Lab Guide: Creating "HelloServlet" in Eclipse J2EE

Contact | Sponsor

Le Ngoc Thach	
(+084) 0908 550 642	facebook.com/ThachLN
Website: https://xmyworkspace.com/learn/sponsor	

1. Introduction

This lab covers the creation of a Java Web Application using **Servlets** and **Apache Tomcat** within the **Eclipse JEE IDE**.

Goal

- Create a "HelloServlet" application.
- Display the message: **"Become a Java Web Application Developer"**.
- Understand web.xml and the Tomcat deployment process.

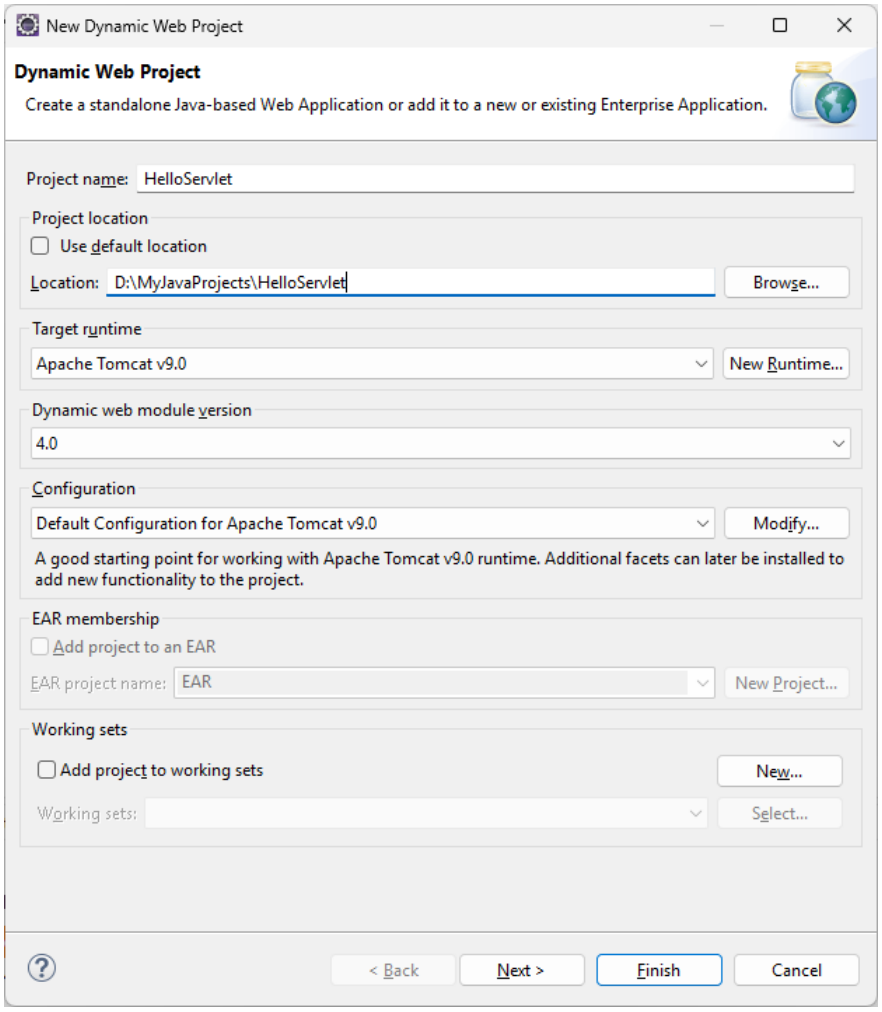
---

