Course: Web Application Development
Lab 3 - Introduction to Servlet Programming

Content:

- Introduction to Java Servlet Technology
- How to create a servlet and run it on Visual Studio Code (VSCode)
- Practices and Exercises

Part 1: Introduction to Servlet and Servlet Life Cycle.

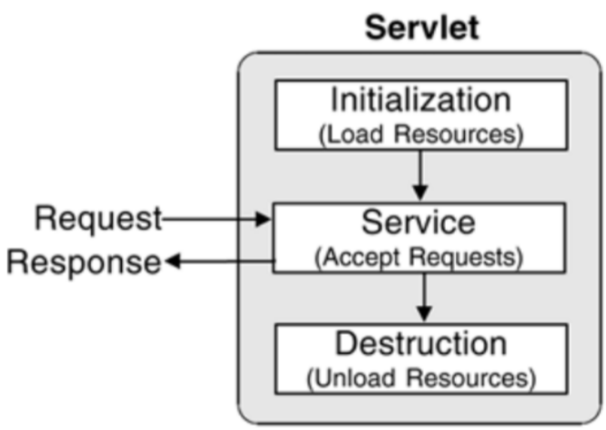
- **Recall: What is Servlet?**
  - A *servlet* is a Java programming language class that is used to extend the capabilities of servers that host applications accessed via a request-response programming model.
  - Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.
- **How does Java support Servlet Programming**
  - The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines servlet life-cycle methods.
  - The **HttpServlet** class provides methods, such as **doGet** and **doPost**, for handling HTTP-specific services.
  - When implementing a generic service, you can use or extend the **GenericServlet** class provided with the Java Servlet API.
- **Tomcat Servlet/JSP container**
  - ***Tomcat*** can act as a stand-alone Web server and also as a servlet/JSP engine for other Web servers. When you download the Tomcat server, you really get a number of packages. **Catalina** and **Jasper** are the names of the servlet and JSP containers**.**
  - Tomcat by itself is a web server. This means that you can use Tomcat to service HTTP requests for servlets, as well as static files (HTML, image files, and so on). In practice, however, since it is faster for non-servlet, non-JSP requests, Tomcat normally is used as a module with another more robust web server, such as Apache webserver or Microsoft Internet Information Server (IIS).
  - Tomcat is not a J2EE application server. However, as J2EE app servers must themselves contain a servlet container to support the servlet/JSP APIs, J2EE app

servers can embed Tomcat into their code to provide support for the Servlet and JSP APIs. One example of just such an application server is the popular open-source JBoss J2EE app server (http://www.jboss.org/).

- **Servlet Life Cycle**
  - The javax.servlet.Servlet interface defines the methods that all servlets must implement and, among others, three methods that are known as **life-cycle methods**:

```
public void init(ServletConfig config) throws ServletException
public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
public void destroy()
```

  - These life-cycle methods are each called at separate times during the life span of a servlet, from the initial creation to the moment it's removed from service and destroyed. These methods are called in the following order:



1. When the servlet is constructed, it is initialized with the init() method.

2. Any requests from clients are handled initially by the service() method before delegating to the doXxx() methods in the case of an HttpServlet. The service() method is responsible for processing the request and returning the response.

3. When the servlet needs to be removed from service, it's destroyed with the destroy() method, then garbage collected and finalized. When the container decides to take a servlet out of service, it first ensures that any service() method calls have been completed.

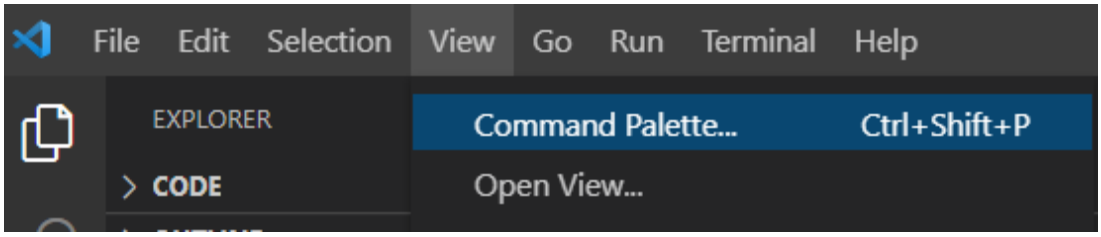
- The init method is called by the servlet container after the servlet class has been instantiated. The servlet container calls this method exactly once to indicate to the servlet that the servlet is being placed into service. The init method is important also because the servlet container passes a *ServletConfig* object, which contains the configuration values stated in the web.xml file for this application.
- The service method is called by the servlet container after the servlet's init method to allow the servlet to respond to a request. The servlet container passes a ServletRequest object and the ServletResponse object. The ServletRequest object contains the client's request and the ServletResponse contains the servlet's response.
- The servlet container calls the destroy method before removing a servlet instance from the service. This normally happens when the servlet container is shut down or the servlet container needs some free memory.

