Java Web Fundamentals Exercise

Servlets

Servlets are the foundation of Java Web Applications. You are going to develop a servlet and deploy it in a Servlet Container, and will finally be able to look at the result in a web browser.

1 - Create the project

Select **File – New – Project** in IDEA
Select **Java EE – Web Application** and klick **Next** until the project is created

Look at the project structure, find the **index.jsp** file in the web directory and the **web.xml** file in the **web/WEB-INF** directory.

The **index.jsp** is a view template of type jsp. We will not be using jsp technology in this exercise, we will only copy and paste a few lines of code into the index.jsp template. A jsp template is a template of an html file with additional functionality for displaying data from Java objects. We will use another view templating technology instead called Thymeleaf in other exercises.

The **web.xml** file is responsible for configuring the web application. This is where the Servlets are declared and where you specify the URL pattern that the Servlet should respond to. We will get to this later in the exercise.

2 - Create the servlet

Right click on the src folder in the project structure and select **New – Servlet**
Give the servlet a name and click **OK**
Notice that a <servlet> tag is automatically created in web.xml

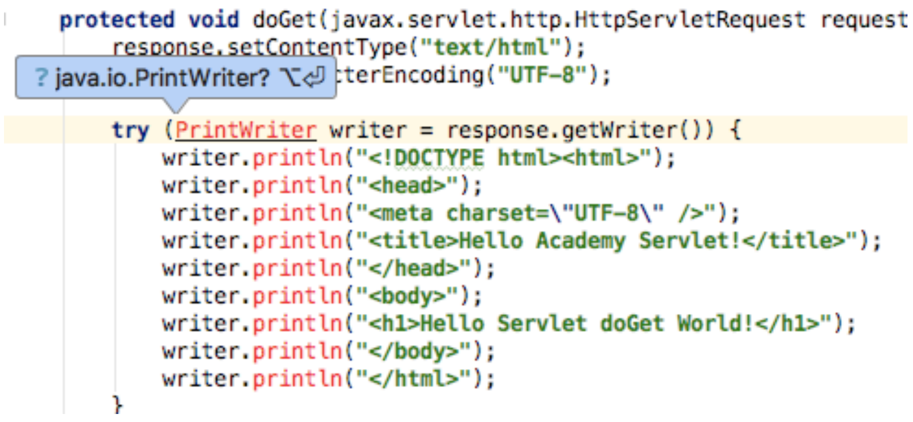
Look at the new servlet that you created in the src folder. If there is an error regarding the path to **javax.servlet.http.HttpServlet** (the word servlet is marked in red), then click on the red text, click the light bulb on the left and Select **Add Java EE 6 JARs to Module Dependencies**, and click **OK** to download the dependencies. Now the errors should go away!

Notice the **doPost** and **doGet** methods in the servlet that responds to HTTP POST and HTTP GET requests to the web application

Add this code to the doGet method:

```
try (PrintWriter writer = response.getWriter()) {
    writer.println("<!DOCTYPE html><html>");
    writer.println("<head>");
    writer.println("<meta charset=\"UTF-8\" />");
    writer.println("<title> Hello Academy World!</title>");
    writer.println("</head>");
    writer.println("<body>");
    writer.println("<h1>Hello Academy Servlet!</h1>");
    writer.println("</body>");
    writer.println("</html>");
}
```

If PrintWriter is marked in red because it is not imported, click on PrintWriter and select to Import the **java.io.PrintWriter**. The errors should go away.



```
protected void doGet(javax.servlet.http.HttpServletRequest request
    response.setContentType("text/html");
? java.io.PrintWriter? ⌥⏎ terEncoding("UTF-8");

    try (PrintWriter writer = response.getWriter()) {
        writer.println("<!DOCTYPE html><html>");
        writer.println("<head>");
        writer.println("<meta charset=\"UTF-8\" />");
        writer.println("<title>Hello Academy Servlet!</title>");
        writer.println("</head>");
        writer.println("<body>");
        writer.println("<h1>Hello Servlet doGet World!</h1>");
        writer.println("</body>");
        writer.println("</html>");
    }
```

