

# REST API Security

## Creating a Basic REST API

---

Guadalupe Ortiz Bellot

Computer Science and Engineering Department



**UCA**

Universidad  
de Cádiz

# Contents

---

1. Creation of a Tomcat Server in Eclipse
2. Web dynamic project creation
3. Jersey library inclusion
4. Package and class creation
5. web.xml file creation
6. Service deployment in the service
7. Testing it from Postman



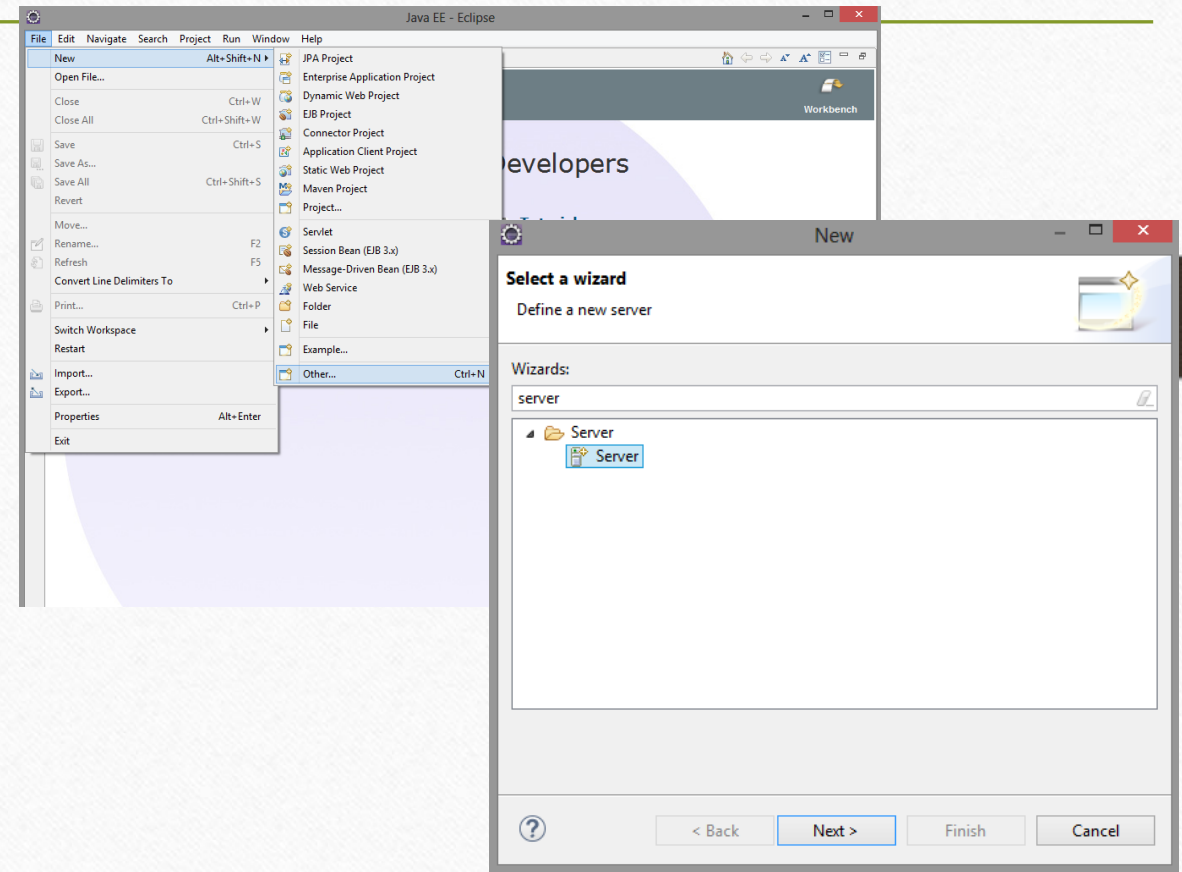
# Contents

---

1. **Creation of a Tomcat Server in Eclipse**
2. Dynamic web project creation
3. Jersey library inclusion
4. Package and class creation
5. web.xml file creation
6. Service deployment in the service
7. Testing it from Postman