(More information: refer from the textbook: ***Core Servlets and Java Server Pages****)*

Part 2: How to create a servlet and run it on Visual Studio Code

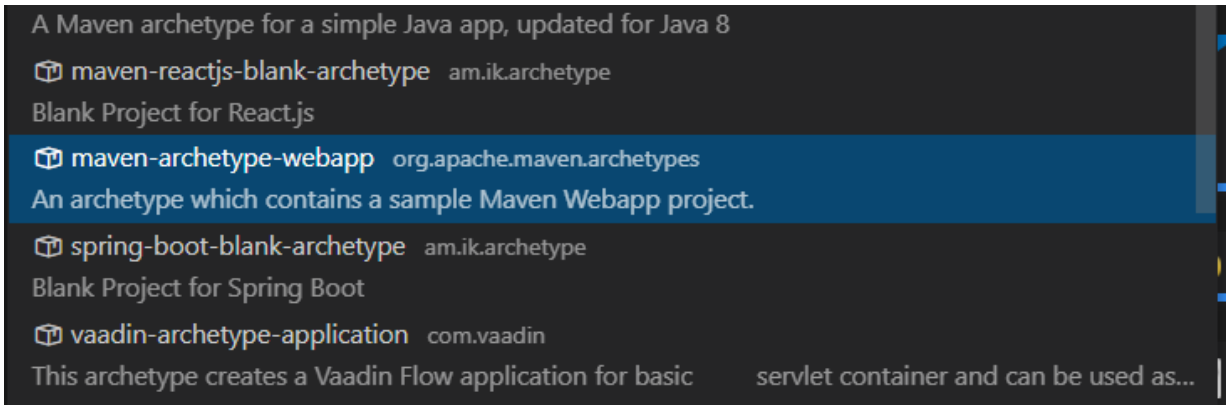
Step 1: Create a new Java Project

- Open **Visual Studio Code**
- Select **View** => **Command Palette** (or you may press **Ctrl + Shift + P**)



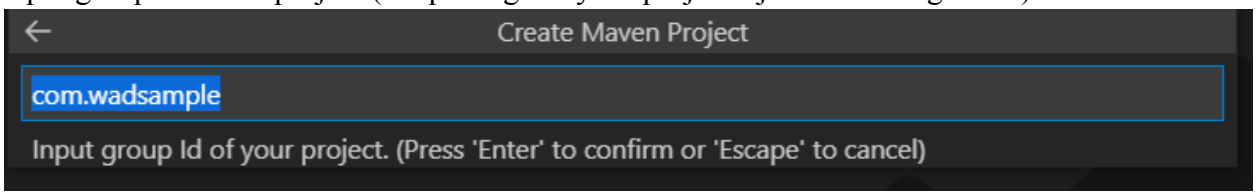
- Type "Java: Create Java Project" in the Command Palette

- Select **Maven** => **maven-archetype-webapp**

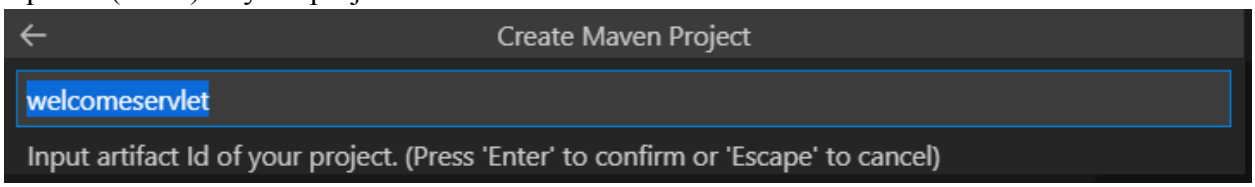


- Select version of maven-archetype-webapp (normally, the latest version is 1.4)

- Input group Id of the project (the package of your project - just for management)

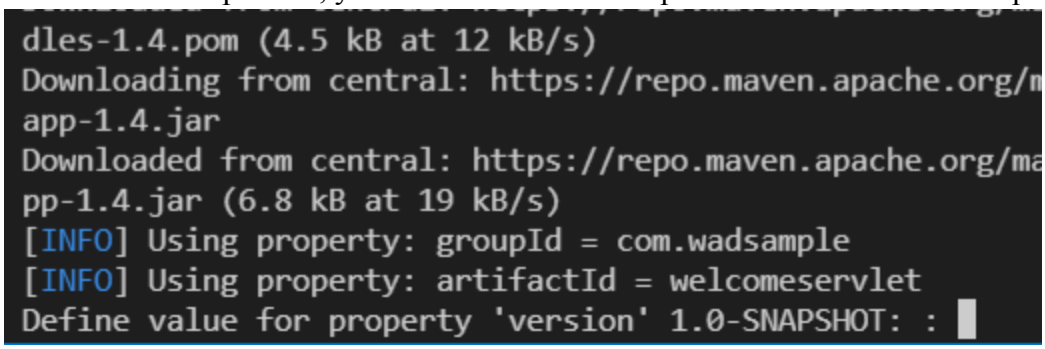


- Input Id (name) of your project



**Note: group Id and Id should be in lower case*

- Browse folder to generate the new project => **Select Destination Folder**
- Wait a few seconds for Maven to generate your project
  **Note: Internet connection is required when creating a new project*
- At the end of the process, you should be asked to input the version for the new project…