2. Step-by-Step Implementation in Eclipse

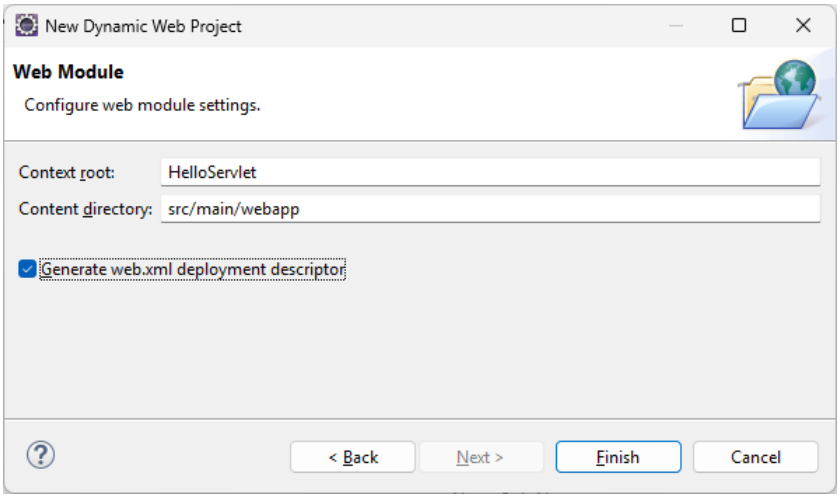
Step 1: Create a Dynamic Web Project

1. Open **Eclipse IDE for Enterprise Java**.
2. Go to **File > New > Dynamic Web Project**.

3. **Project Name:** HelloServlet.
4. **Uncheck** Use default location
5. **Location:** D:\MyJavaProjects\HelloServlet
6. **Target Runtime:** Select **Apache Tomcat (v9.0 or later)**.



7. Click **Next** until you see the "Web Module" page.

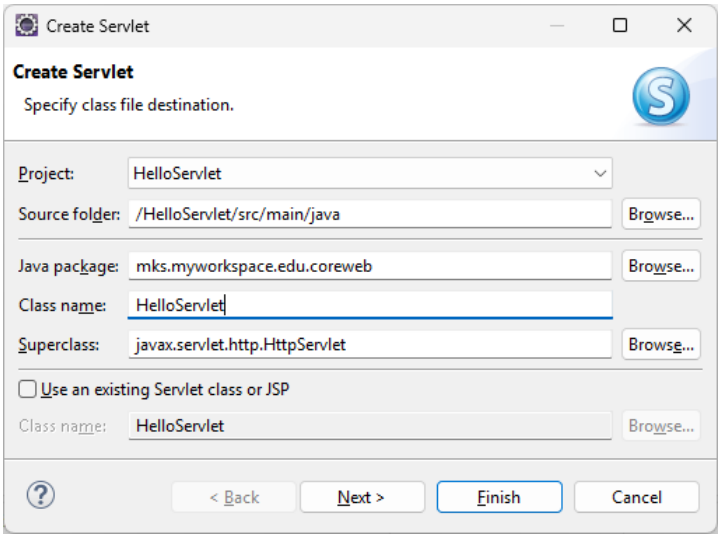


8. **Crucial:** Check the box **"Generate web.xml deployment descriptor"**.