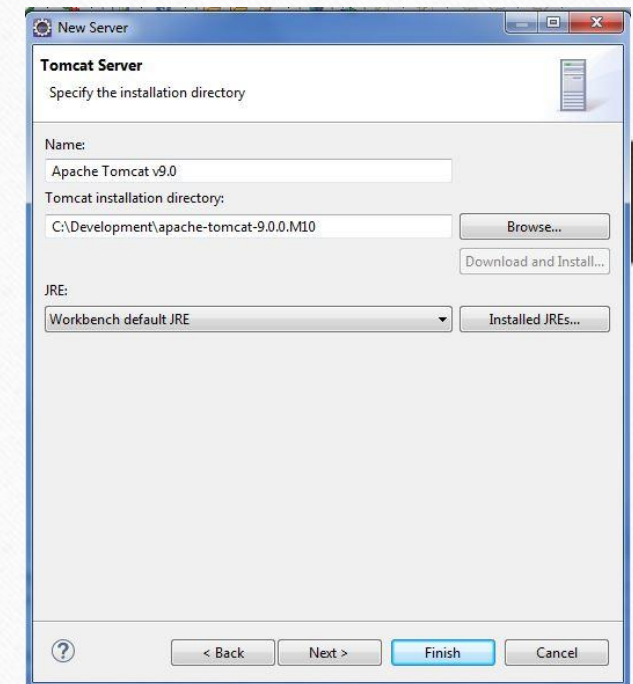
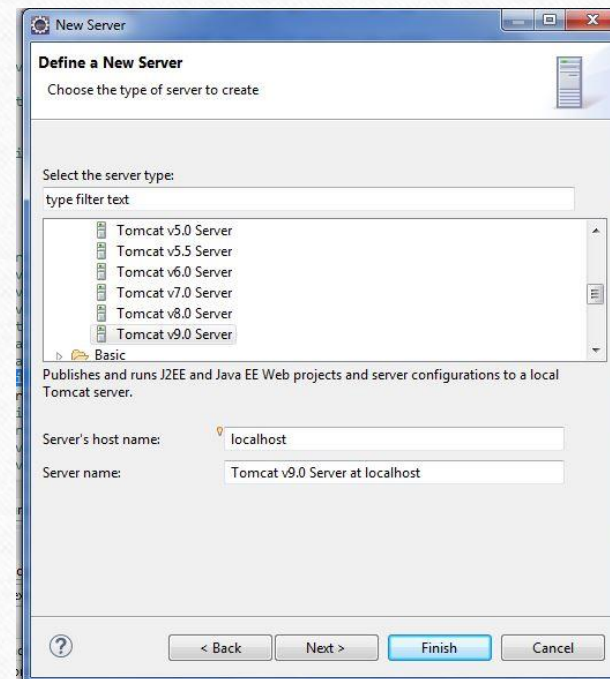
# 1. Creation of a Tomcat Server in Eclipse (i)

- **REMINDER:** it is recommended to create a folder for the workspace, for instance inside the root folder created for the software installation (Development)
- We open Eclipse and select the workspace where we will store the created projects.
- We create a Tomcat server instance:  
File → New → Other → Server → Server



# 1. Creation of a Tomcat Server in Eclipse (ii)

- Select Tomcat 9 and click Next
- We select the folder where we have installed the Tomcat and the JRE we desire to use.
- For the JRE we have to select the installed JDK. Usually, by default, it is JRE selected. If so, follow the steps in the following slide.