- …and confirm the final configuration

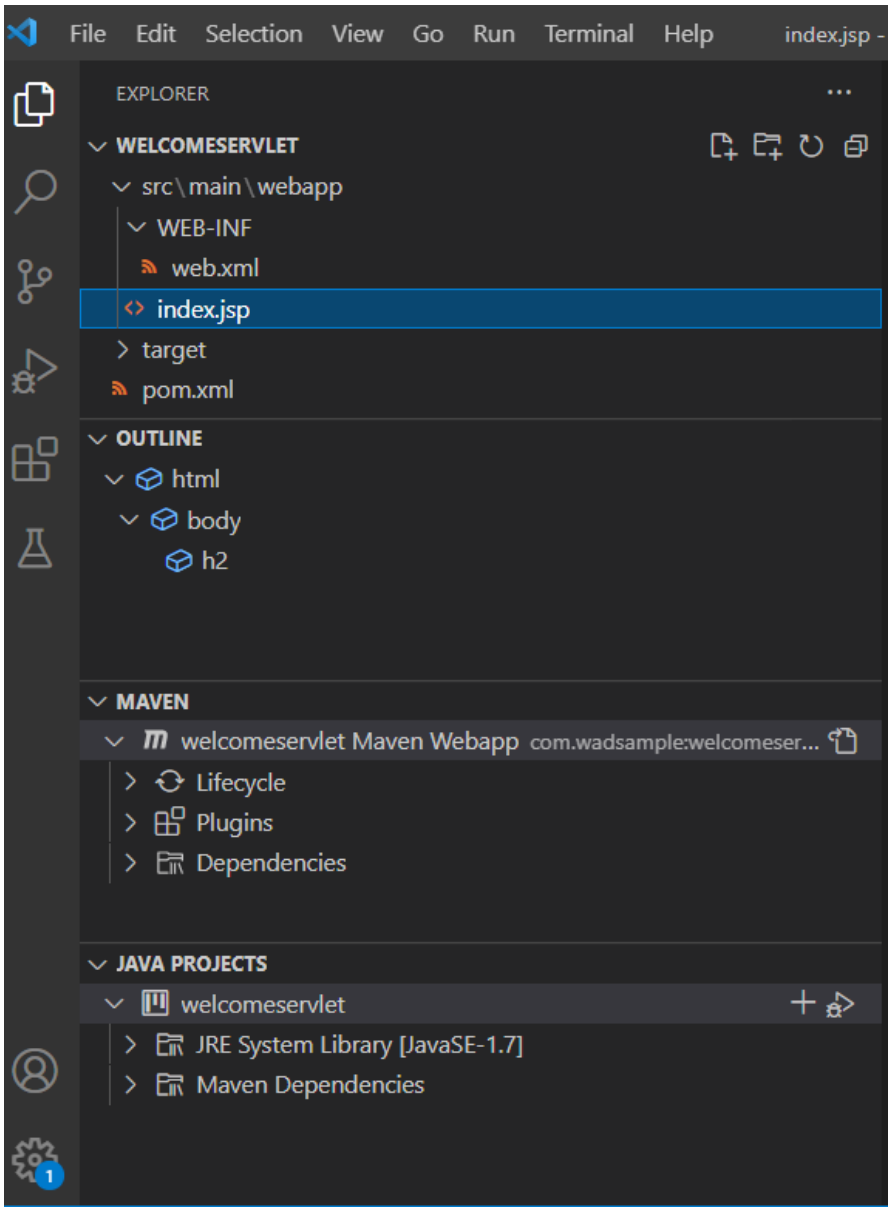
```
[INFO] Using property: groupId = com.wadsample
[INFO] Using property: artifactId = welcomeservlet
Define value for property 'version' 1.0-SNAPSHOT: : 1
[INFO] Using property: package = com.wadsample
Confirm properties configuration:
groupId: com.wadsample
artifactId: welcomeservlet
version: 1
package: com.wadsample
 Y: :
```

- When the process is done, you may press any key to close the current terminal section

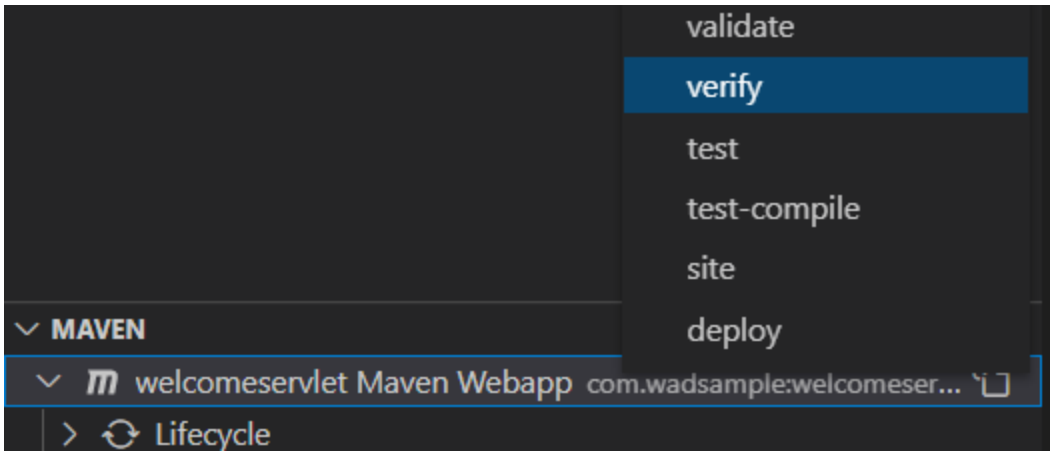
```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  04:51 min
[INFO] Finished at: 2022-03-22T20:53:10+07:00
[INFO] ------------------------------------------------------------------------

Terminal will be reused by tasks, press any key to close it.
```