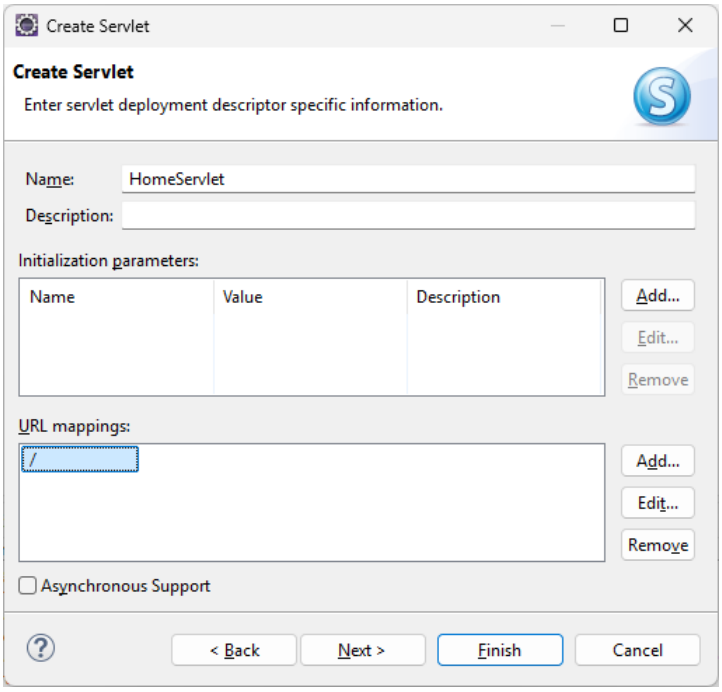
9. Click **Finish**

Step 2: Create the Servlet

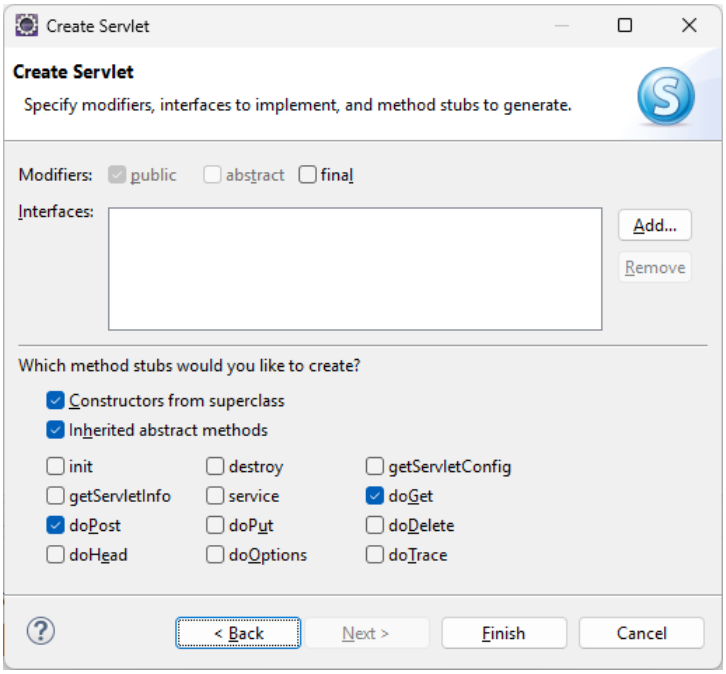
1. Right-click the Java Resources > src/main/java folder.
2. Select **New > Servlet**.
3. **Java package:** mks.myworkspace.edu.coreweb
4. **Class name:** HomeServlet



5. Click **Next**. Ensure the **URL mapping** is set to **/**.



6. Click **Next**



7. Click **Finish**.

Step 3: Write the Logic

Locate the doGet method in HomeServlet.java and update it as follows:

Java

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

    // Set response content type
    response.setContentType("text/html");

