**National University of Computer & Emerging Sciences, Karachi**

**Computer Science Department**

**Fall 2024, Lab Manual - 06**

|  |  |
| --- | --- |
| **Course Code: SL3001** | **Course: Software Development and construction** |
| **Instructor:** | **Yasir Arfat** |

**Lab # 06**

# Servlets in Java

Java Servlets are server-side Java programs that handle client requests and generate dynamic responses. They are part of the Java Enterprise Edition (Java EE) platform and serve as the foundation for creating web applications. Servlets run within a web server or application server, and they are used to create dynamic content such as HTML pages, handle form submissions, or interact with databases.

**Key Concepts of Java Servlets**

1. **Servlet Lifecycle**:
2. **Initialization**: The servlet is initialized using the init() method.
3. **Service**: The service() method processes client requests (such as GET or POST).
4. **Destruction**: The servlet is destroyed using the destroy() method when it is no longer needed.
5. **Servlet API**: The Java Servlet API provides interfaces like Servlet, GenericServlet, and HttpServlet. The HttpServlet class is most commonly used as it supports HTTP-specific methods such as doGet() and doPost().
6. **Deployment Descriptor (web.xml)**: The web.xml file (also known as the deployment descriptor) is used to configure servlets, set initialization parameters, define URL mappings, and specify security constraints. It is an XML file located in the WEB-INF directory of a web application.
7. **Annotations**: Servlets can be configured using annotations (like @WebServlet) instead of web.xml. Annotations simplify the development process by allowing configuration directly within the servlet class.
8. **Servlet Container**: A servlet container (such as Apache Tomcat) is a part of a web server or application server that interacts with servlets. It manages the servlet lifecycle, handles client requests, and provides services like request dispatching and security.

**Prerequisites for Learning Java Servlets**

**Basic Understanding of Java:** Students should be comfortable with Java fundamentals, including classes, objects, inheritance, and interfaces.

**Web Technologies:** Familiarity with HTML, CSS, and basic HTTP concepts is essential for working with servlets.

**Integrated Development Environment (IDE):** It is recommended to use an IDE like IntelliJ IDEA or Eclipse, which simplifies the process of developing and deploying servlets.

**Java Development Kit (JDK):** Make sure the latest version of JDK is installed. For servlet development, a version supporting Java EE (or Jakarta EE) should be used.

**Servlet Container:** Apache Tomcat is a popular servlet container for learning servlets. It can be integrated with an IDE for seamless deployment and testing.

You can download Apache Tomcat from here:

<https://tomcat.apache.org/download-90.cgi>

**Why Use Apache Tomcat for Java Servlets?**

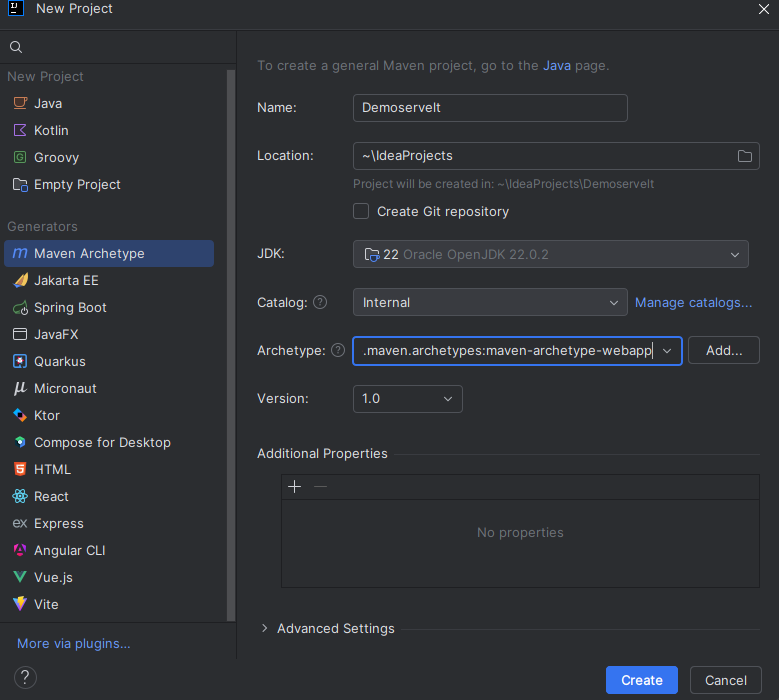
Apache Tomcat is one of the most widely used open-source web servers and servlet containers. It is specifically designed to execute Java Servlets and render web pages that include Java Server Pages (JSP). Here’s why Tomcat is an essential tool when working with servlets:

1. **Servlet Container**: Tomcat is a Java servlet container, meaning it provides an environment in which servlets can run and interact with client requests. It implements the Java Servlet and JavaServer Pages (JSP) specifications, allowing developers to build web applications with dynamic content.
2. **Lightweight and Easy to Use**: Tomcat is lightweight compared to full-featured application servers like GlassFish or WildFly. It is easy to set up and provides sufficient capabilities for most servlet-based web applications. This makes it an ideal choice for learning and developing servlet-based applications.
3. **Integration with IDEs**: Tomcat integrates seamlessly with popular IDEs like IntelliJ IDEA and Eclipse, making it easy to deploy, test, and debug applications directly from the development environment.
4. **Supports Core Java EE Specifications**: Tomcat supports essential Java EE specifications such as Servlets, JSP, and WebSocket. Although it is not a full Java EE application server, it covers the core specifications needed for most web applications.
5. **Robust Request Handling**: Tomcat efficiently manages client requests and responses. It handles multiple client requests, manages threads, and provides a stable environment for servlets to execute. This helps in building high-performance web applications.
6. **Community and Documentation**: Tomcat has an active community and extensive documentation, making it easy to find resources, troubleshoot issues, and get support when needed.
7. **Learning and Experimentation**: Tomcat is perfect for learning and experimentation. Its simplified setup allows beginners to get started with servlets and JSP without having to deal with complex configurations.

**Creating First Servlets Program**

Following the Instruction below to create your first **servlet** program

* Create a New Project
* In New Project Window Select The Maven Archetype
* Name Your Project as DemoServlet and in the archetype select org.apache.maven.archetypes:maven-archetype-webapp then click Ok and wait for intelij to create your project



* After creating your project add the following lines of code inside the dependencies

tag to in pom.xml

<dependency>  
 <groupId>jakarta.servlet</groupId>  
 <artifactId>jakarta.servlet-api</artifactId>  
 <version>5.0.0</version> <!-- Use any stable version of your choice -->  
 <scope>provided</scope>  
</dependency>

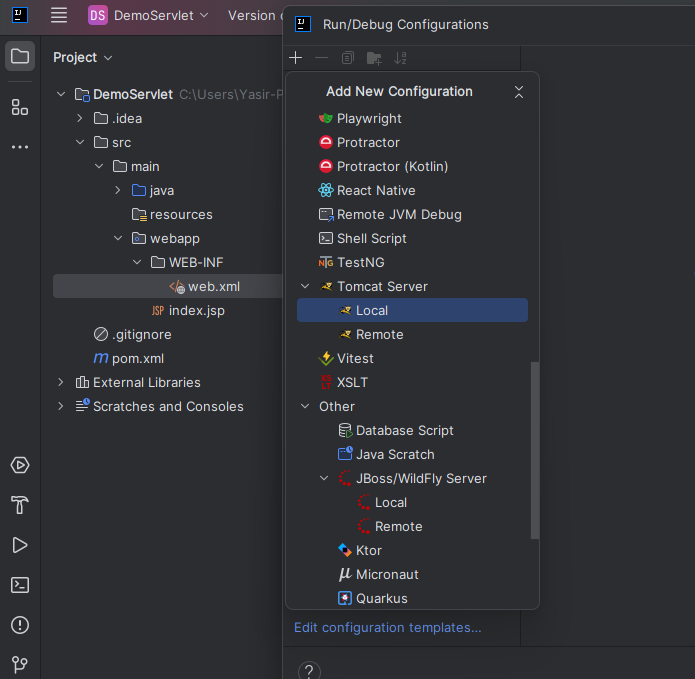
* Now Create the First Servlet Class
* In the src/main/java directory (if you did not found java Directory add the directory manually), right-click and select **New → Package**.
* Name the package something like com.example.servlet (or any package name you prefer).
* Now Right-click on the newly created package, select **New → Java Class**.
* Name the class DemoServlet and click Ok to create class.
* Open the DemoServlet class and make it extend HttpServlet.
* Override the doGet method to handle GET requests. The class should look like this:

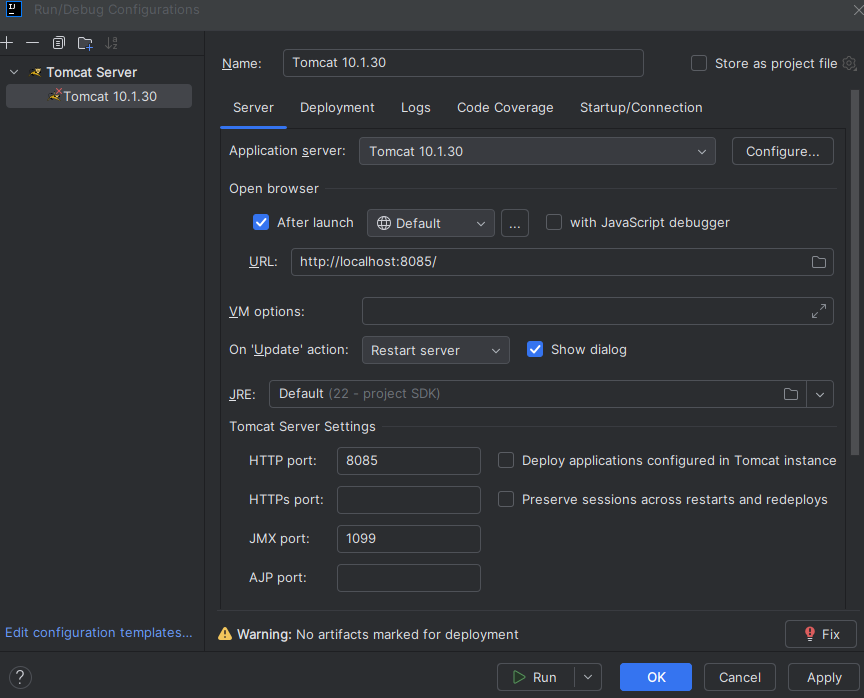
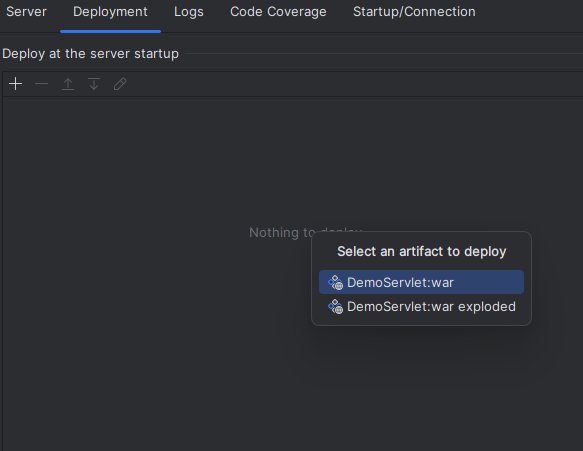
package com.example.servlet;  
  
import jakarta.servlet.\*;  
 import jakarta.servlet.http.\*;  
 import java.io.IOException;  
import java.io.PrintWriter;  
  
public class DemoServlet extends HttpServlet {  
  
 @Override  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 // Set the response type  
 response.setContentType("text/html");  
  
 // Write response content  
 PrintWriter out = response.getWriter();  
 out.println("<html><body>");  
 out.println("<h1>Hello, World! This is my first Servlet.</h1>");  
 out.println("</body></html>");  
 }  
}

* If you see any errors after adding this code then right click on the m icon on the right click on the demoServlet Maven Web App and click Reload Project.
* Now locate Locate the web.xml File
* The web.xml file is usually located in the src/main/webapp/WEB-INF directory.
* If the web.xml file is not present, create one by right-clicking on the WEB-INF directory and selecting **New → File**. Name it web.xml.
* Open the web.xml file and add the following code to configure your servlet.