Step 2: Checking Apache Maven

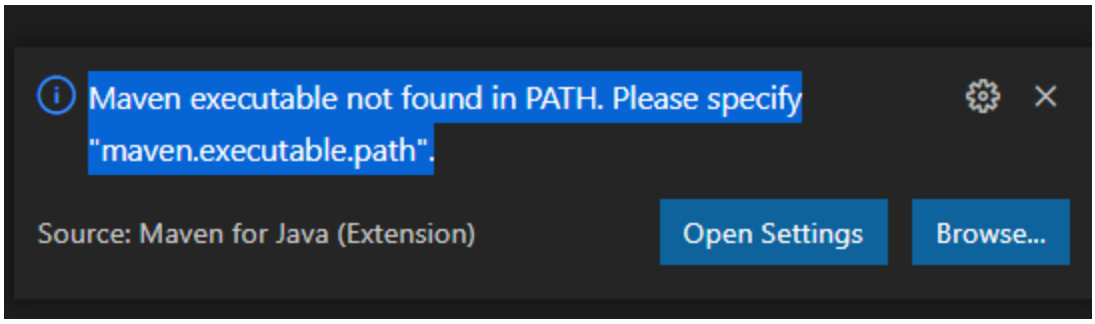
- Open the new project, there should be several files created by VSCode and Maven



- To understand the sections of Java project's explorer, please refer to:
  - https://code.visualstudio.com/docs/java/java-project
  - https://code.visualstudio.com/docs/java/java-build
- Right-click on the project under MAVEN section, choose **verify**

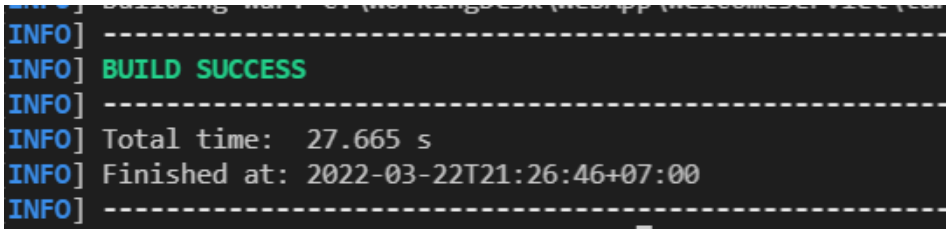


If you received the following error, you need to install Apache Maven on your computer



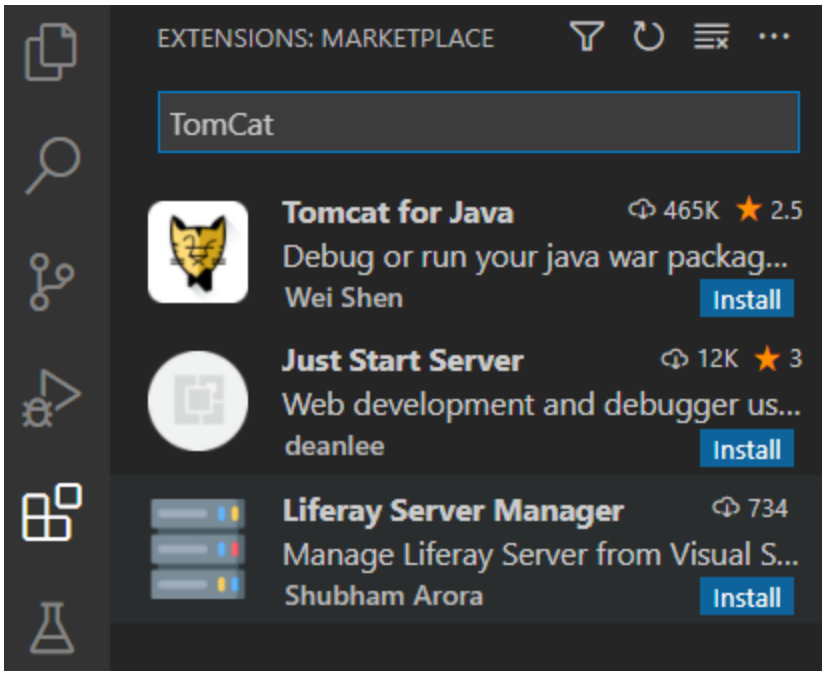
Please follow the instructions in this link (https://maven.apache.org/install.html) to install Apache Maven (The installation process will be different depending on your OS).
**Note: Remember to restart your VSCode after installing Apache Maven.*

You will receive the following message if your Apache installation is working (after choosing **verify** again)

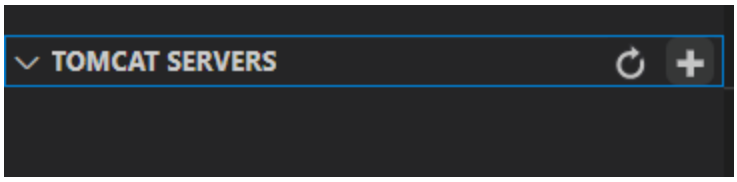


Step 3: Connect to Tomcat Server

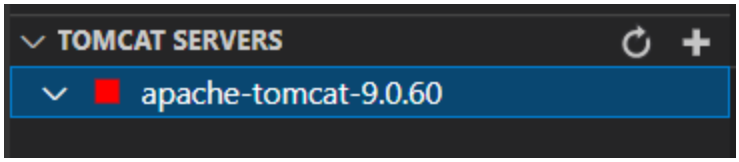
- If you did not install Tomcat for Java (VSCode extension), come to the VSCode marketplace and get it.