    // Write the output
    var out = response.getWriter();
    out.println("<html><body>");
    out.println("<h1>Become a Java Web Application
Developer</h1>");
    out.println("</body></html>");
}
```

3. Run the project

1) Right-click the project HelloServlet.
2) Select **Run As > Run on Server**.
3) Choose your Tomcat server and click **Finish**.

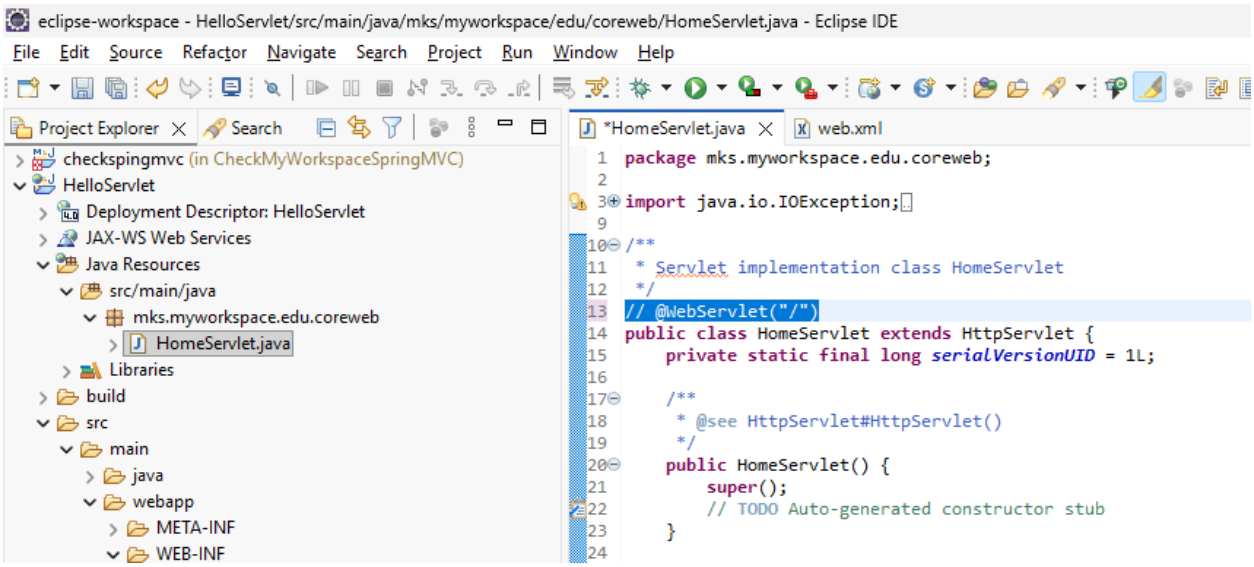
The default browser will be opened with link http://localhost:8080/HelloServlet/

Open your browser to this URL if Eclipse don't open it.

4. Change somethings

Step 1: Clean the Java File

Open HomeServlet.java. Ensure there is NO line starting with @WebServlet. If it's there, delete or comment it.



Step 2: Configure web.xml

Open **src/main/webapp/WEB-INF/web.xml**. Paste the <servlet> and <servlet-mapping> blocks (shown in section 1 above) before the </web-app> closing tag.

The completed web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
  <display-name>HelloServlet</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>index.xhtml</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.jsp</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.xhtml</welcome-file>
  </welcome-file-list>
```

```
    <servlet>
        <servlet-name>MyHomeServlet</servlet-name>
        <servlet-class>
         mks.myworkspace.edu.coreweb.HomeServlet
        </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>MyHomeServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

5. Technical Concepts

Understanding web.xml (Deployment Descriptor)

The web.xml file acts as the "map" for Tomcat.

- **<servlet>**: Tells Tomcat where the compiled Java class is located.
- **<servlet-mapping>**: Links the Java class to a specific URL (e.g., /).

How Tomcat Works

Tomcat is a **Servlet Container**. It performs three main tasks:

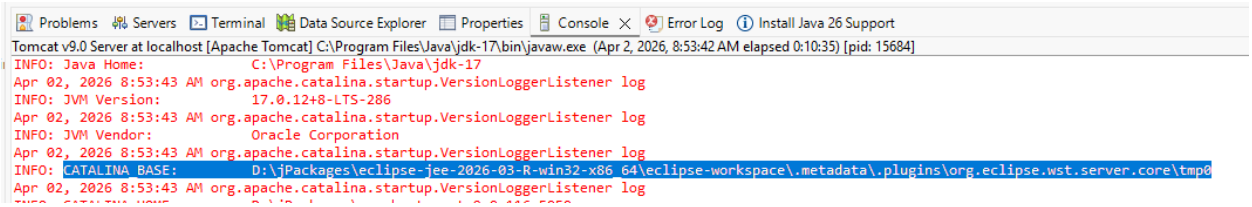
1) **Lifecycle Management:** It loads the Servlet class and calls init(), service(), and destroy().
2) **Multithreading:** It handles multiple user requests simultaneously.
3) **Request Mapping:** It reads web.xml to decide which Servlet should handle a specific URL.

How Eclipse and Integrated Tomcat Work Together

1) **The Deployment Process (Run As > Run on Server)**

When you right-click your project and select **Run on Server**, Eclipse does not run the code directly from your workspace. Instead, it creates a "shadow" copy in a temporary directory.

Monitor the **Console** in the bottom panel. In the first 10 lines of the log, you will see a path labeled CATALINA_BASE. It looks like this: INFO: CATALINA_BASE: D:\jPackages\...\eclipse-workspace\.metadata\.plugins\...\tmp0



- **CATALINA_BASE:** This is the specific folder where Eclipse deploys and runs your projects.
- **tmp0:** The last folder is usually tmp0. If a previous run fails or multiple servers are defined, Eclipse may use tmp1 or tmp2.

2) Exploring the Deployment Package

If you navigate to tmp0\wtpwebapps\HelloServlet, you will see the **Exploded Archive** (the unpacked version of your web app). This is what Tomcat actually "sees":