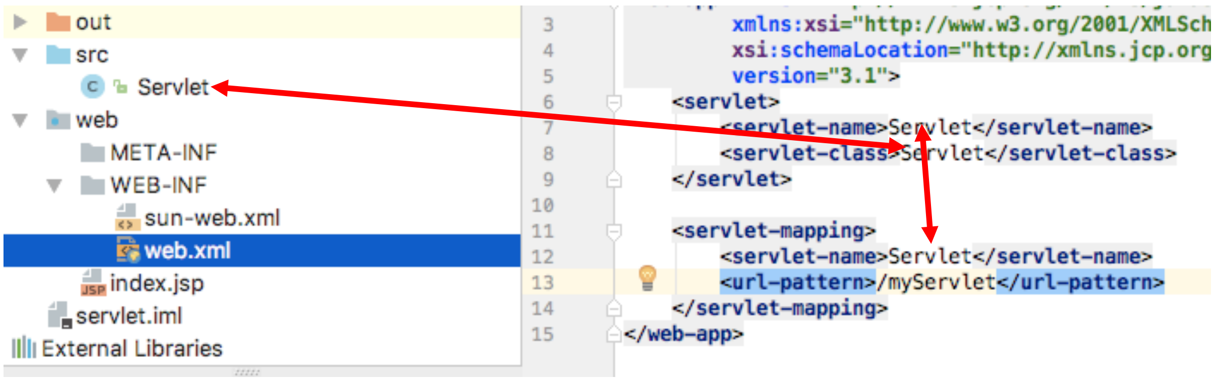
3 – Edit Web.xml

Look at the **web.xml** file

Add a servlet mapping to web.xml (keep the <servlet> tag as it is, just add a new servlet mapping). You can use your own name for the servlet instead of **Servlet** in servlet-name.

```
<servlet-mapping>
    <servlet-name>Servlet</servlet-name>
    <url-pattern>/myServlet</url-pattern>
</servlet-mapping>
```

The **<servlet-class>** in the **<servlet>** tag in **web.xml** must correspond with the class name of the Servlet. The **<servlet-name>** of the **<servlet-mapping>** must correspond with the **<servlet-name>** of the **<servlet>** tag. A link to the Servlet in a web page must have a href attributet that corresponds to **<url-pattern>**.



```
         xmlns:xsi="http://www.w3.org/2001/XMLSch
         xsi:schemaLocation="http://xmlns.jcp.org
         version="3.1">
    <servlet>
        <servlet-name>Servlet</servlet-name>
        <servlet-class>Servlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Servlet</servlet-name>
        <url-pattern>/myServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Now the **web.xml** is complete!

4 – Edit index.jsp

Edit **index.jsp**, so that it looks like this:

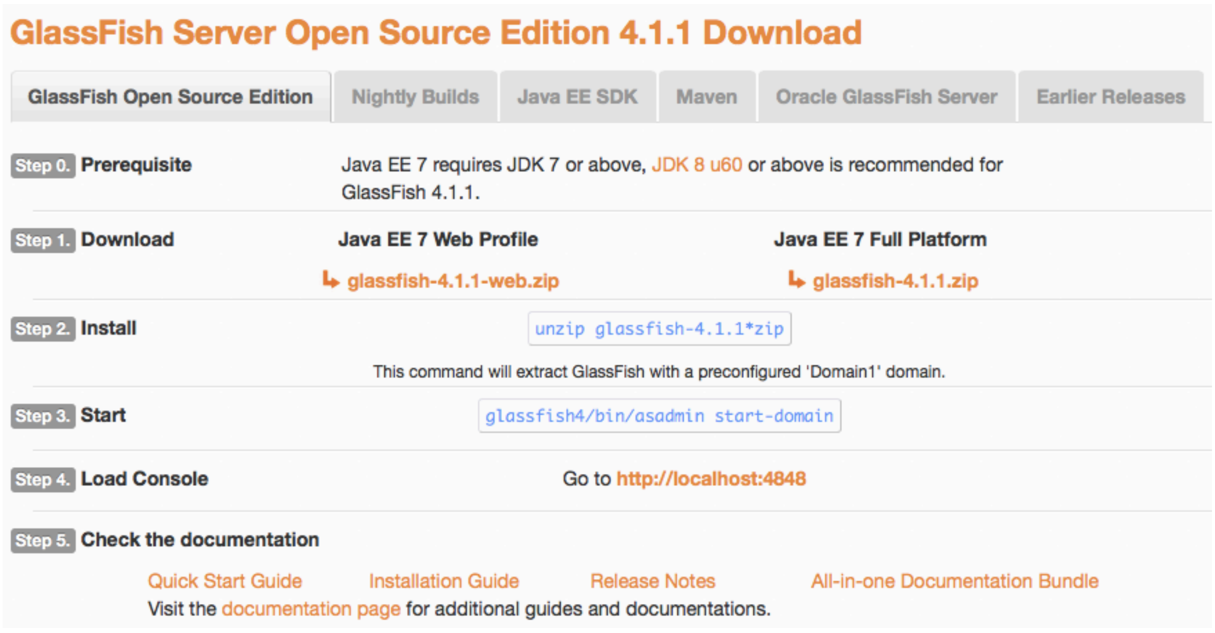
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
    <title>$Title$</title>
  </head>
  <body>
  <p>To invoke the java servlet click <a href="myServlet">here</a></p>
  </body>
</html>
```