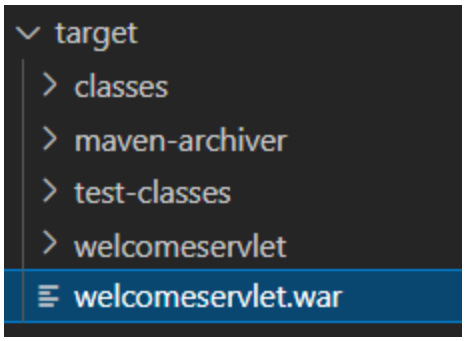
- After installation, there will be TOMCAT SERVERS section in the Explorer



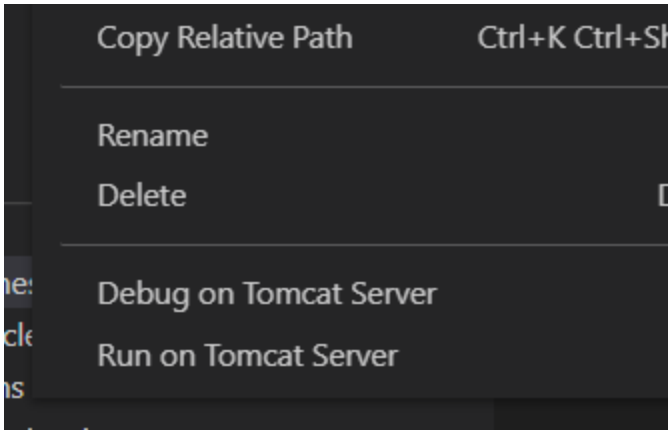
- Click on the **Add Tomcat Server** button (right next to the **refresh** button)
- Browse to your Tomcat installation folder
  (It should be installed on your computer from Lab 01)
  *Note: for the lab sections, you should use Tomcat version 9 (for newer version, the libraries for Servlet will be different)
- The selected server will be displayed in TOMCAT SERVERS section



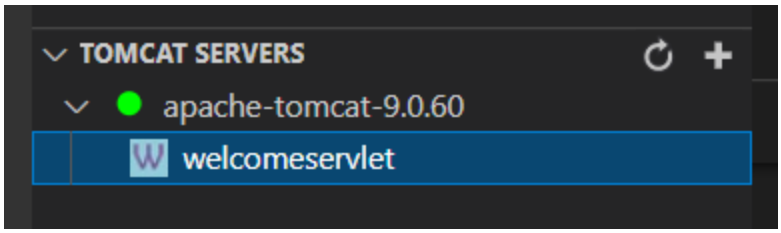
- Expand the target folder in your project explorer, you will see a folder named your project and a.war file (these were generated when you run verify command of Maven)



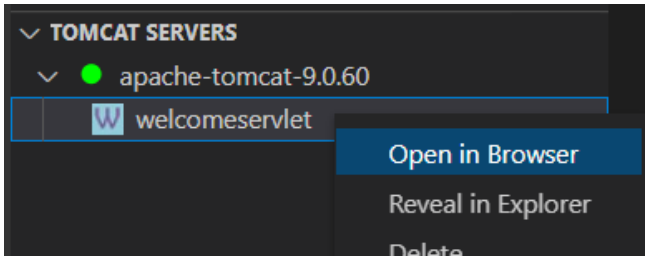
- Right-click on the folder and select **Run on Tomcat Server**
  **Note: the war file can also be used to deploy to Server*



- Your project will be deployed on the Tomcat Server

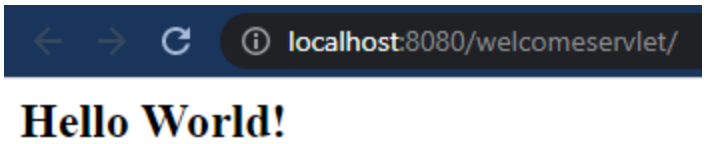


- To open your project on Browser, simply right click on it and select Open in Browser



- And the content of **index.jsp** (default page) will be displayed

  *Note: You may change default page (or homepage, welcome-page) by adding <welcome-file-list> tag to WEB-INF/web.xml
  Ref:https://cloud.google.com/appengine/docs/standard/java/config/webxml#welcome-files