```
HelloServlet/ (Root)
└── WEB-INF/
    ├── web.xml (The "Map" for the server)
    ├── classes/ (Compiled .class files)
    │   └── mks/myworkspace/edu/coreweb/HomeServlet.class
    └── lib/ (External .jar libraries)
```

3) Request Processing Workflow

When a browser sends a **GET** request to http://localhost:8080/HelloServlet/, Tomcat follows these steps:

1. **Context Lookup:** Tomcat looks at the URI (/HelloServlet) and matches it to the folder HelloServlet inside tmp0\wtpwebapps\.
2. **Configuration Check:** Tomcat reads \WEB-INF\web.xml within that folder to find instructions.

3. **Servlet Mapping:** It looks at the <servlet-mapping> entries. It finds that the URL pattern / belongs to the servlet named MyHomeServlet.
4. **Class Invocation:** It finds the definition for MyHomeServlet and identifies the class: mks.myworkspace.edu.coreweb.HomeServlet.
5. **Method Execution:** Since the browser sent a **GET** request, Tomcat invokes the doGet() method of that class.

---

4) Generating and Rendering the Response

Inside your doGet() method, you define how the server talks back to the browser:

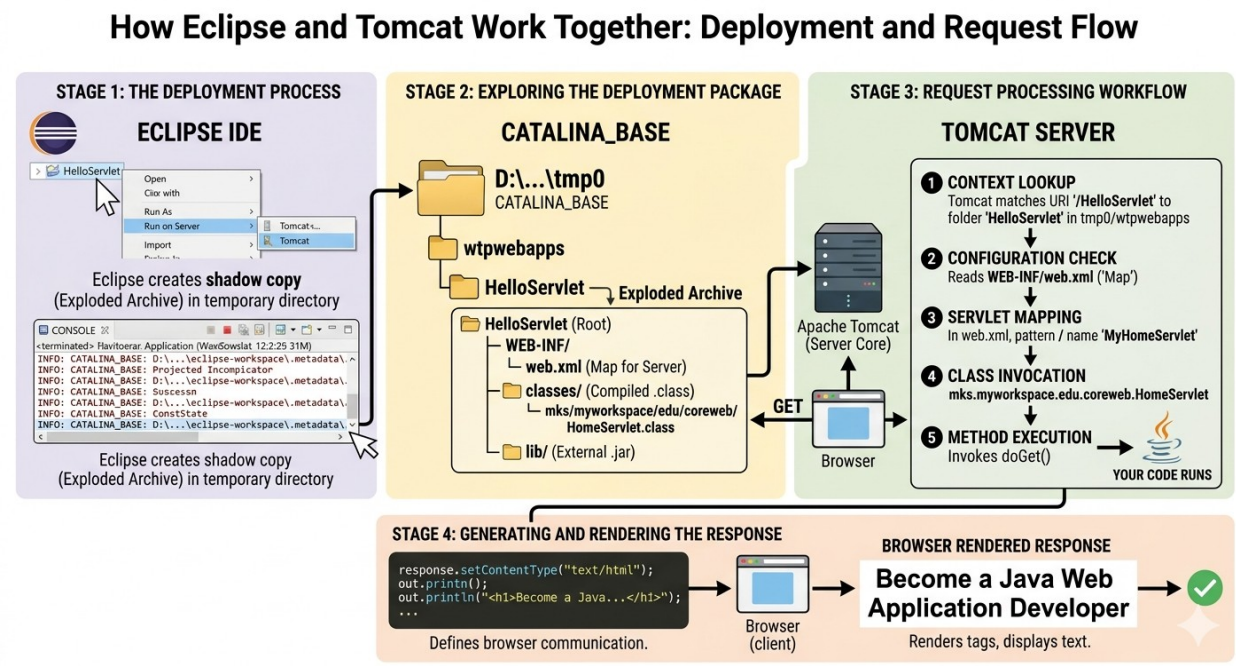
```
// 1. Tell the browser to expect HTML
response.setContentType("text/html");