Replace ”myServlet” in the href attribute to the same URL as you have in the **<url-pattern>** tag in the **web.xml** file, but without the / character. For example, if the URL in **<url-pattern>** is “/myServlet” then the href attribute in index.jsp should be only “myServlet” without the slash character.

5 – Install the Servlet Container

Download the Glassfish server from here:

https://glassfish.java.net/download.html


GlassFish Server Open Source Edition 4.1.1 Download

Step 0. Prerequisite — Java EE 7 requires JDK 7 or above, JDK 8 u60 or above is recommended for GlassFish 4.1.1.

Step 1. Download — Java EE 7 Web Profile: glassfish-4.1.1-web.zip — Java EE 7 Full Platform: glassfish-4.1.1.zip

Step 2. Install

```
unzip glassfish-4.1.1*zip
```

This command will extract GlassFish with a preconfigured 'Domain1' domain.

Step 3. Start

```
glassfish4/bin/asadmin start-domain
```

Step 4. Load Console — Go to http://localhost:4848

Step 5. Check the documentation

Quick Start Guide   Installation Guide   Release Notes   All-in-one Documentation Bundle

Visit the documentation page for additional guides and documentations.

Download the Java EE 7 Web Profile zip-file, not the full platform.

Extract the zip file somewhere, you will select this location later in the configuration of the Servlet Container in IDEA.

To run the servlet in IDEA we need the IDEA Web Application Support (only possible in Ultimate Edition)

Right click your project and select **Open Module Settings**, check that you have **Web** in Modules.

If you do have it, click **Web** and click the **Add Application Server specific descriptor** and select **GlassFish Server**, it gets added to your Deployment Descriptors. Click **OK**.

If you don't have it, right click the project and select **Add Framework Support** and click **Web application**.

6 - Build project and run in the Servlet Container

Click **Build – Build** project in the menu to build the project.

Click **Run – Run** in the menu and click **Edit Configurations…**

Click the + sign and select **GlassFish Server – Local**.

Click **Configure…** button to configure the Application server. Select the location of your GlassFish server.

Select **Server Domain: domain1**

Click Deployment and select **servlet:war exploded** under **Deploy at the server startup**.

Click **Run** button.

Wait for it.

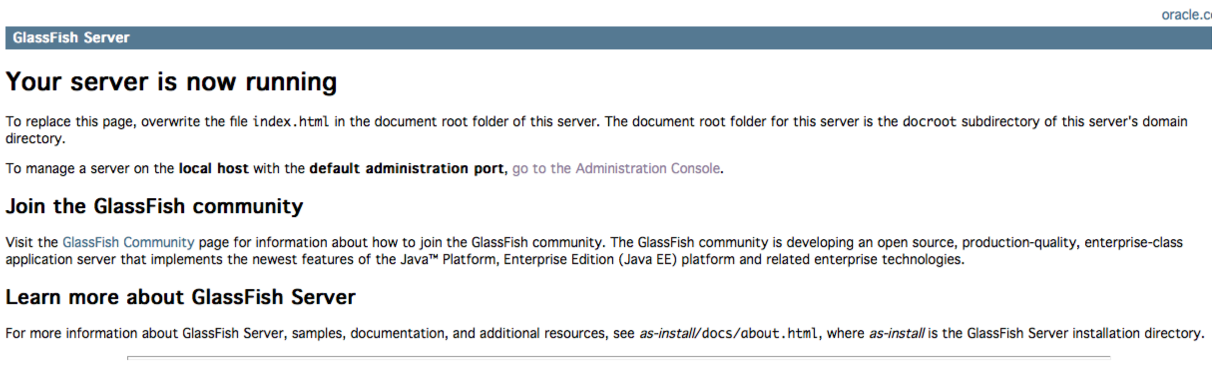
Wait for it (this could take a while)

Then you should see **Waiting for domain1 to start…**, and then you should see the Server running in IDEA, with **Command start-domain executed successfully**.