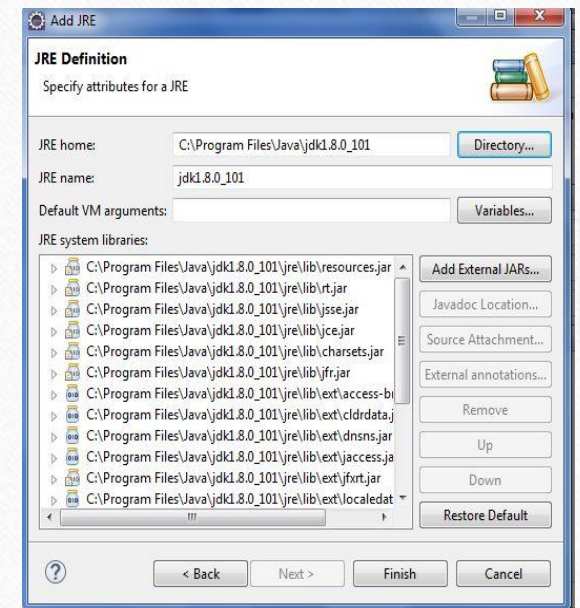
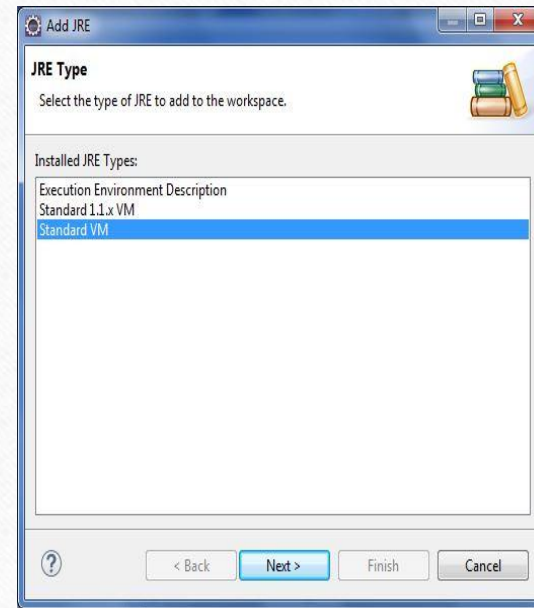




# 1. Creation of a Tomcat Server in Eclipse (iii)

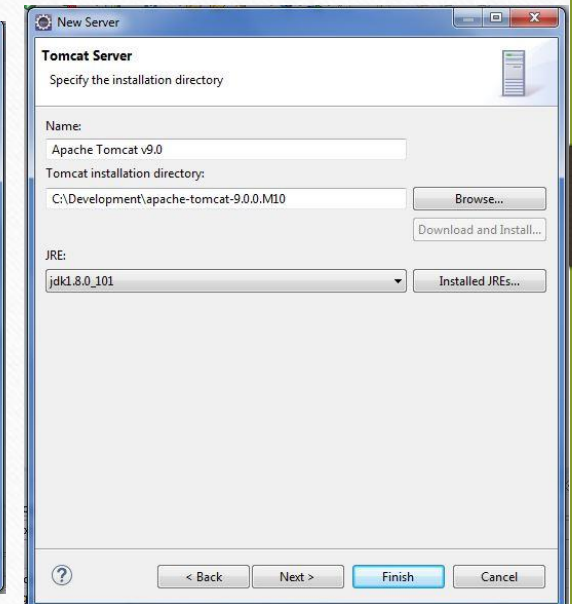
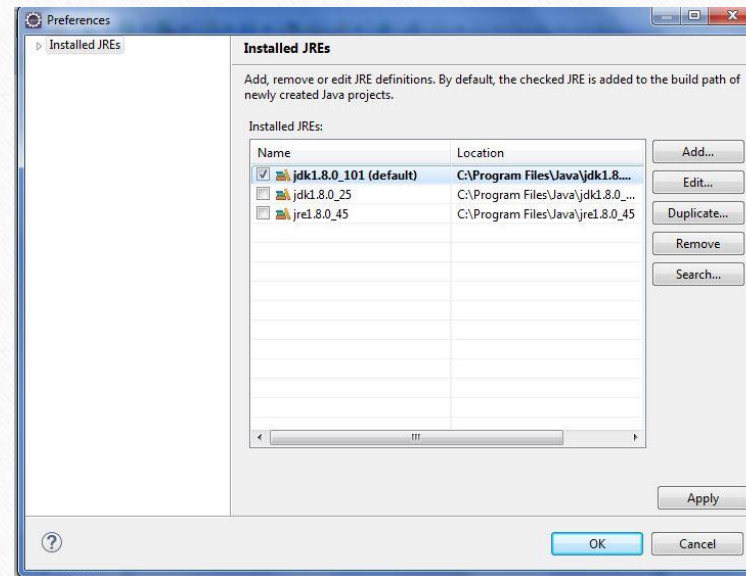
How to select the JDK: Click on *Installed JRE*

- OPTION 1. If the JDK is not in the list, follow the steps below:
  - Click on the Installed JRE → Add → Standard VM → Next
  - In the pop-up screen, select the folder where the JDK is installed. Then click on Finish
  - Now follow OPTION 2 Steps



# 1. Creation of a Tomcat Server in Eclipse (iv)

- OPTION 2. If the JDK is in the list:
  - Click on Installed JRE → Select the JDK and click OK.
  - It may happen that after clicking OK in the following Windows the JRE is yet selected. You have to open the deployable list, select the JDK and Finish.