// 2. Get the "Writer" to send text data
var out = response.getWriter();

// 3. Send the HTML structure
out.println("<html><body>");
out.println("<h1>Become a Java Web Application
Developer</h1>");
out.println("</body></html>");
```

What the Browser Does: When the browser receives this plain text, it sees the text/html header. It then **renders** the tags:

- It hides the <html> and <body> tags.
- It displays the text inside <h1> as a large, bold heading on the screen.

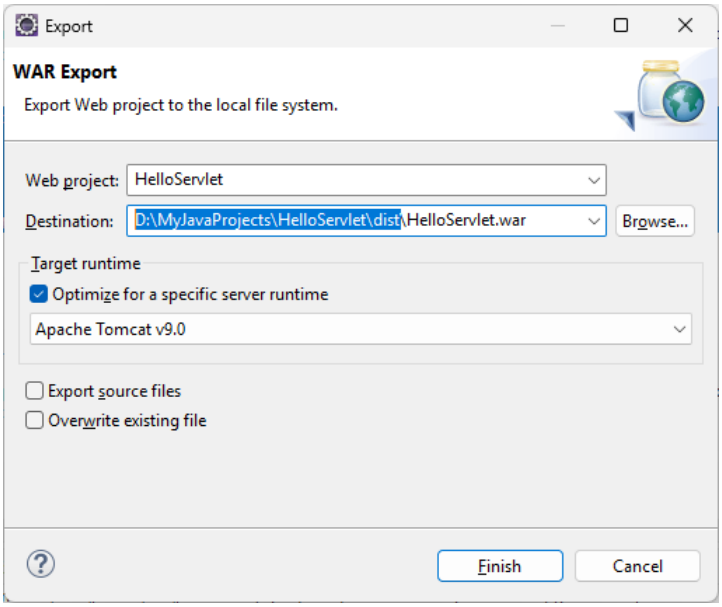


6. Deployment and Execution

Exporting to .WAR

1) Right-click the project folder.
2) Select **Export > WAR file**.
3) Choose your destination folder and click **Finish**.

   Destination: D:\MyJavaProjects\HelloServlet\dist\HelloServlet.war



4) **Note:** This .war file can be dropped into any Tomcat webapps folder globally to deploy the app.

Deploying to a Standalone Testing Tomcat

1. Download and Prepare Tomcat

1. **Download:** Visit tomcat.apache.org and download the **version 9+** binary distribution (the .zip or .tar.gz package).
2. **Extract:** Unzip the files into your designated package directory, for example: D:\jPackages\apache-tomcat-9.0.116.
3. **Rename for Clarity:** To easily identify this specific instance, rename the folder to reflect its configuration, such as: D:\jPackages\apache-tomcat-9.0.116-9090.

2. Configure Ports to Avoid Conflicts

Since Eclipse usually occupies port **8080**, you should change your standalone server to port **9090**.

1. Open the configuration file: D:\jPackages\apache-tomcat-9.0.116-9090\conf\server.xml.
2. **Shutdown Port:** Change the Server Shutdown port from 8005 to **9005**.
3. **HTTP Connector Port:** Locate the <Connector> tag and change the port from 8080 to **9090**.

3. Deploy the .WAR File

1. **(Optional) Clean the Server:** For a fresh start, delete all existing folders (like ROOT, examples, docs) inside the \webapps\ directory.

2. **Copy your application:** Locate your generated .war file (e.g., D:\MyJavaProjects\HelloServlet\dist\HelloServlet.war) and paste it into:

D:\jPackages\apache-tomcat-9.0.116-9090\webapps

