either need to:
  - Add a Servlet container (e.g., Apache Tomcat, GlassFish), which implements the Servlet API.
  - Use a Java EE (Jakarta EE) application server that includes support for Servlets, JSPs, and other web technologies.
- In summary, servlets are part of the Servlet API, which is provided by application servers or servlet containers like GlassFish or Tomcat, not the standard JDK.

## Eclipse IDE 2023-09 R Packages



### Eclipse IDE for Java Developers

319 MB

900,606 DOWNLOADS

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration



Windows x86\_64  
macOS x86\_64 | AArch64  
Linux x86\_64 | AArch64



### Eclipse IDE for Enterprise Java and Web Developers

518 MB

461,683 DOWNLOADS

Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data Tools, Maven and Gradle, Git, and more.

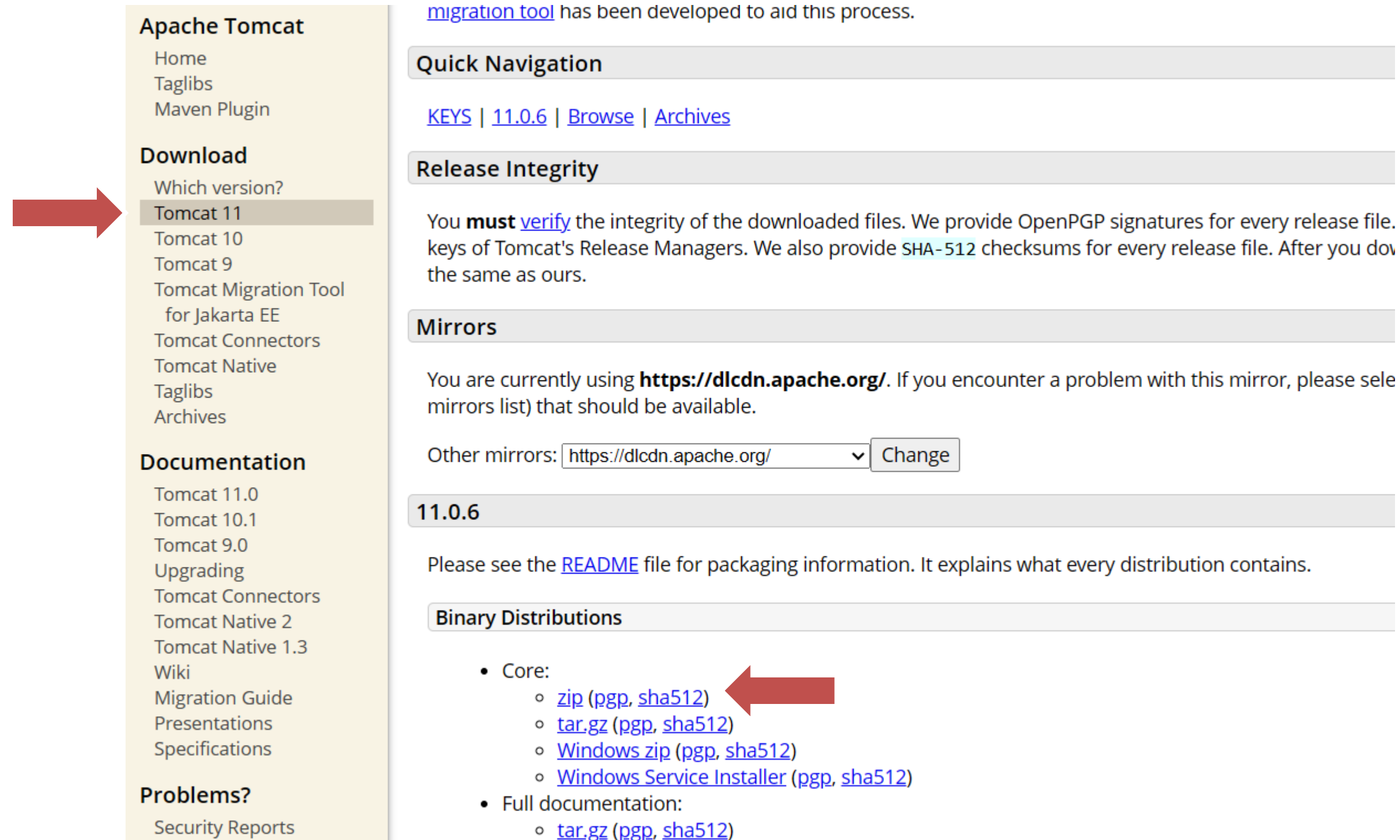


Windows x86\_64  
macOS x86\_64 | AArch64  
Linux x86\_64 | AArch64

Click [here](#) to open a bug report with the Eclipse Web Tools Platform.  
Click [here](#) to raise an issue with the Eclipse Platform.  
Click [here](#) to raise an issue with Maven integration for web projects.  
Click [here](#) to raise an issue with Eclipse Wild Web Developer (incubating).



# Download Tomcat 11



The screenshot shows the Apache Tomcat 11 download page. A red arrow points to the 'Tomcat 11' link in the 'Download' section of the left sidebar. Another red arrow points to the 'zip (pgp, sha512)' link in the 'Core' list under 'Binary Distributions'.

**Apache Tomcat**  
Home  
Taglibs  
Maven Plugin

**Download**  
Which version?  
**Tomcat 11**  
Tomcat 10  
Tomcat 9  
Tomcat Migration Tool for Jakarta EE  
Tomcat Connectors  
Tomcat Native  
Taglibs  
Archives

**Documentation**  
Tomcat 11.0  
Tomcat 10.1  
Tomcat 9.0  
Upgrading  
Tomcat Connectors  
Tomcat Native 2  
Tomcat Native 1.3  
Wiki  
Migration Guide  
Presentations  
Specifications

**Problems?**  
Security Reports

[migration tool](#) has been developed to aid this process.

**Quick Navigation**  
[KEYS](#) | [11.0.6](#) | [Browse](#) | [Archives](#)

**Release Integrity**  
You **must** [verify](#) the integrity of the downloaded files. We provide OpenPGP signatures for every release file. keys of Tomcat's Release Managers. We also provide [SHA-512](#) checksums for every release file. After you do the same as ours.

**Mirrors**  
You are currently using **<https://dlcdn.apache.org/>**. If you encounter a problem with this mirror, please select a mirror from the mirrors list) that should be available.  
Other mirrors:

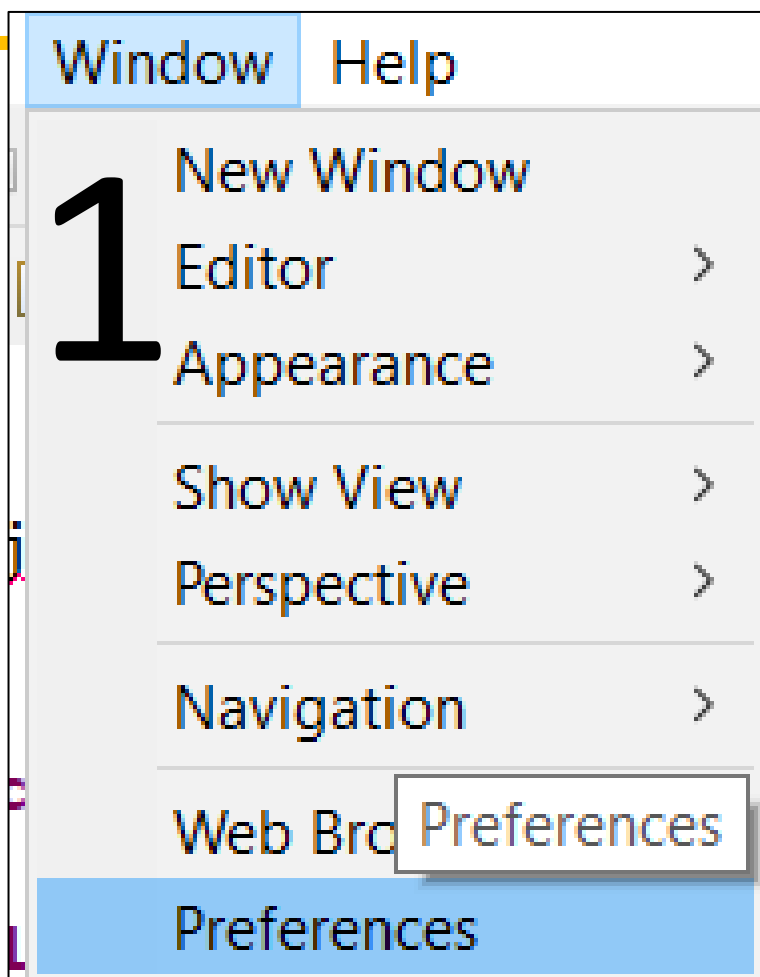
**11.0.6**  
Please see the [README](#) file for packaging information. It explains what every distribution contains.