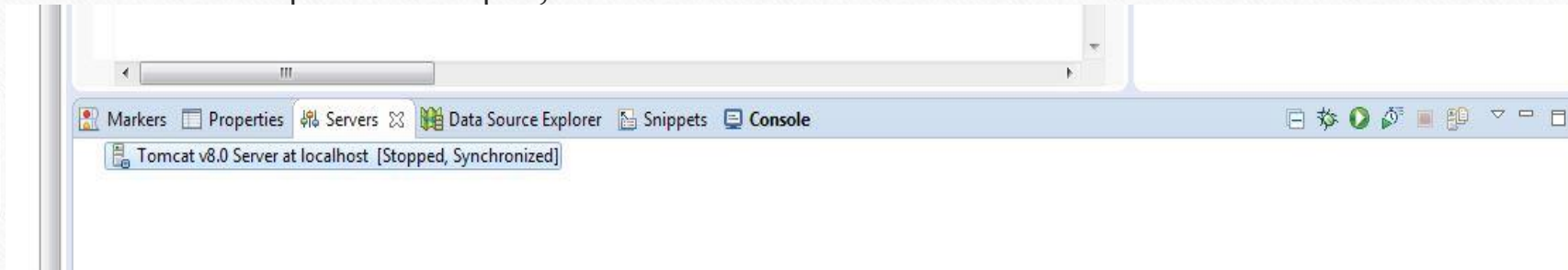




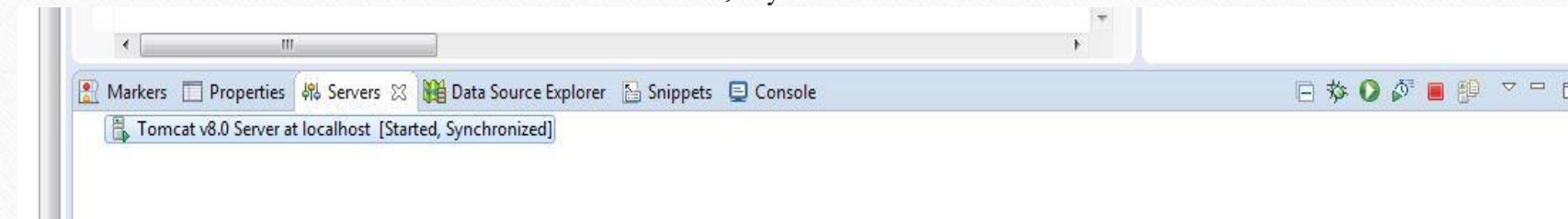
# 1. Creation of a Tomcat Server in Eclipse (v)

Start the Apache Tomcat server:

- In the down part of Eclipse, we have to click on the Server tab and there the Tomcat should be (Stopped):



- In order to launch the server we right-click on it and select Start. We wait until the start is finished, once finished we will see that the server status is Started, Synchronized:



- NOTE: To see the servers view in Eclipse, Window → Show View → Server → Servers.