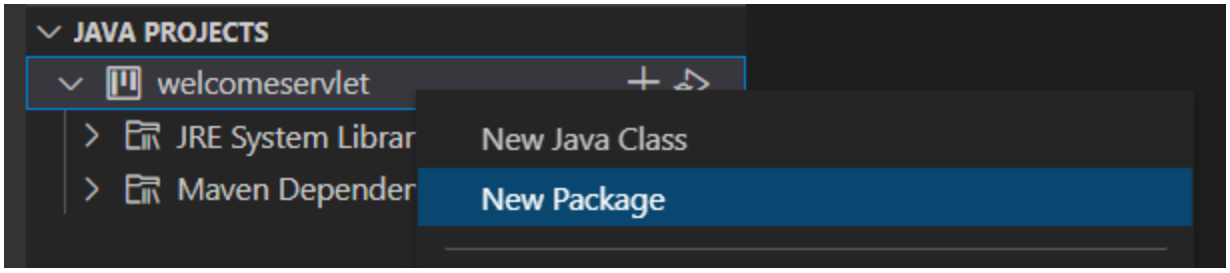
Step 4: Create a new Servlet

- Open **pom.xml** and add servlet dependency to your project

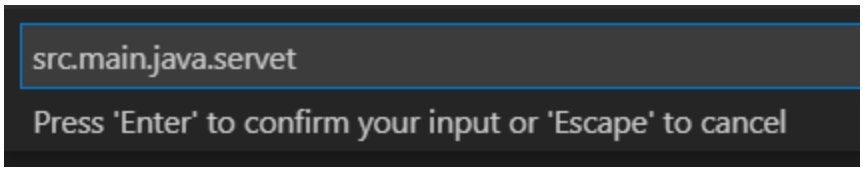
```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

**Note: You may check the latest version of servlet via this link: https://search.maven.org/search?q=g:javax.servlet*

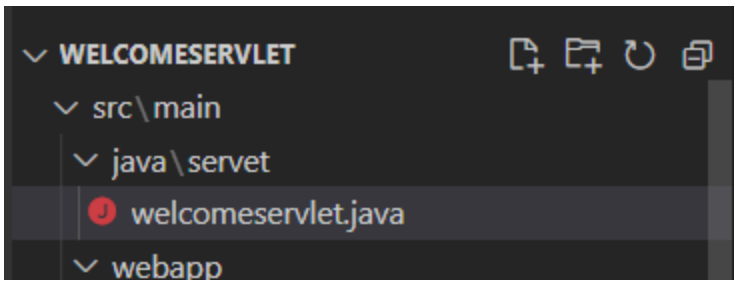
- Create a new package (folder) for servlets



*Note: your package should be inside src/main/java i.e. src.main.java.<new package> since it is the default source directory of Maven. (Actually, you may change this default value in pom.xml - https://maven.apache.org/pom.html#directories)



- Create file **welcomeservlet.java** in the new folder src.main.java.servlet



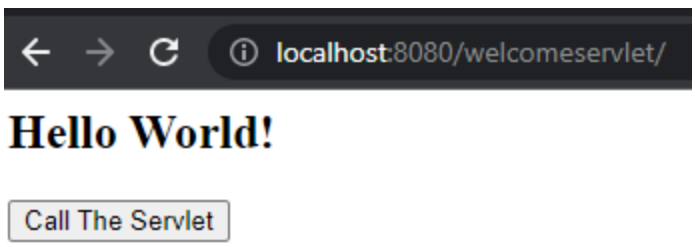
- Update content of the new file as follows:

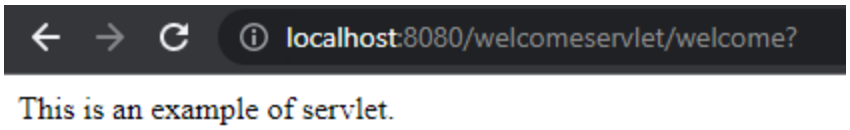
```
package servet;

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("/welcome")
public class welcomeservlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        resp.setContentType("text/html");

        PrintWriter out = resp.getWriter();
        out.println("This is an example of servlet.");
    }

}
```

- Update the content of **index.jsp** as follow:

```
<html>
<body>
<h2>Hello World!</h2>
<form action="welcome" method="get">
    <input type="submit" value="Call The Servlet">
</form>
</body>
</html>
```

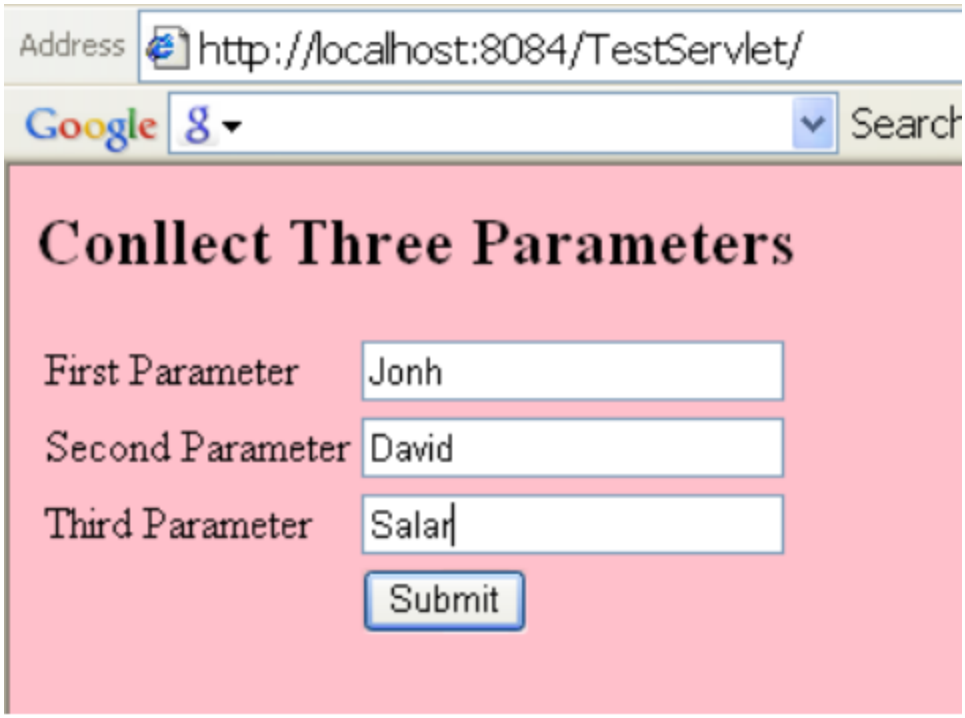
- Use Maven to verify this project again, deploy it to Tomcat and check if our servlet work correctly





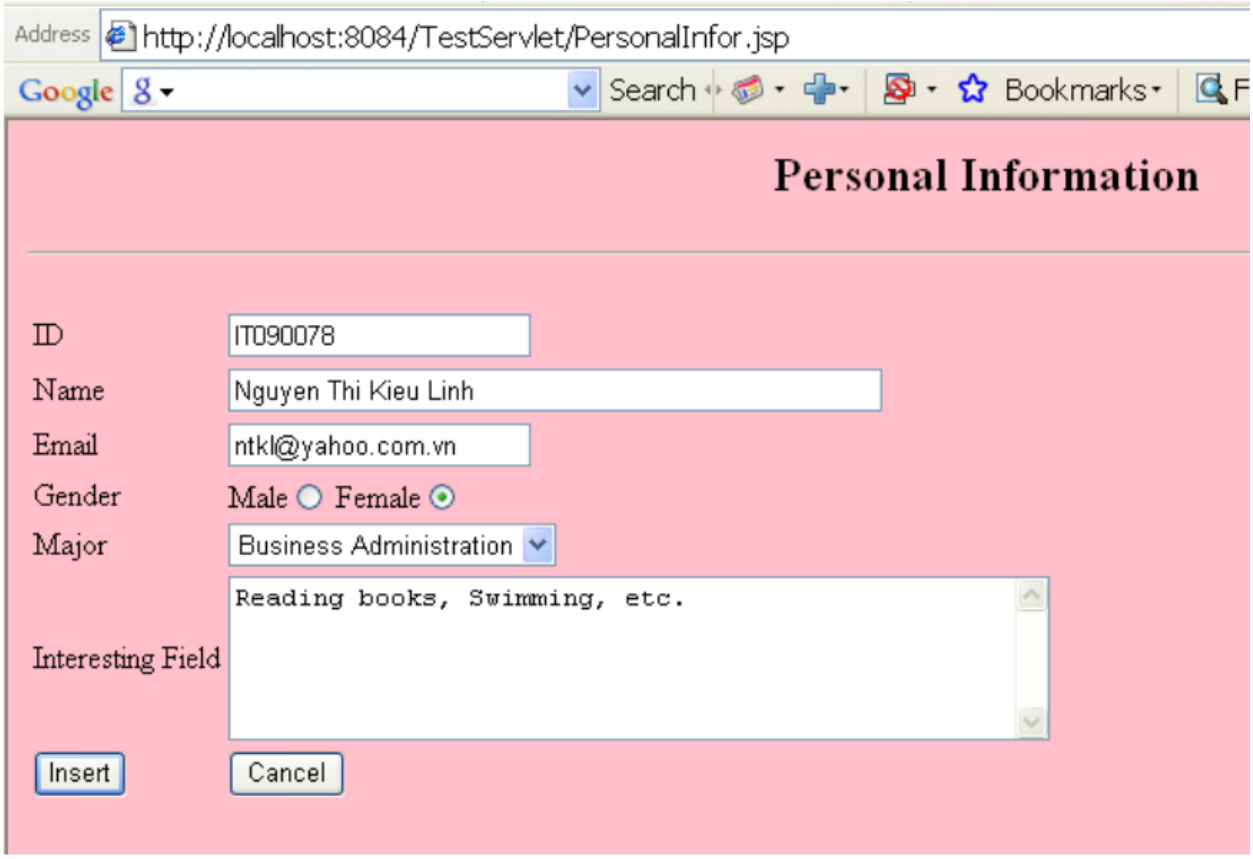
Part 3: Practices and Exercises.

Exercise1: design a form (ThreeParams.JSP) as below:

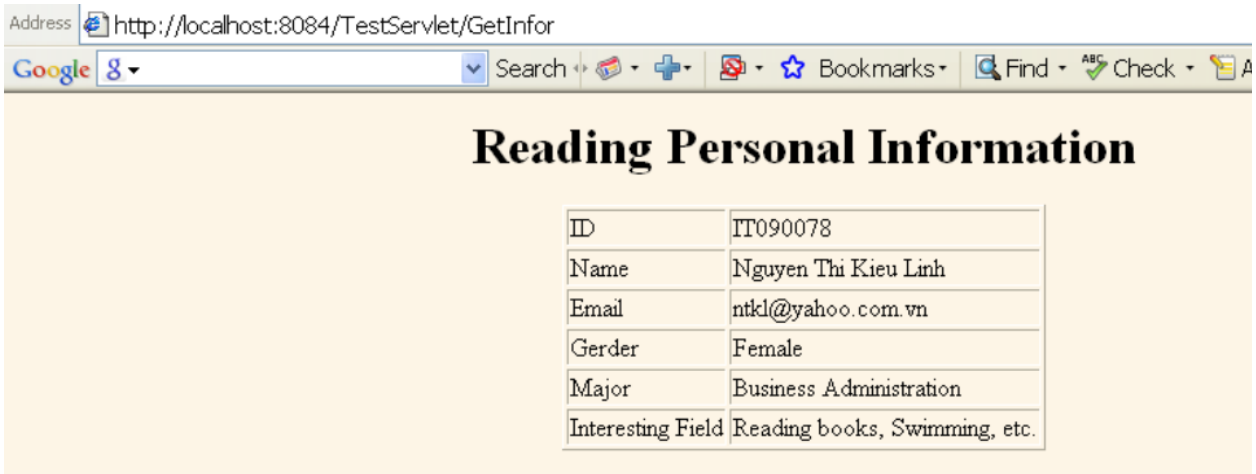