**Binary Distributions**

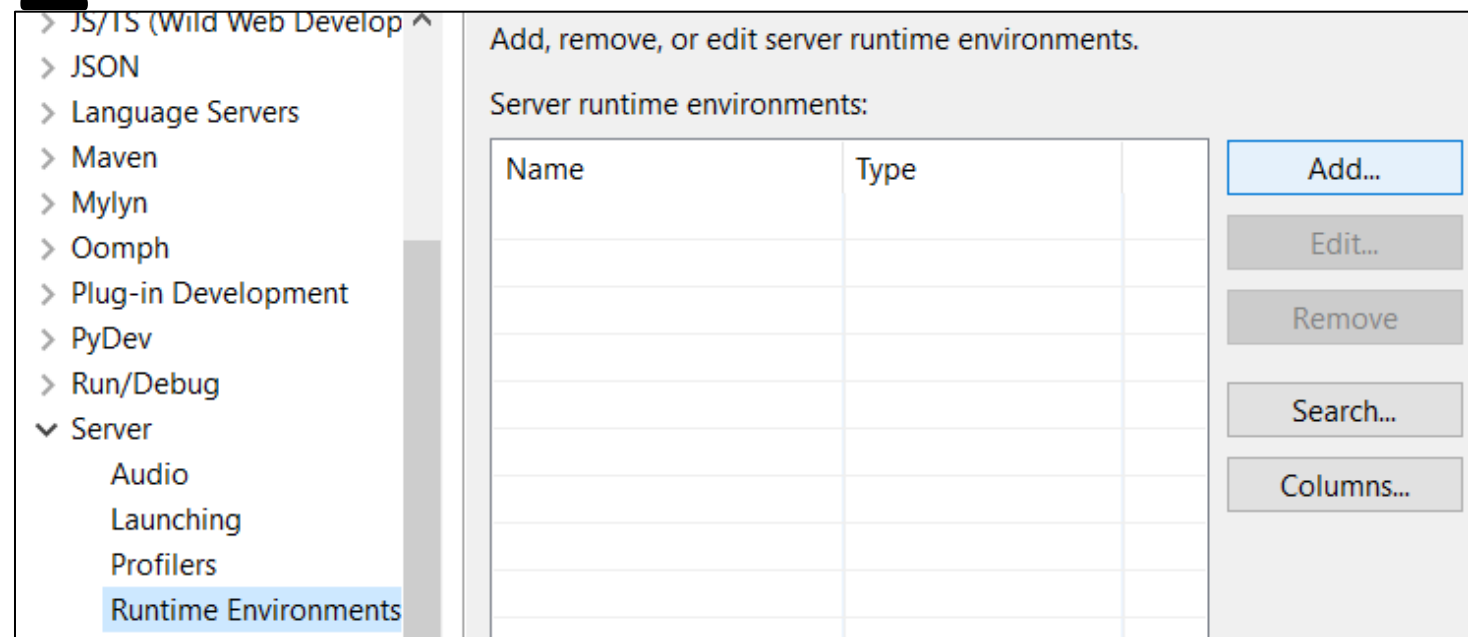
- Core:
  - [zip \(pgp, sha512\)](#)
  - [tar.gz \(pgp, sha512\)](#)
  - [Windows zip \(pgp, sha512\)](#)
  - [Windows Service Installer \(pgp, sha512\)](#)
- Full documentation:
  - [tar.gz \(pgp, sha512\)](#)