Check if you can access the server from a web browser with the following URL:

http://localhost:8080

You should see the GlassFish Server responding:



Your server is now running

To replace this page, overwrite the file `index.html` in the document root folder of this server. The document root folder for this server is the `docroot` subdirectory of this server's domain directory.

To manage a server on the **local host** with the **default administration port**, go to the Administration Console.

Join the GlassFish community

Visit the GlassFish Community page for information about how to join the GlassFish community. The GlassFish community is developing an open source, production-quality, enterprise-class application server that implements the newest features of the Java™ Platform, Enterprise Edition (Java EE) platform and related enterprise technologies.

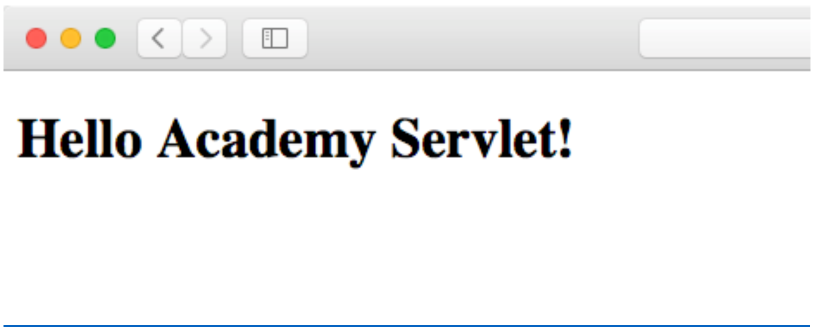
Learn more about GlassFish Server

For more information about GlassFish Server, samples, documentation, and additional resources, see *as-install*/`docs/about.html`, where *as-install* is the GlassFish Server installation directory.

And finally check if you can access your servlet with the following URL:

http://localhost:8080/servlet_war_exploded/myServlet

(change ”myServlet” to whatever you have in your **<url-pattern>** in **web.xml**)



Hello Academy Servlet!

You can control the server with the buttons to the right. Stop the server with the red stop button and start the server with the green Run button.

Congratulations!

You are now running the Servlet that you have developed in a Servlet Container that you have downloaded and installed!

7 - Debug the web app and look at the request and response objects

Run the web app in debug mode and put a break point at the beginning of the **doGet** method.

If there is a problem running in debug mode, click the **Configurations** dropdown next to the run/debug buttons and look for a **Fix** button to fix the problem. Try stopping the web app and run it again in debug mode. It could take some time to start again.

When it has restarted, enter the URL in a web browser and expand the request and respond objects at the break point in the Servlet.

Notice for example that the request object has several attributes, for example cookies, attributes, a parameterMap, a session among other variables. The response object has some attributes, for example the writer attribute. That is the one we are using for printing out the html by calling the method **response.getWriter()** in the Servlet.

8 - Add more behavior to the web app

Think about how you could add more behavior to your web app. Now, your web app has only one response to one URL. What if you need your web app to respond to more URLs? What do you need to do?

You could add another Servlet and another Servlet Mapping in **web.xml** that responds to a different url-pattern. But this could quickly add up to quite many Servlets.

You could also add parameters to your request and let your Servlet read the parameters and take different actions depending on the parameter.

9 - Handle parameters in the URL

Stop the web app and add this line to the Servlet code in the beginning of the **doGet** method, before the break point:

```
String view = request.getParameter("view");
```

Run the web app in debug mode again.