After inputting three values in the textbox -> click Submit button, it will call the servlet GetThreeParam.java

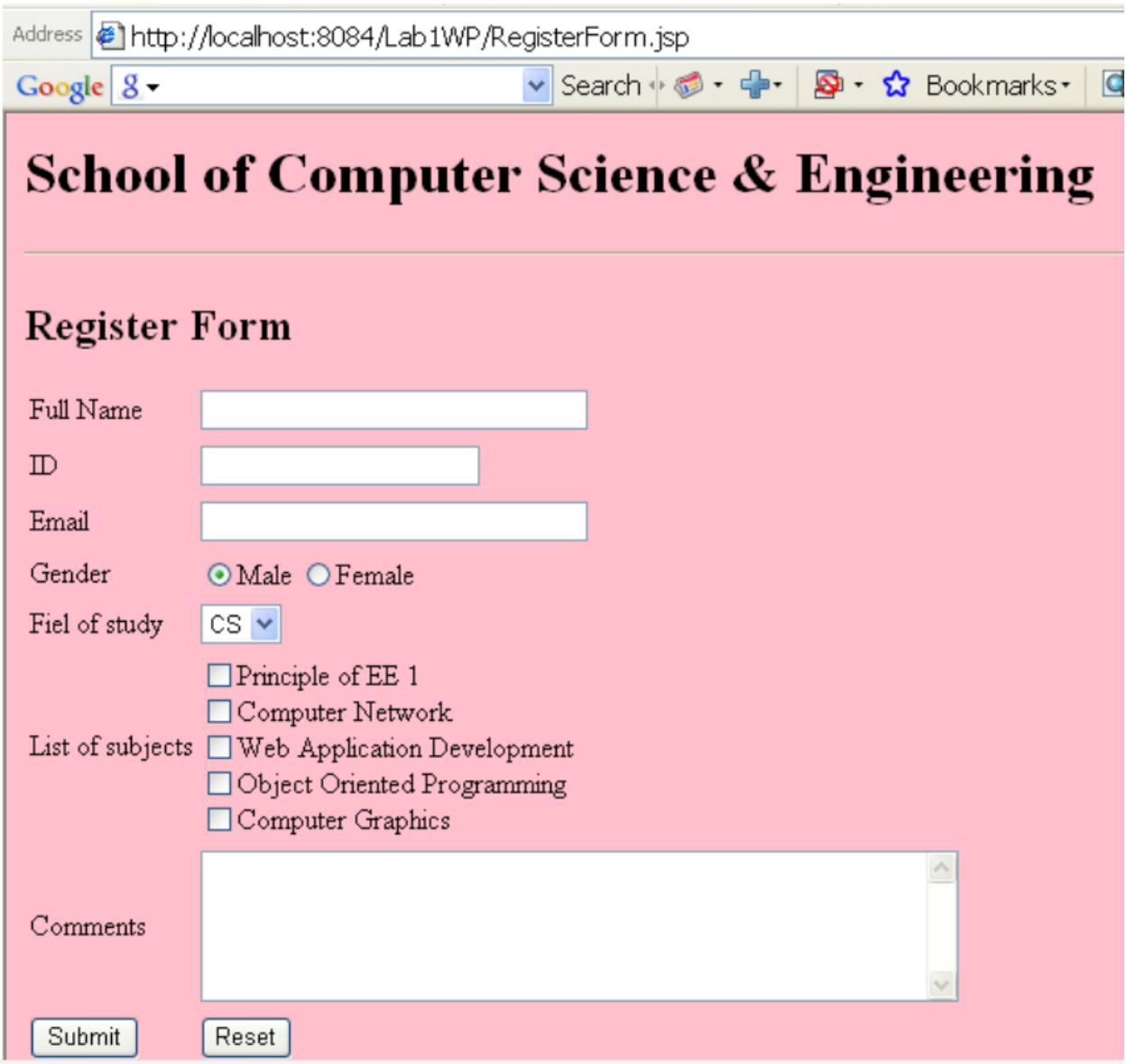
Exercise 2: Design form Personal Information (PersonalInfor.jsp)



The result from GetInfor.java



Exercise 3: Design a register form, use Servlet to get all information when the user clicks the Submit button, and put it into a new page.



The result show parameter Servlet (RegisterCourse.java)