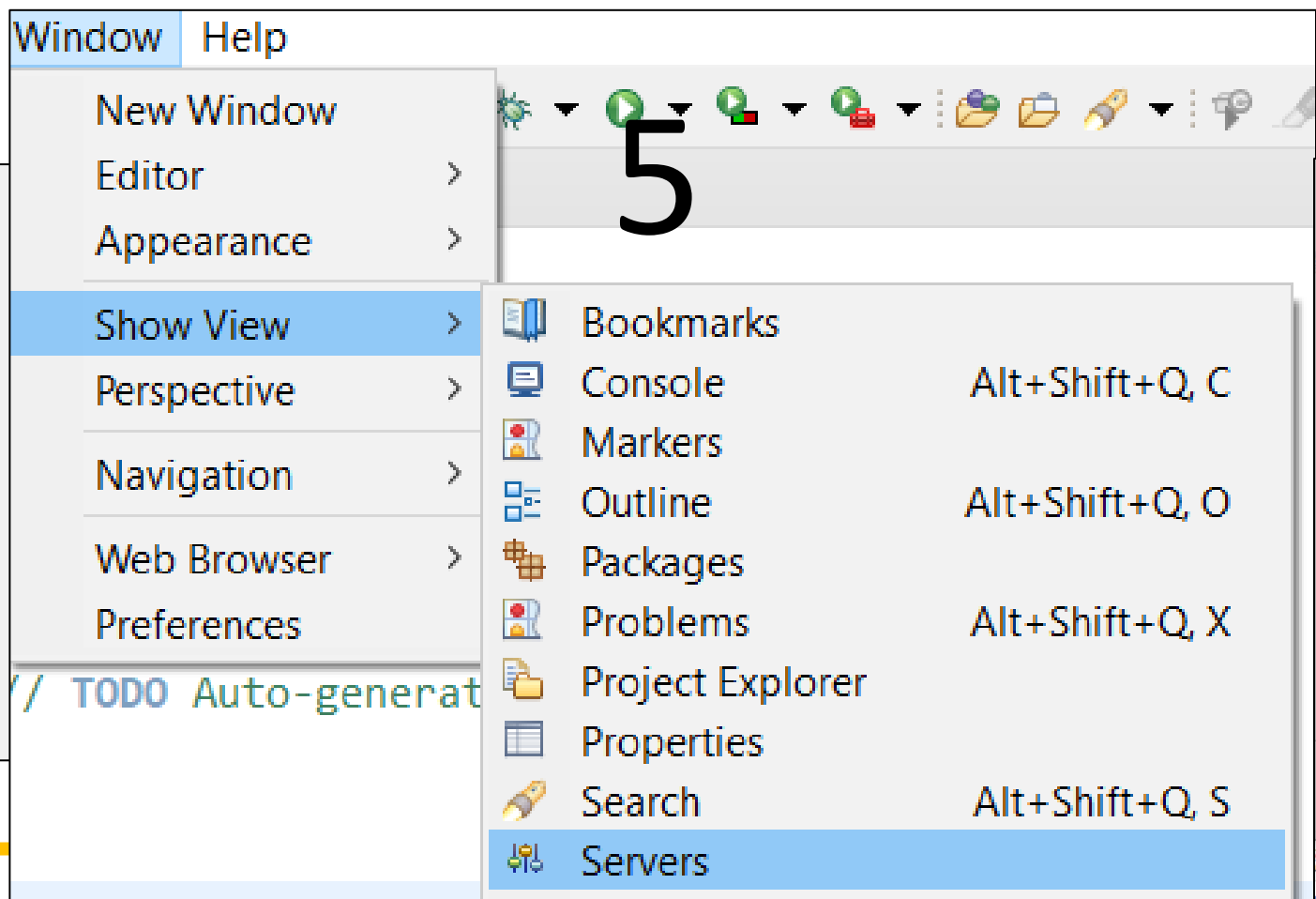
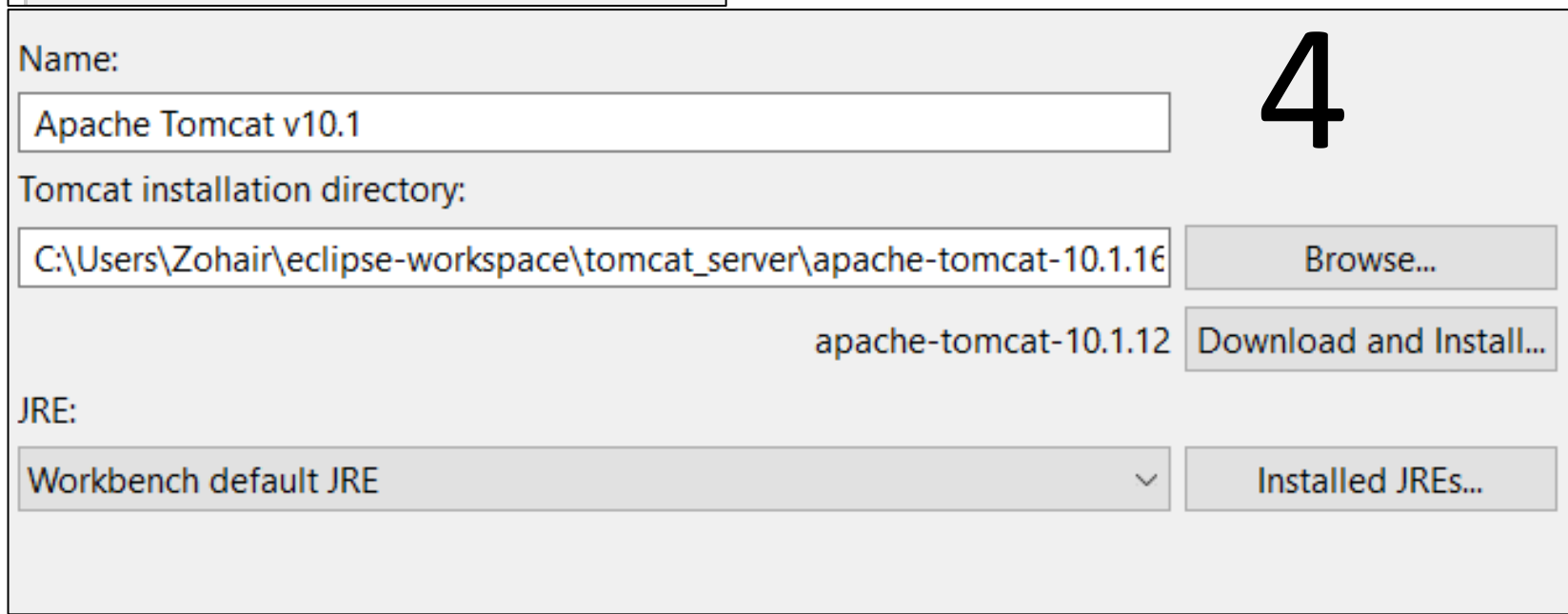
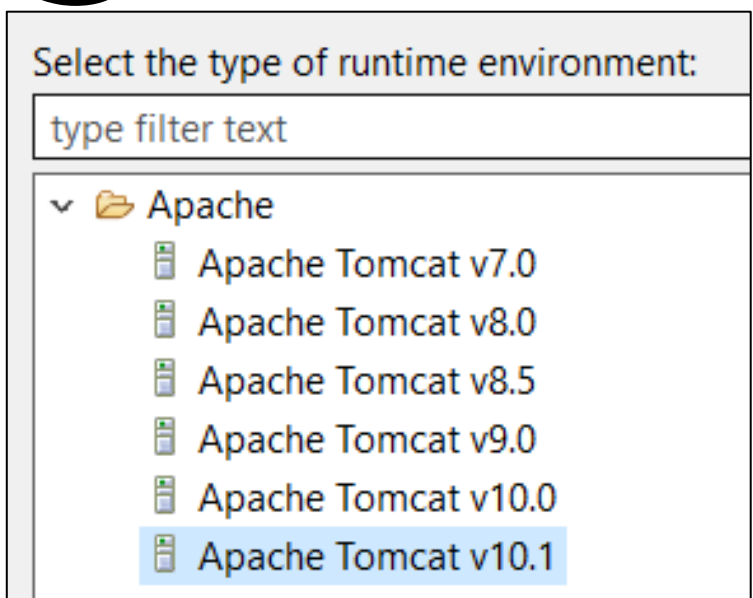
# Configure Tomcat