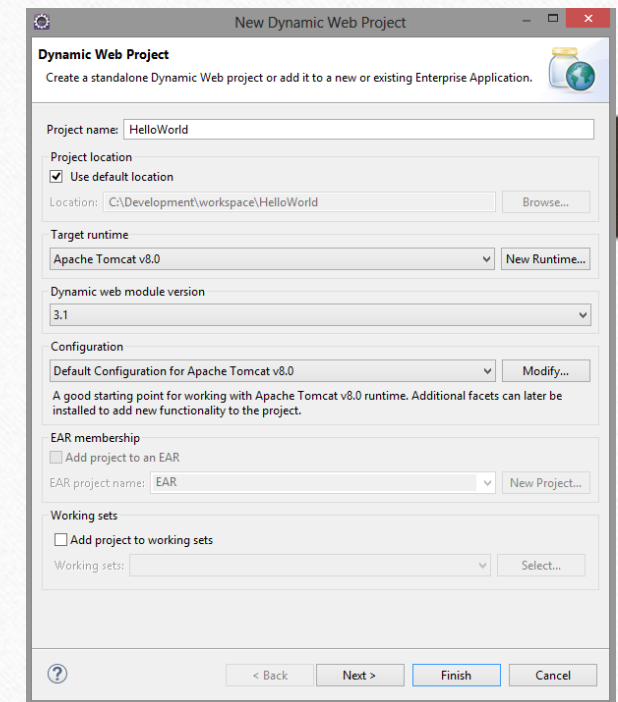
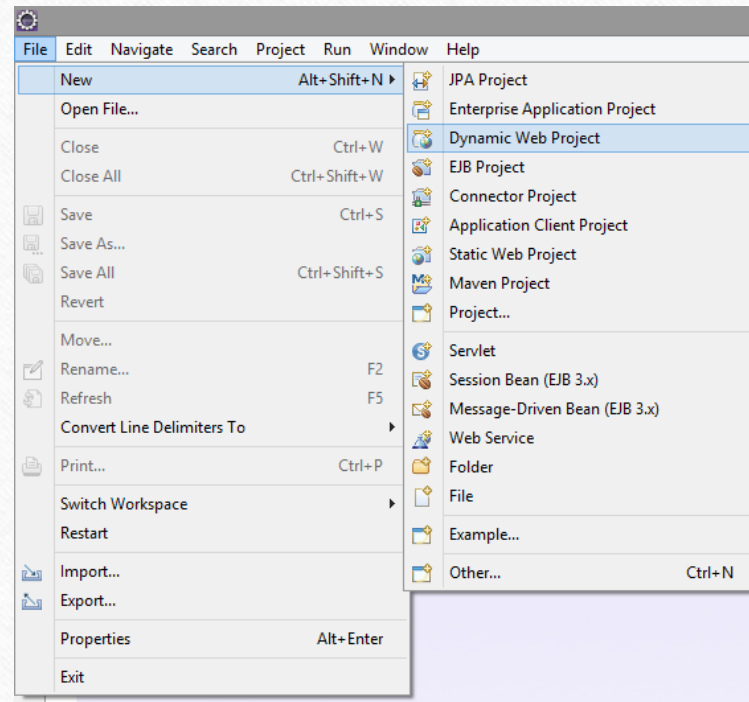
# Contents

---

1. Creation of a Tomcat Server in Eclipse
2. **Web dynamic project creation**
3. Jersey library inclusion
4. Package and class creation
5. web.xml file creation
6. Service deployment in the service
7. Testing it from Postman

## 2. Dynamic Web Project Creation

- We create a Dynamic Web Project named HelloWorld:
  - New → Dynamic Web Project
  - Fill in Project name and select the Tomcat recently created as the Target runtime and → Finish
- **VERY IMPORTANT: Respect upper-case and lower-case conventions for Java, specially for the first letter**





