Creating a Java Servlet using the 2.5 Servlet API with Maven and IntelliJ

Requesites

- Apache Maven (version 3.3.9 used)
- Idea IntelliJ (version 2016.3 used)
- Apache Tomcat (version 9.0 used)

1. Generate new Maven project

From the command line, go to the projects directory and execute the following command:

```
mvn archetype:generate -DgroupId=com.example.servlet25 -DartifactId=servlet25-example -
DarchetypeArtifactId=maven-archetype-webapp
```

2. Configure IntelliJ newly created Maven project

On IntelliJ choose from the main menu “File >> New >> Project from existing sources …”;

Select the directory created by the previous maven command in the directory tree and click “OK”;

Select “Import project from external model >> Maven” and click “Next”;

Besides the default options choose “Keep project files in:”, “Import Maven projects automatically” and “Automatically download: sources” options and always click “Next” until “Finish”;

3. Create project source folder

On the (left) project panel you should see “servlet25-example”. Navigate to “src\main” folder and create a folder “java”;

Right click on the “java” folder and select “Mark Directory as … >> Sources Root” from the context menu. The directory should be marked in blue now;

4. Create package

Right click on the directory “java” and choose “New >> Package”. Name it “com.example.servlet25” and click “OK”;

5. Create servlet class file

Right click on the “com.example.servlet25” package and choose “New >> Java Class”. Name it “HelloWorldServlet” and click “OK”;

6. Import dependencies

Open the “pom.xml” file and add the following to the “<dependencies>” element:

```
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
        </dependency>
```

Save the file. The dependency should be automatically downloaded and added to the “External Libraries” section on the project panel;

7. Configure the “web.xml” file for the servlet 2.5 version

Open the file “src\webapp\WEB-INF\web.xml” and replace its contents with the following:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
             http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">
</web-app>
```

Save the file.

8. Creating the servlet definition:

Inside the “web-app” element add the following:

```
    <servlet>
        <servlet-name>HelloWorldServlet</servlet-name>
        <servlet-class>com.example.servlet25.HelloWorldServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloWorldServlet</servlet-name>
        <url-pattern>/helloworld</url-pattern>
    </servlet-mapping>
```

The “<servlet-name>” can be anything you want as long as it matches between the “<servlet>” and “<servlet-mapping>” definitions;

The “<servlet-class>” is the mapping for the class we have created before in the package;

The “<url-pattern>” is the pattern the browser will recognize from the URL;

9. Coding the “HelloWorldServlet” class

Import the following packages on the top of the class:

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

Define the class as extending from “HttpServlet” as follows:

```
/**
 * HelloWorldServlet class
 */
public class HelloWorldServlet extends HttpServlet {
}
```

Implement the class with the following code:

```
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
        // Set the response message's MIME type.
        response.setContentType("text/html;charset=UTF-8");
        // Allocate a output writer to write the response message into the network socket.
        PrintWriter out = response.getWriter();

        // Write the response message, in an HTML document.
        try {
            out.println("<!DOCTYPE html>");  // HTML 5
            out.println("<html><head>");
            out.println("<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>");
            out.println("<title>Servlet 2.5 Example</title></head>");
            out.println("<body>");
            out.println("<h1>Hello World!</h1>");  // Prints "Hello, world!"
            // Set a hyperlink image to refresh this page
            out.println("<a href='" + request.getRequestURI() + "'><img
src='images/return.gif'></a>");
            out.println("</body></html>");
        } finally {
            out.close();  // Always close the output writer
```

```
        }
    }
```

10. Running the Web Application

Right click on “src\main\webapp\index.jsp” and choose “Run index.jps”. IntelliJ will create an automatic configuration for you given you have Apache Tomcat already configured as your application server inside IntelliJ;

The following address will be opened in your default browser: http://localhost:8080/index.jsp. Change it to http://localhost:8080/helloworld and press the “enter” key;

You should be now seeing the response given by the servlet: “Hello World!” (and a broken image below).

11. Debugging the Web Application

Place a breakpoint inside the “doGet” method inside your “HelloWorldServlet” class anywhere you want. On the right upper corner of the IntelliJ IDE window click on the green bug icon instead of the green play icon to run the application in debug mode. Navigate to the http://localhost:8080/helloworld and you should hit the breakpoint you placed before.