<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee   
 http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  
 version="4.0">  
 <servlet>  
 <servlet-name>DemoServlet</servlet-name>  
 <servlet-class>com.example.servlet.DemoServlet</servlet-class>  
 </servlet>  
  
 <servlet-mapping>  
 <servlet-name>DemoServlet</servlet-name>  
 <url-pattern>/Demo</url-pattern>  
 </servlet-mapping>  
</web-app>

* Now Deploy and Run the Servlet
* go to **Run → Edit Configurations**.
* Click the **+** icon and select **Tomcat Server → Local**.



* Add the following configuration
* Now Select Your Application server here ( you will chose the file which you download from Tomcat website ) and also add the port on which you want to run your application.
* Now Click on The Fix You and you will see the deployment window as show below
* Chose the DemoServlet::war and the Run Your Project
* Once the server is running, open your web browser.
* Go to http://localhost:8085/DemoServlet\_war/Demo (replace DemoServlet\_war with your actual context path if you used a different one).

**Output.**

* Now you can add additional .html file to your project and then view it by changing the url.

**Now Let’s Exchange data Between Two files**

* Now we will add form.html in our webapp folder

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Registration Form</title>  
</head>  
<body>  
<h1>Registration Form</h1>  
<form action="data" method="post">  
 <label for="name">Name:</label>  
 <input type="text" id="name" name="name" required><br><br>  
  
 <label for="email">Email:</label>  
 <input type="email" id="email" name="email" required><br><br>  
  
 <label for="password">Password:</label>  
 <input type="password" id="password" name="password" required><br><br>  
  
 <input type="submit" value="Register">  
</form>  
</body>  
</html>

* After adding this we will make another servlet in the same way we created demo servlet name it as DataServlet add the below code

package com.example.servlet;  
  
import jakarta.servlet.\*;  
import jakarta.servlet.http.\*;  
import java.io.IOException;  
  
public class DataServlet extends HttpServlet {  
  
 @Override  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 // Retrieve form data  
 String name = request.getParameter("name");  
 String email = request.getParameter("email");  
 String password = request.getParameter("password");  
  
 // Set the retrieved data as request attributes to be accessed in the next page  
 request.setAttribute("name", name);  
 request.setAttribute("email", email);  
 request.setAttribute("password", password);  
  
 // Forward the request to data.html (we'll create this next)  
 RequestDispatcher dispatcher = request.getRequestDispatcher("data.jsp");  
 dispatcher.forward(request, response);  
 }  
}

Now create data.jsp file in webapp folder and add below code  
  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Registration Data</title>  
</head>  
<body>  
<h1>Registration Successful</h1>  
<p><strong>Name:</strong> <%= request.getAttribute("name") %></p>  
<p><strong>Email:</strong> <%= request.getAttribute("email") %></p>  
<p><strong>Password:</strong> <%= request.getAttribute("password") %></p>  
</body>  
</html>

Now we will update the web.xml file also and add this code to you web.xml file  
  
<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee  
 http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  
 version="4.0">  
 <servlet>  
 <servlet-name>DemoServlet</servlet-name>  
 <servlet-class>com.example.servlet.DemoServlet</servlet-class>  
 </servlet>  
  
 <servlet>  
 <servlet-name>DataServlet</servlet-name>  
 <servlet-class>com.example.servlet.DataServlet</servlet-class>  
 </servlet>  
  
 <servlet-mapping>  
 <servlet-name>DemoServlet</servlet-name>  
 <url-pattern>/Demo</url-pattern>  
 </servlet-mapping>  
  
 <servlet-mapping>  
 <servlet-name>DataServlet</servlet-name>  
 <url-pattern>/data</url-pattern>  
 </servlet-mapping>  
</web-app>

* Now Run Project

A white background with black text

Description automatically generated 