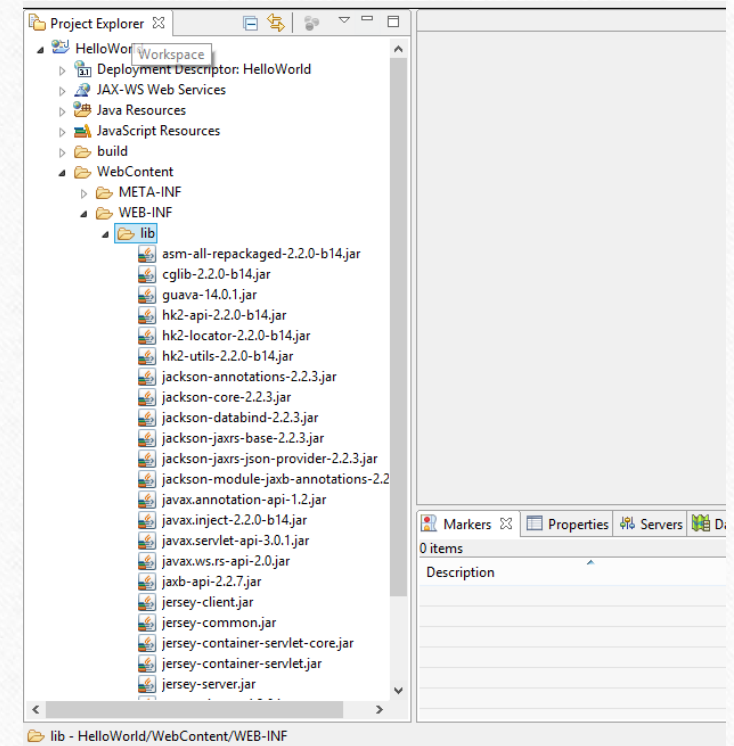
# Contents

---

1. Creation of a Tomcat Server in Eclipse
2. Dynamic Web project creation
- 3. Jersey library inclusion**
4. Package and class creation
5. web.xml file creation
6. Service deployment in the service
7. Testing it from Postman

# 3. Jersey Library Inclusion

- We copy Jersey libraries in WebContent/WEB-INF/lib (we should have unzipped them previously)
- If they are not shown inside the folder after pasting them, you should refresh the project (Right click on the project → Refresh)





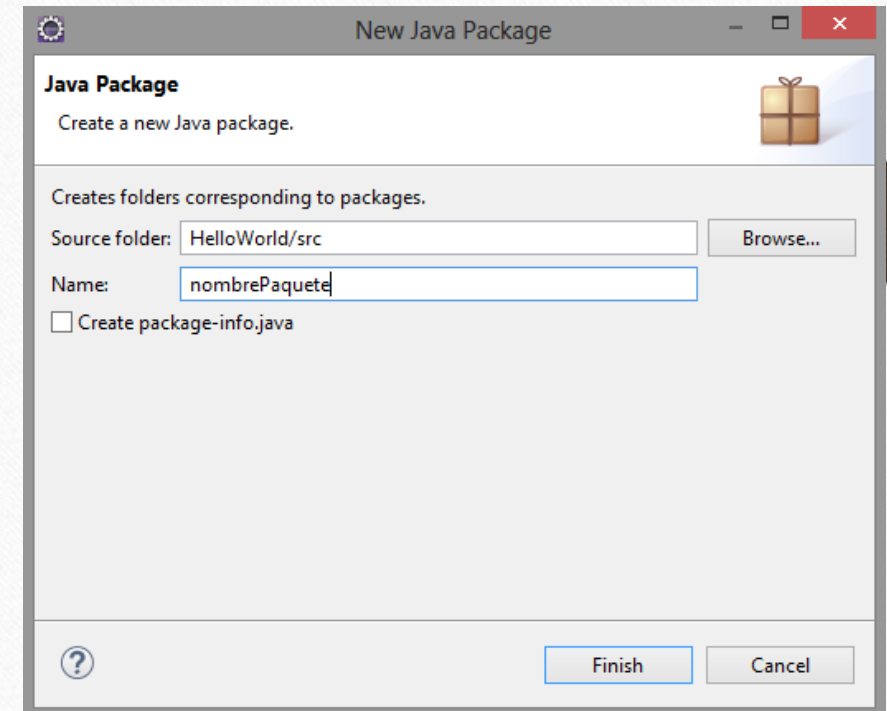
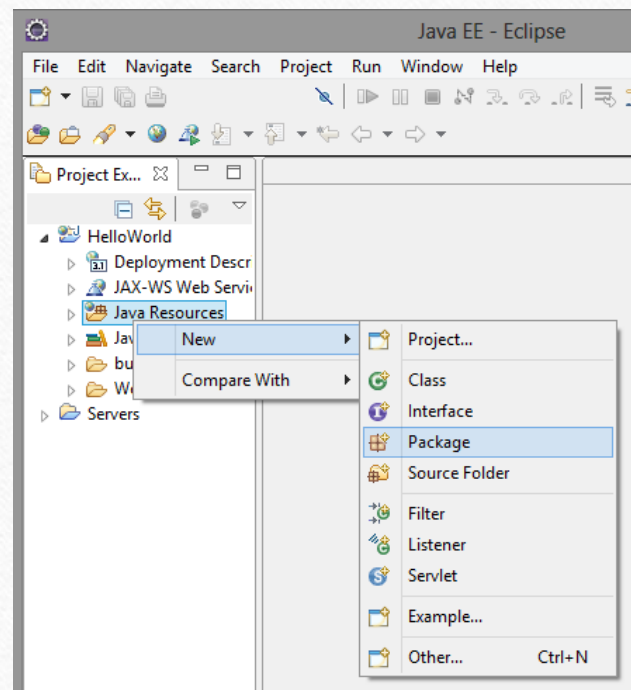
# Contents