4. Start the Server and Verify

1. **Launch:** Navigate to the \bin folder and run **startup.bat** (Windows) or startup.sh (Linux/Mac). A new console window will appear showing the server logs.
2. **Automatic Extraction:** Tomcat will detect the .war file, automatically create a folder named HelloServlet, and deploy the application.
3. **Access the App:** Open your browser and navigate to:

http://localhost:9090/HelloServlet/

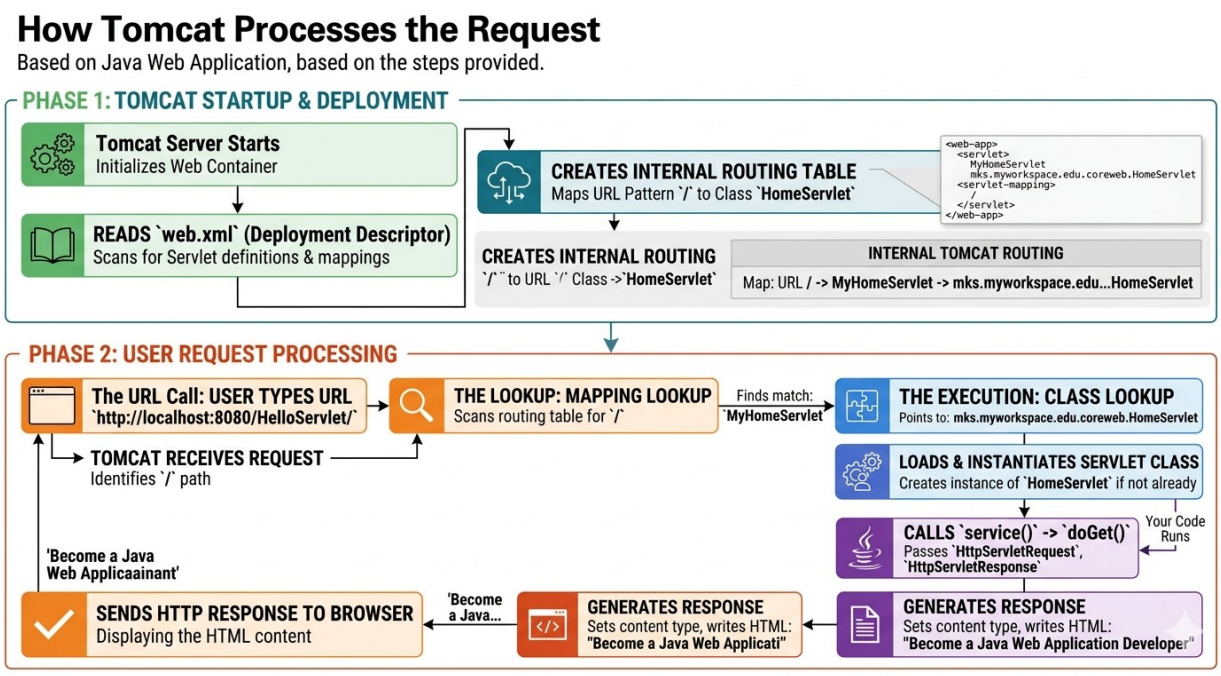
Understanding the Difference: Development vs. Testing

Feature	Development Tomcat (Eclipse)	Testing Tomcat (Standalone)
Control	Controlled by Eclipse IDE.	Controlled via startup.bat / shutdown.bat.
Location	Uses the tmp0 metadata folder.	Uses its own webapps folder.
Purpose	Rapid coding, debugging, and "Hot Swap".	Final verification of the .war package before production.
Port	Usually 8080.	Usually 9090 (to avoid conflicts).

How Tomcat Processes the Request

What happens when you click "Run":

1. **Deployment:** Tomcat reads web.xml. It sees there is a servlet named MyHomeServlet located at mks.myworkspace.edu.coreweb.HomeServlet.
2. **The URL Call:** You type http://localhost:8080/HelloServlet/ in your browser.
3. **The Lookup:** Tomcat looks at all <servlet-mapping> entries. It finds that / belongs to MyHomeServlet.
4. **The Execution:** Tomcat looks up MyHomeServlet in the definitions, finds the Java class mks.myworkspace.edu.coreweb.HomeServlet, creates an instance of it, and runs your doGet() method.



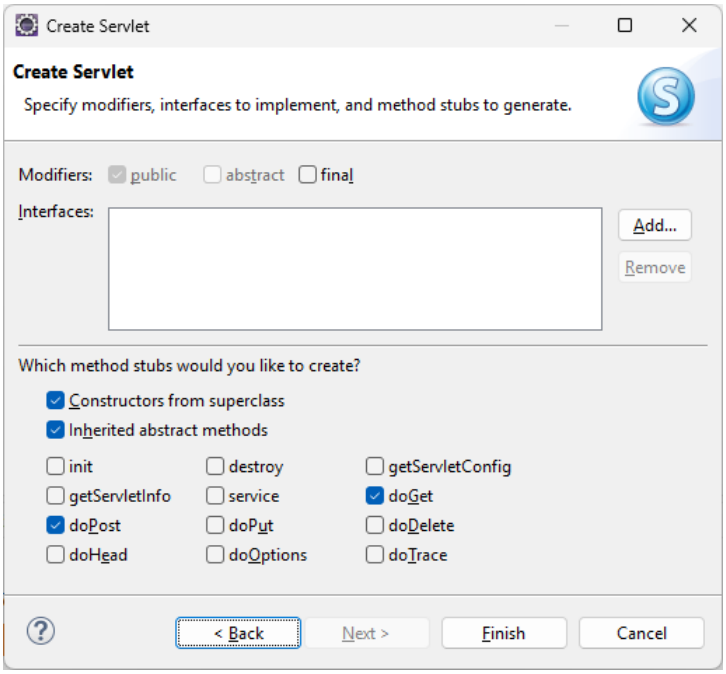
Contact | Sponsor

Le Ngoc Thach	
(+084) 0908 550 642	facebook.com/ThachLN
Website: https://xmyworkspace.com/learn/sponsor	

7. Do It Yourself (DIY) Challenges

DIY 1: Understanding the HTTP Method Stubs

In **Step 2 (Item 6)** of the Servlet Creation Wizard, Eclipse asks which method stubs you want to generate (e.g., doGet, doPost, doPut, doDelete).



Exercise:

Create a new Servlet named MethodTestServlet. Check the boxes for doGet, doPost, doPut, and doDelete.

Research and Answer:

1. **Purpose:** What is the specific role of each HTTP method? (Hint: Research **RESTful API** principles).