Enter the URL to the web app, and append this string to the end of the URL:

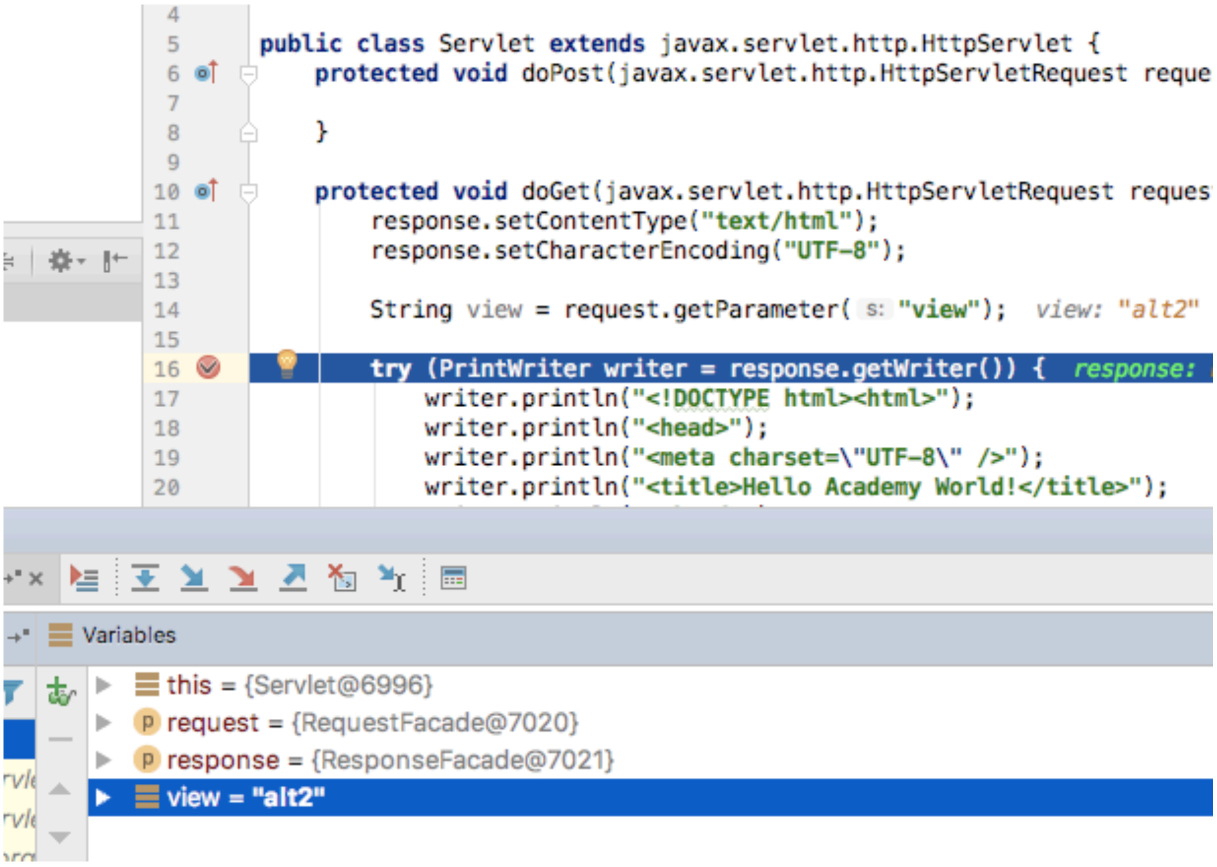
?view=alt2

The entire URL should look something like this:

localhost:8080/servlet_war_exploded/myServlet?view=alt2

You are adding a parameter to the end of the URL with the name “view” and the value “alt2”.

Look at the view variable in debug mode. It should now have the value alt2. You have successfully got a hold of the value of the view parameter that you sent in the URL.



```

public class Servlet extends javax.servlet.http.HttpServlet {
    protected void doPost(javax.servlet.http.HttpServletRequest reque

    }

    protected void doGet(javax.servlet.http.HttpServletRequest reques
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");

        String view = request.getParameter( s: "view");   view: "alt2"

        try (PrintWriter writer = response.getWriter()) {  response:
            writer.println("<!DOCTYPE html><html>");
            writer.println("<head>");
            writer.println("<meta charset=\"UTF-8\" />");
            writer.println("<title>Hello Academy World!</title>");
```

Variables

```
this = {Servlet@6996}
request = {RequestFacade@7020}
response = {ResponseFacade@7021}
view = "alt2"
```

Stretch Tasks (if you have time)

What could you do with this functionality (reading request parameters from the request)?

Try to print an alternative html output depending on the value of the view parameter!

This way you can change the behavior of the web app using the same URL only by changing the value of the view parameter.

Try to add another parameter like this:

```
?view=alt2&anotherParam=test3
```

Try to get a hold of the value in the Servlet and change the output depending on the value of the parameters.