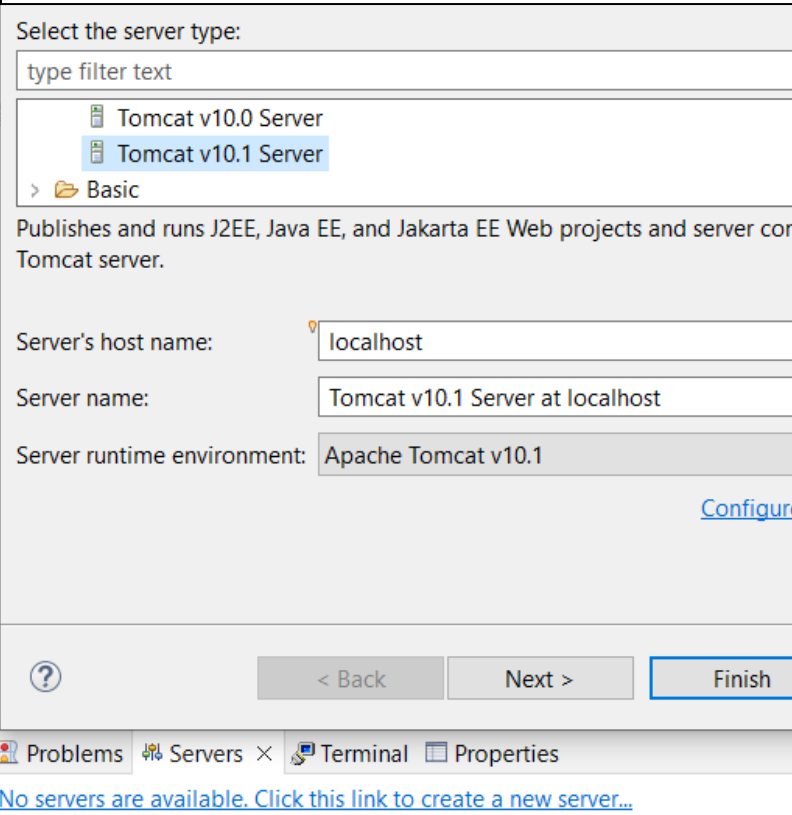
2



3



6

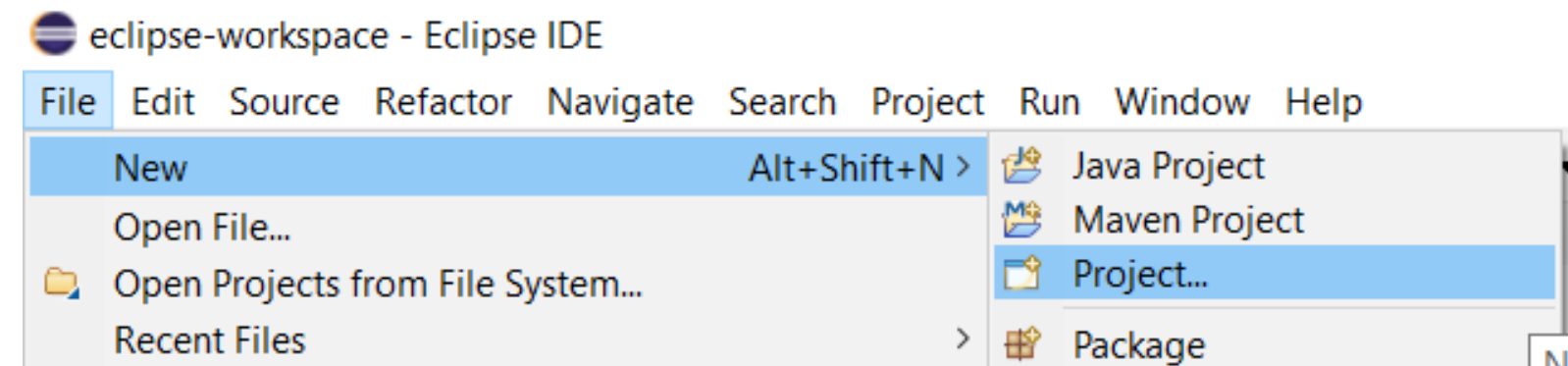




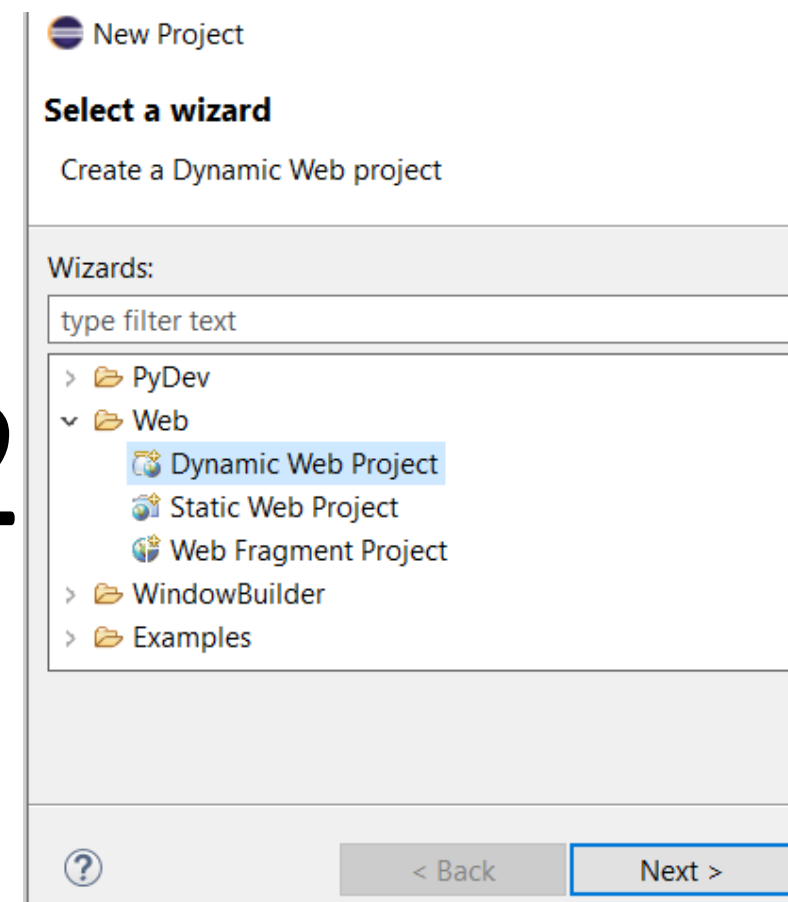
# Create Web Project in Eclipse

- Must Installed JAVA EE packages (e.g. Web etc.)

1



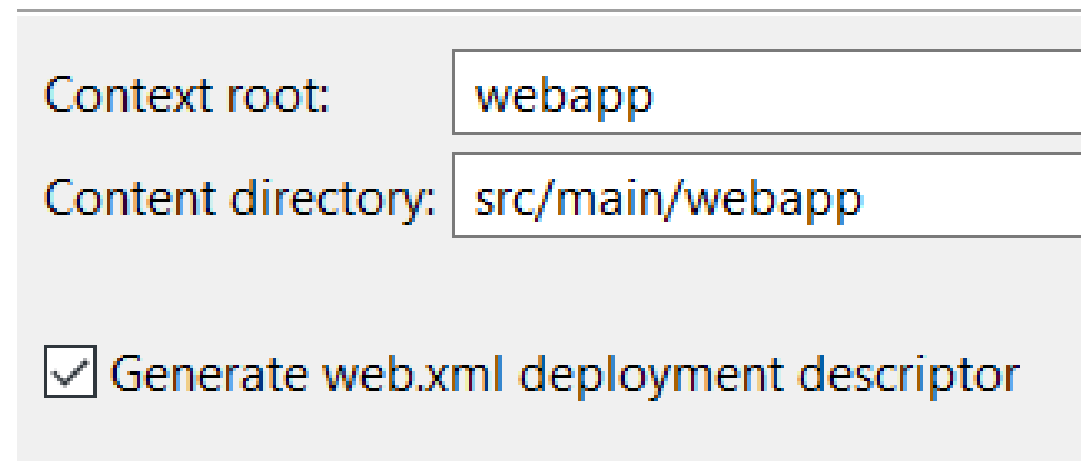
2



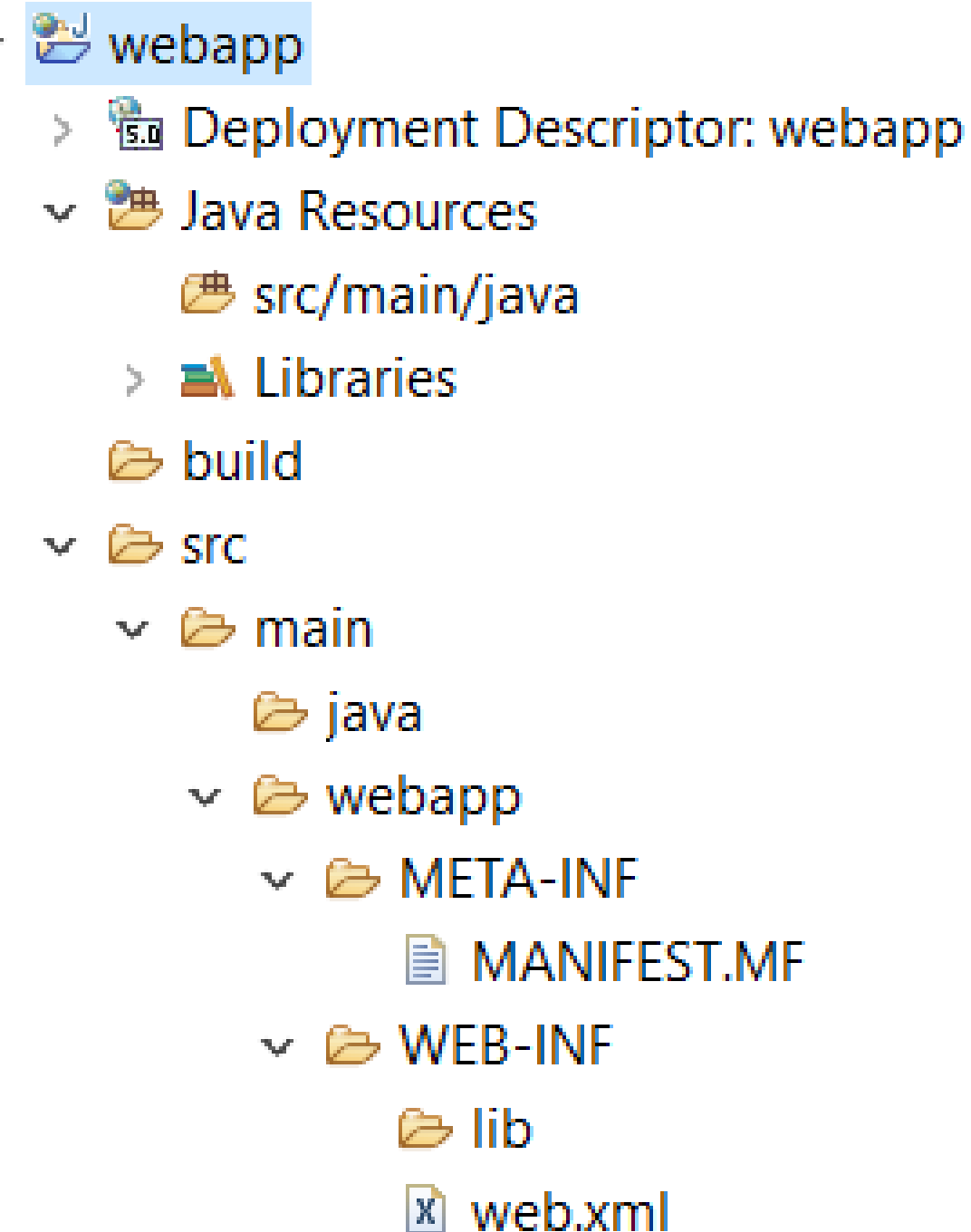
New Dynamic Web Project

## Web Module

3 Configure web module settings.



4



- After Every screen click NEXT, don't change any setting – Web Module Version should be 5.0

# Deployment Descriptor

## Deployment Descriptor File: (web.xml)

An XML file named **web.xml** used to configure a Java web application deployed to a servlet container like Apache Tomcat.