2. **Implementation:** When would you use doPost instead of doGet? (Think about security and data length).
3. **Why:** Why does the service() method exist if we usually only write code in doGet?

Code Challenge:

Write a unique message for each method. For example:

- In doPost: out.println("Data has been saved successfully!");
- In doDelete: out.println("Resource has been removed.");

Pro Tip: Use a tool like **Postman** or the **cURL** command to send POST or DELETE requests to your server, as a standard browser address bar can only send GET requests.

DIY 2: The "ROOT" Deployment Strategy

By default, Tomcat uses the filename of your .war file as the **Context Path** (e.g., HelloServlet.war becomes /HelloServlet).

Exercise:

1. Stop your Testing Tomcat (shutdown.bat).
2. Go to the webapps folder and delete the existing HelloServlet folder and .war file.
3. Rename your HelloServlet.war to **ROOT.war** (must be all capitals).
4. Restart Tomcat (startup.bat).
5. Try accessing: http://localhost:9090/ (without the project name).

What did you learn?

Notice how the application now loads as the "Home" application of the server. This is how real-world websites (like google.com) are deployed so users don't have to type a folder name after the domain.

Shared Learning & Feedback

To get the most out of these labs, maintain a professional workflow:

- **Git Integration:** Commit your changes after every DIY success.
- **Documentation:** Add a README.md to your repository explaining what you discovered in DIY 1 and 2.
- **Mentor Review:** Share your Git repository link with your mentor. Real-world feedback on your code structure and Git commit messages is the fastest way to grow as a developer.

Summary Table: HTTP Methods at a Glance

Method	Action	Common Use Case
GET	Read	Fetching a webpage or searching.

Method	Action	Common Use Case
POST	Create	Submitting a login form or a new comment.
PUT	Update	Updating a user's profile information.
DELETE	Delete	Removing a photo or a record from a database.

Contact | Sponsor

Le Ngoc Thach

(+084) 0908 550 642 facebook.com/ThachLN

Website: https://xmyworkspace.com/learn/sponsor