## Override Methods

- **doGet (request, response)**
  - Request: HttpServletRequest
  - Response: HttpServletResponse
- **doPost (request, response)**
  - Request: HttpServletRequest
  - Response: HttpServletResponse

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="https://jakarta.ee/xml/ns/jakartaee"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd" id="WebApp_ID"
version="5.0">
<display-name>webapp</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.jsp</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>default.html</welcome-file>
<welcome-file>default.jsp</welcome-file>
<welcome-file>default.htm</welcome-file>
</welcome-file-list>
</web-app>
```

**<!-- Define a Servlet -->**

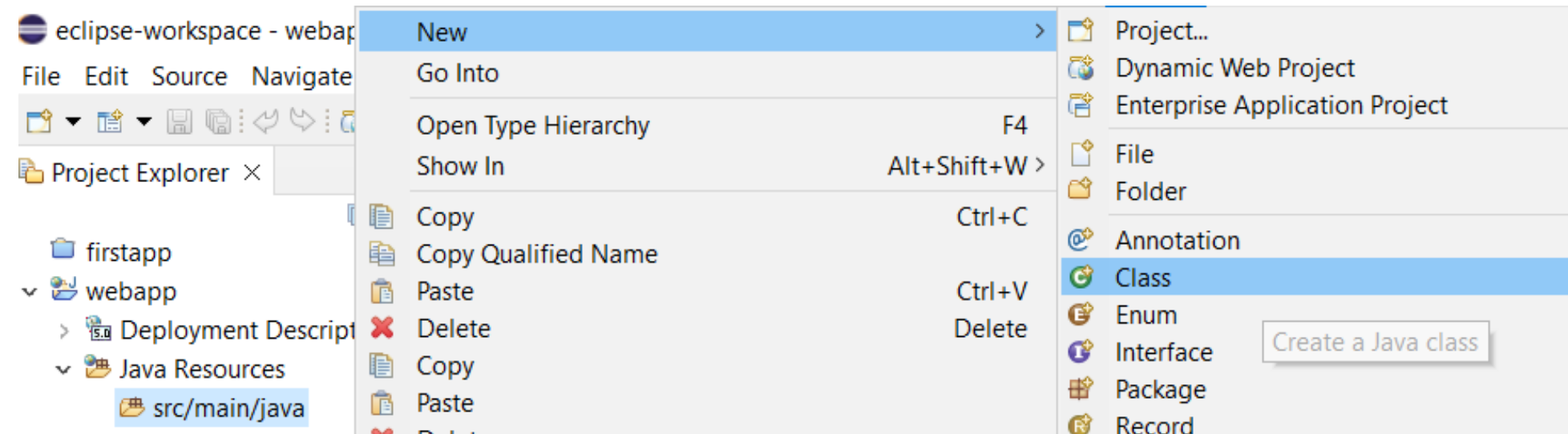
```
<servlet>
  <servlet-name>helloServlet</servlet-name>
  <servlet-class>com.example.HelloServlet</servlet-class>
</servlet>
```

**<!-- Map the Servlet to a URL pattern -->**

```
<servlet-mapping>
  <servlet-name>helloServlet</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```



# Create Servlet



Source folder: webapp/src/main/java

Package: com.java.servlet

☐ Enclosing type:

Name: MyServlet

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static  
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass: java.lang.Object

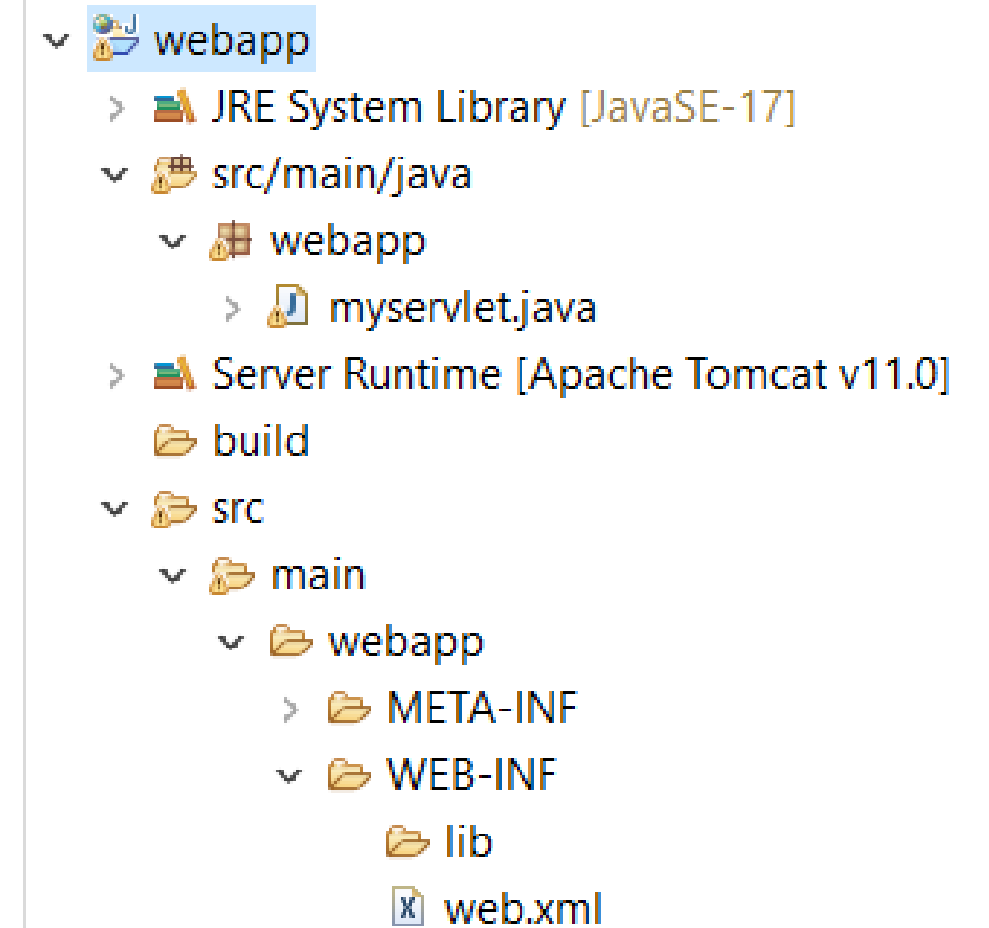
Interfaces:

Which method stubs would you like to create?  
☐ public static void main(String[] args)  
☒ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

```
import java.io.IOException;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
public class MyServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {

    }
}
```



# Update Web.xml

---

- Deployment Descriptor, Remove all code except **<web-app>** tag.

## Inside web-app tag

```
<servlet>
```

```
<servlet-name>ServletName</servlet-name>
```

```
<servlet-class>com.java.servlet.MyServlet</servlet-class>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
<servlet-name>ServletName</servlet-name>
```

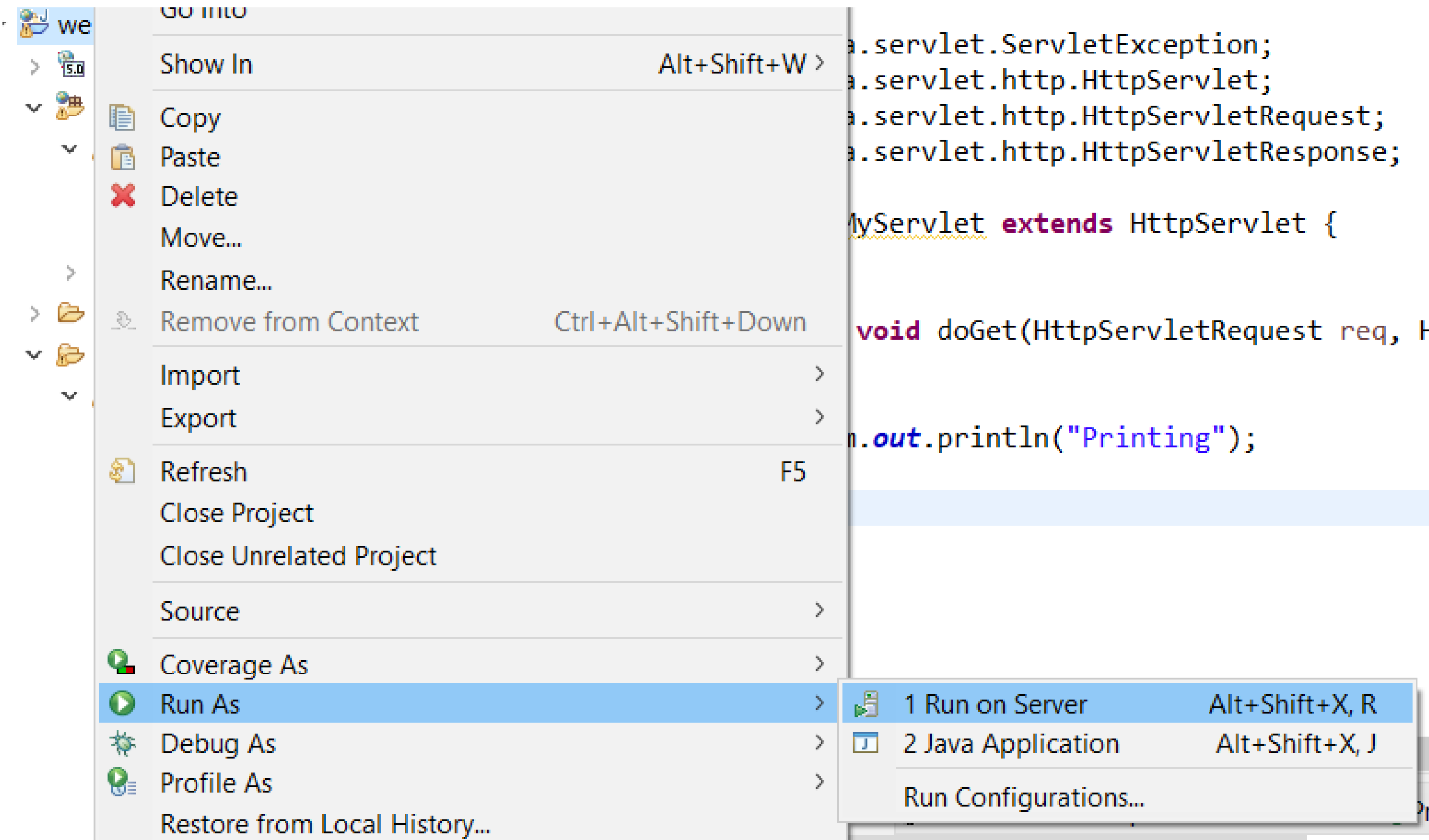
```
<url-pattern>/index</url-pattern>
```

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



# Run Server

- Localhost:8080/webapp/index



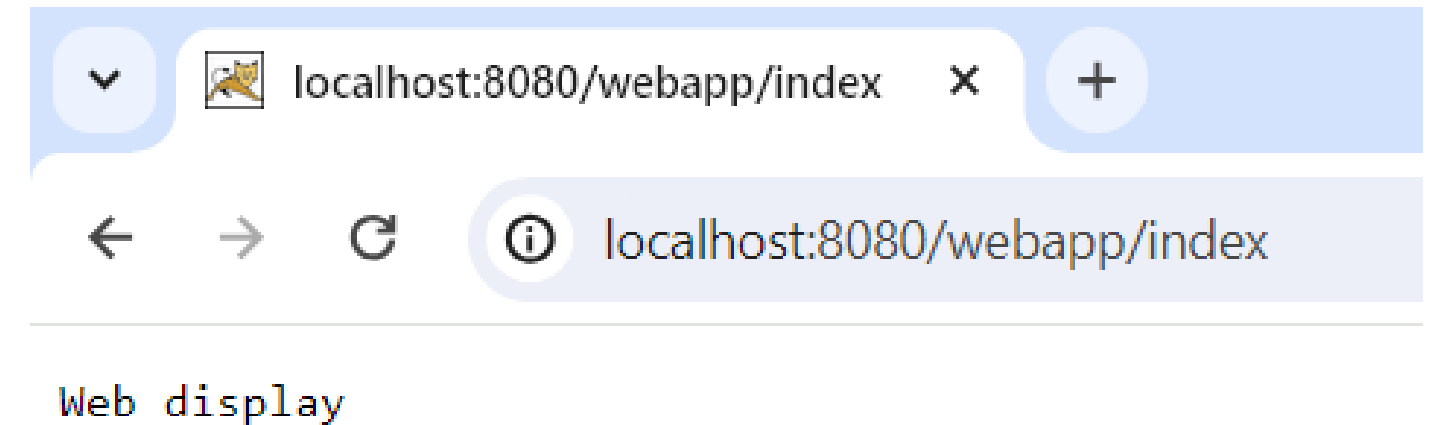
# Web Content Display

```
package com.java.servlet;

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

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
    IOException {
        System.out.println("Console display");
        PrintWriter pw = resp.getWriter();
        pw.print("Web display");
    }
}
```





# Task

---

- Create Two Servlet
  - Login
  - Register
- Create Two URL and these two URL print different Servlet Get Request



# Create Two Servlet

```
package com.java.servlet;

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

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class MyServlet1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req,
        HttpServletResponse resp) throws
        ServletException, IOException {
        System.out.println("MyServlet1 Console");
        PrintWriter pw = resp.getWriter();
        pw.print("MyServlet1 Web");
    }
}
```

```
package com.java.servlet;

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

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

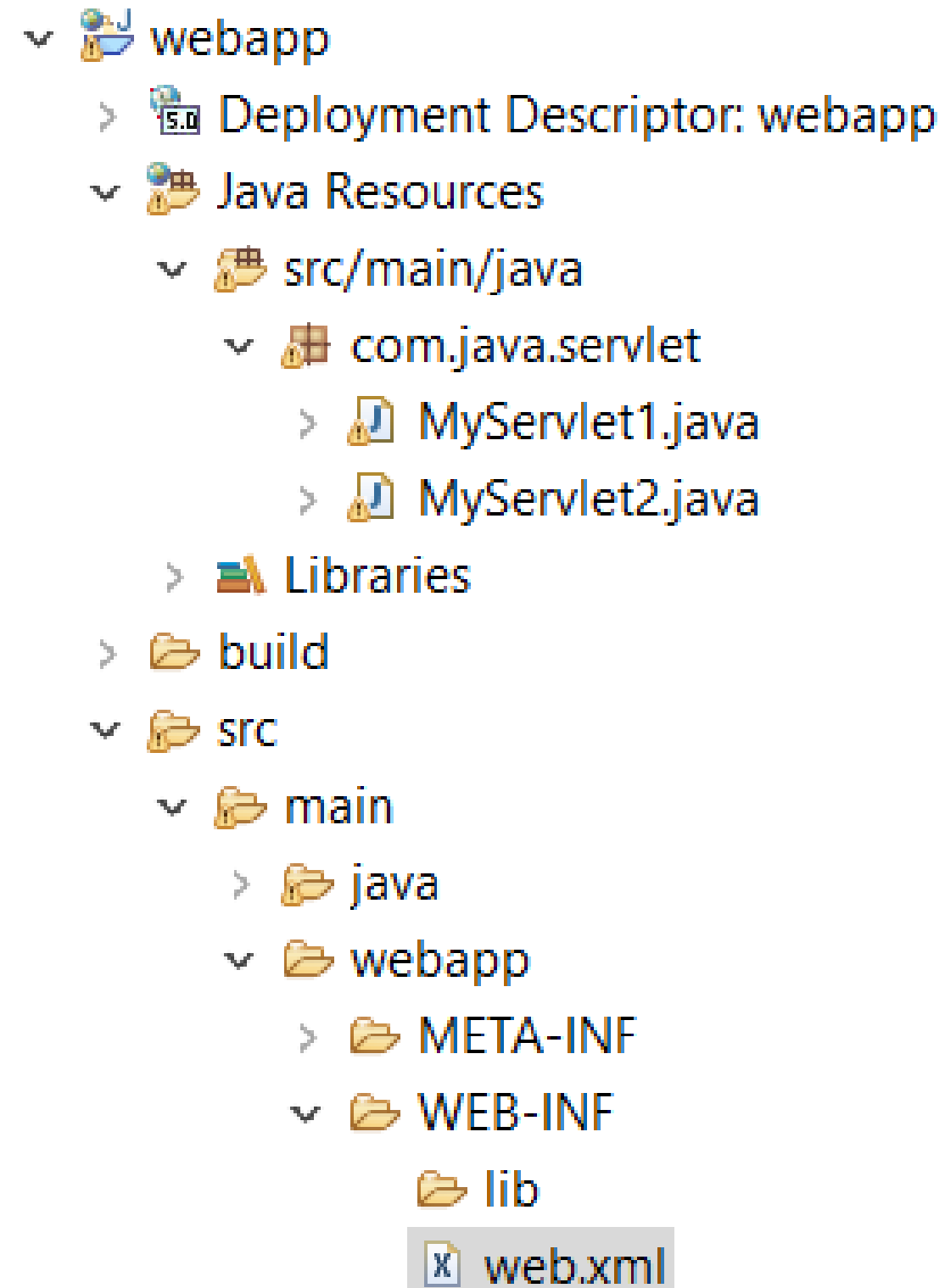
public class MyServlet2 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req,
        HttpServletResponse resp) throws ServletException,
        IOException {
        System.out.println("MyServlet2 Console");
        PrintWriter pw = resp.getWriter();
        pw.print("MyServlet2 Web");
    }
}
```



# Web.xml

```
<web-app>
<servlet>
<servlet-name>map1</servlet-name>
<servlet-class>com.java.servlet.MyServlet1</servlet-
class>
</servlet>
<servlet-mapping>
<servlet-name>map1</servlet-name>
<url-pattern>/index</url-pattern>
</servlet-mapping>
<servlet>

<servlet-name>map2</servlet-name>
<servlet-class>com.java.servlet.MyServlet2</servlet-
class>
</servlet>
<servlet-mapping>
<servlet-name>map2</servlet-name>
<url-pattern>/index2</url-pattern>
</servlet-mapping>
</web-app>
```



# HttpServletRequest, HttpServletResponse

---

- HttpServletRequest, HttpServletResponse both are interface.
- Request object responsible for **sending Requests**
- Response object responsible for **Response back**
- There are many methods for request and response pipeline.

- **HttpServletRequest**

- getParameter(String name)
- get\_cookies()
- getMethod()
- getSession(Boolean )

- **HttpServletResponse**

- getwriter()
- sendRedirect(String Location)
- sendError(int code, String Message)
- And many more.....



# Servlet Life Cycle

---

- Loading and Instantiation
  - When server is started, servlet class is loading in the memory and servlet object is created.
- Initialization
  - Servlet object will be initialized by invoking **init()** method.
- Request Handling
  - It will handle or serve the client request by invoking **service()** method. It use threads by multiple request.
- Destroy
  - When the server is shutdown **destroy()** method will be executed and servlet object will be deleted

# Servlet Annotation

- No Need of web.xml in latest version: The version 3.0 or above

- @webServlet

- @webFilter

- @webListener

- @webMultipartConfig

```
package com.java.servlet;
```

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

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

```
import jakarta.servlet.ServletException;
```

```
import jakarta.servlet.http.HttpServlet;
```

```
import jakarta.servlet.http.HttpServletRequest;
```

```
import jakarta.servlet.http.HttpServletResponse;
```

```
import jakarta.servlet.annotation.WebServlet;
```

```
@WebServlet("/link")
```

```
public class MyServlet3_annot extends HttpServlet {
```

```
@Override
```

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
```

```
throws ServletException, IOException {
```

```
//System.out.println("MyServlet2 Console");
```

```
PrintWriter pw = resp.getWriter();
```

```
pw.print("MyServlet3 using Web Annot. No web.xml");
```

```
}
```

```
}
```





# HTML Form Post and Servlet Print on Web

## Student Info Form

Name:

Class:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Student Info Form</title>
</head>
<body>
<div style="text-align:center;">
<h2>Student Info Form</h2>
<form action="submitform" method="post">
Name: <input type="text" name="std_name" /> <br /><br />
Class: <input type="text" name="std_class" /> <br /><br />
<input type="submit" value="Save Records" /> <br />
</form>
</div>
</body>
</html>
```

```
package com.java.servlet;
```

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

```
@WebServlet("/submitform")
public class stdinfo_servlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req,
        HttpServletResponse resp) throws ServletException, IOException {
        String std_name = req.getParameter("std_name");
        String std_class = req.getParameter("std_class");
        PrintWriter pw = resp.getWriter();
        pw.print("Student Name: " + std_name);
        pw.print("Student Class: " + std_class);
    }
}
```



# Send Redirect Method | Request Dispatcher | Include Vs. Forward

---



# Send Redirect()

- HttpServletResponse → **sendredirect()**;
- In HTML form has one textbox for searching and direct search in Google using query String
- Also, **sendredirect** change the URL for external Redirection.

```
@WebServlet("/submitForm")
public class MyServlet extends HttpServlet
{
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
    {
        String mysearch = req.getParameter("search1");

        //resp.sendRedirect("https://www.google.com");
        resp.sendRedirect("https://www.google.com/search?q="+mysearch);
    }
}
```

# Request Dispatcher

---

- The **RequestDispatcher** is an interface in Java that provides mechanisms for forwarding a request to another resource (such as a servlet, JSP, or HTML page) or including content from another resource in the current request's response.
- It is used in Java Servlets for controlling the flow of requests within the server.
- There are two main methods in the **RequestDispatcher** interface that facilitate this behavior:
  - **forward()**: Ends the current processing, hands control to another resource, and the response is generated by that resource.
  - **include()**: Adds the output of another resource to the current response, and processing continues in the current resource.

# Request Dispatcher

---

- The **RequestDispatcher** is an interface in Java that provides mechanisms for forwarding a request to another resource (such as a servlet, JSP, or HTML page) or including content from another resource in the current request's response.
- It is used in Java Servlets for controlling the flow of requests within the server.
- There are two main methods in the **RequestDispatcher** interface that facilitate this behavior:
  - **forward()**: Ends the current processing, hands control to another resource, and the response is generated by that resource.
  - **include()**: Adds the output of another resource to the current response, and processing continues in the current resource.

# Request Dispatcher

---

## Forward (forward())

- One-way transfer of control to another resource
- Original URL in browser remains unchanged
- Previous output is cleared/discarded
- Control never returns to calling servlet

```
RequestDispatcher rd = request.getRequestDispatcher("target.html");  
rd.forward(request, response);
```





# Request Dispatcher

---

## Include (include())

- Includes content of another resource in current response
- Original servlet maintains control
- Previous output is preserved
- Control returns to calling servlet after inclusion

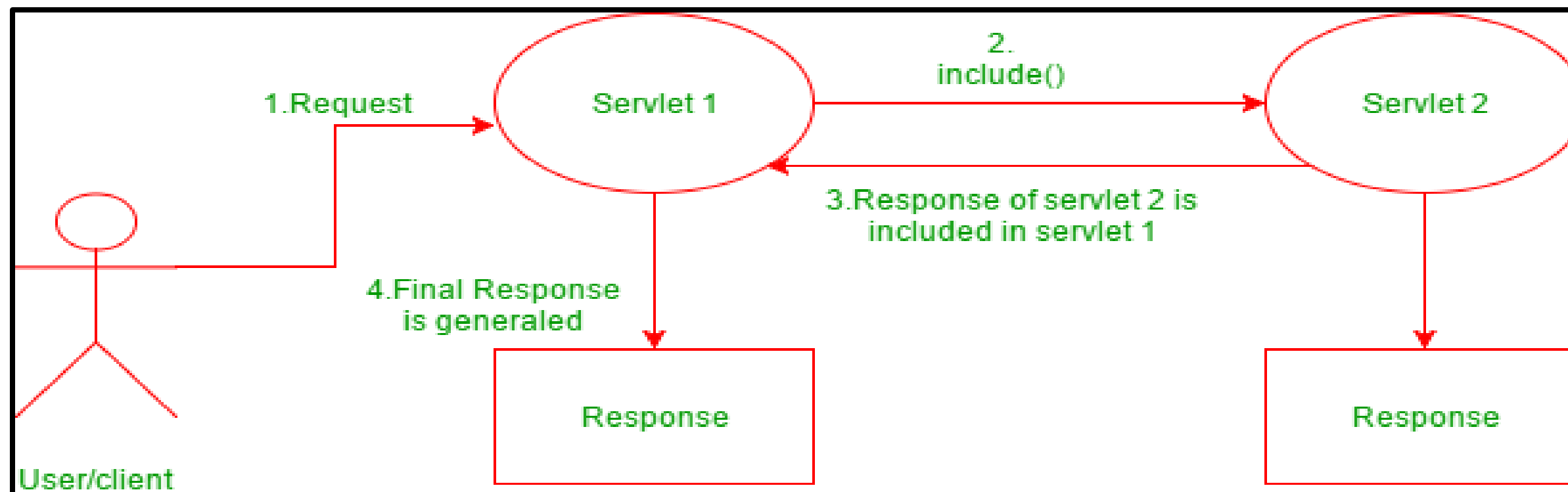
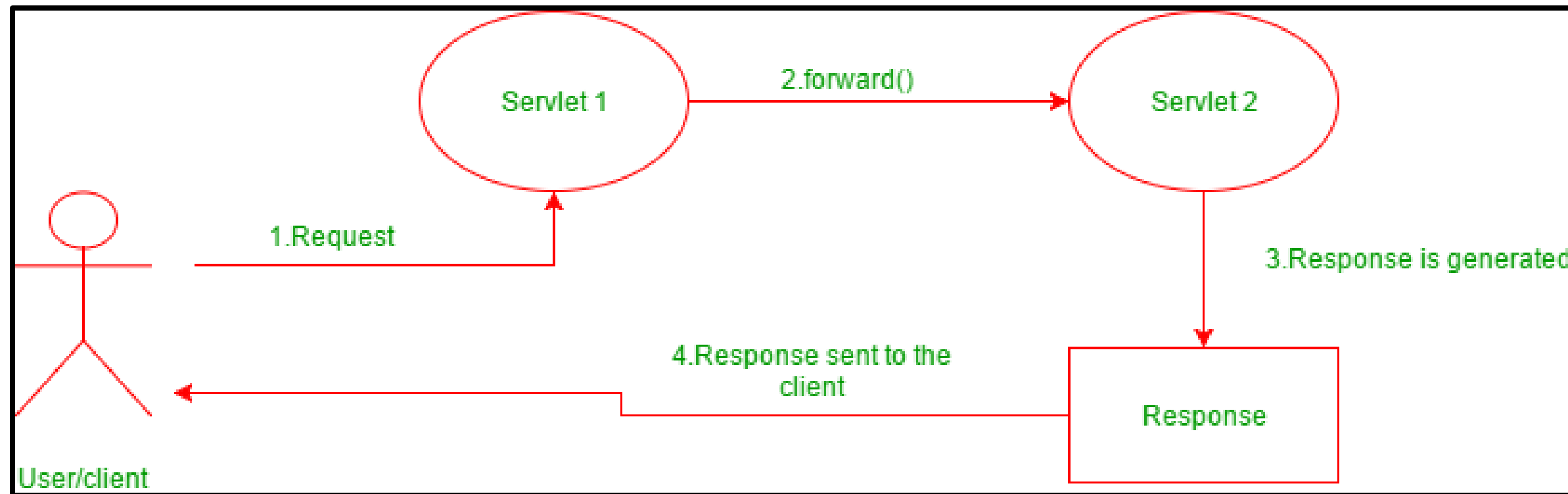
```
RequestDispatcher rd = request.getRequestDispatcher("header.html");
```

```
rd.include(request, response);
```

```
// Continue processing in current servlet
```



# Request Dispatcher



# Request Dispatcher – Go to Welcome Page if Login

- `RequestDispatcher rd = req.getRequestDispatcher("/welcome.html");`
- `rd.forward(req, resp);`
- It cannot change the URL

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Login Form</title>
</head>
<body>
<div style="text-align:center;">
<h2>Student Login Form</h2>
<form action="submitloginform" method="post">
User Name: <input type="text" name="user_name" />
Password: <input type="text" name="user_password" />
<input type="submit" value="Login" />
</form>
</div>
</body>
</html>
```

```
@WebServlet("/submitloginform")
public class login extends HttpServlet {

@Override
protected void service(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {

String user_name = req.getParameter("user_name");
String user_password = req.getParameter("user_password");

if (user_name.equals("zohair") && user_password.equals("123"))
{
RequestDispatcher rd = req.getRequestDispatcher("/welcome.html");
rd.forward(req, resp);
}}
}
```



# Request Dispatcher – Go to Error Page if Invalid Login

```
if (user_name.equals("zohair") && user_password.equals("123")) {  
    RequestDispatcher rd = req.getRequestDispatcher("/welcome.html");  
    rd.forward(req, resp);  
}  
else {  
    resp.setContentType("text/html");  
    PrintWriter pw = resp.getWriter();  
    pw.print("<h3 style='color:red;'>Invalid UserName and Password<h3>");  
    RequestDispatcher rd = req.getRequestDispatcher("/login.html");  
    rd.include(req, resp);  
}
```

If not set print html source code

Use Include for response page and  
print message

# Output- URL Not Changing (Login/Invalid)

localhost:8080/webapp/login.html

## Student Login Form

User Name:

Password:

Login

localhost:8080/webapp/submitloginform

## Welcome to the System

localhost:8080/webapp/login.html

## Student Login Form

User Name:

Password:

Login

localhost:8080/webapp/submitloginform

## Invalid UserName and Password

## Student Login Form

User Name:

Password:

Login

webapp

- Deployment Descriptor: webapp
- Java Resources
  - src/main/java
    - com.java.servlet
      - login.java
      - MyServlet1.java
      - MyServlet2.java
      - MyServlet3\_annot.java
      - stdinfo\_servlet.java
  - Libraries
    - JRE System Library [JavaSE-20]
    - Server Runtime [Apache Tomcat v10.1]
  - build
  - src
    - main
      - java
      - webapp
        - META-INF
        - WEB-INF
        - index.html
        - login.html
        - stdinfo.html
        - welcome.html

# Task- Must be Submitted within class on LMS- Otherwise not Considered – 1 Hour

- Create Two Html Pages with Bootstrap
  - Login
  - Profile


### LOGIN FORM

  
  
LOG IN

Invalid username or password.  
Please try again.

### LOGIN FORM

  
  
LOG IN






  

Current Data & Time

## Your Name

Student ID/RollNo

Your Detailed Bio/Intro

Message Contact