---

1. Creation of a Tomcat Server in Eclipse
2. Dynamic Web project creation
3. Jersey library inclusion
4. **Package and class creation**
5. web.xml file creation
6. Service deployment in the service
7. Testing it from Postman

# 4. Package and Class Creation (i)

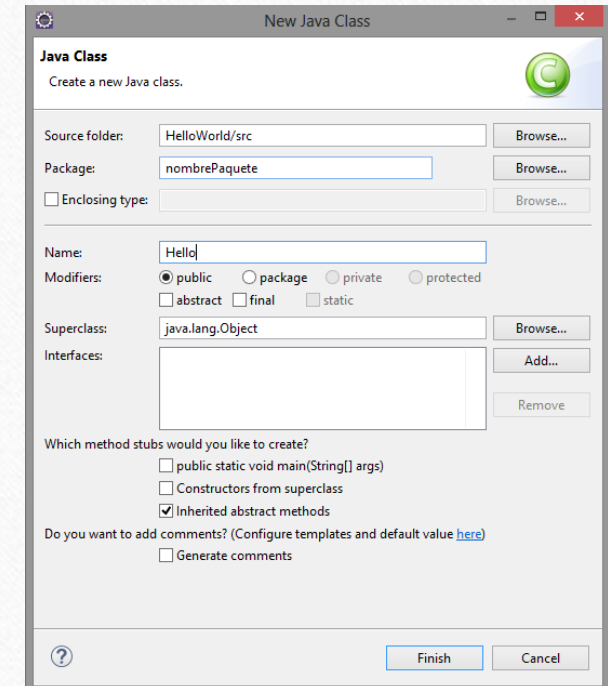
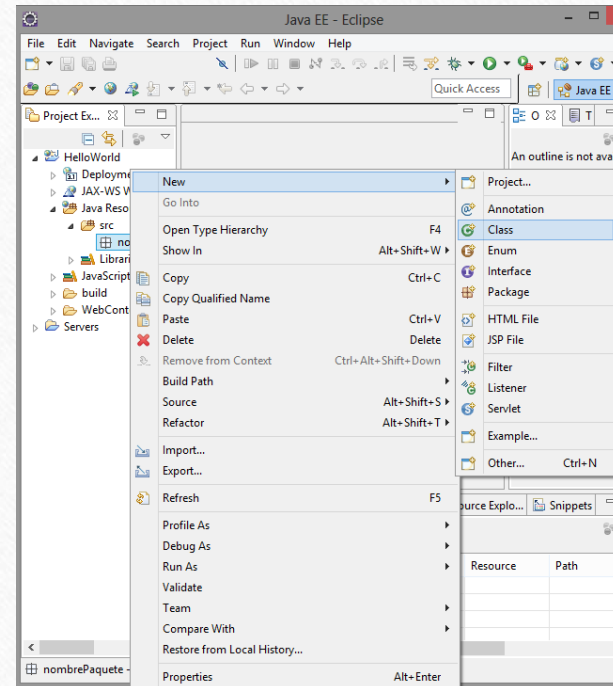
- We create a package inside the `JavaResources/src` name `packageName`.
- **VERY IMPORTANT:** Respect upper-case and lower-case conventions for Java, specially for the first letter





## 4. Package and Class Creation (ii)

- We create the class Hello and we introduce the code in the following slide.
- **VERY IMPORTANT: Respect upper-case and lower-case conventions for Java, specially for the first letter**



## 4. Package and Class Creation (iii)

```
package packageName;  
  
import javax.ws.rs.GET;  
import javax.ws.rs.Path;  
import javax.ws.rs.Produces;  
import javax.ws.rs.core.MediaType;
```

Use the package  
name you chose

```
@Path("/hello")  
public class Hello {  
  
    @GET  
    @Produces(MediaType.TEXT_PLAIN)  
    public String  
    sayPlainTextHello() {return "Hello  
    Plain";}  
  
}
```

Use the class  
name you chose

NOTE: Do not forget to save the file after pasting or  
modifying the code



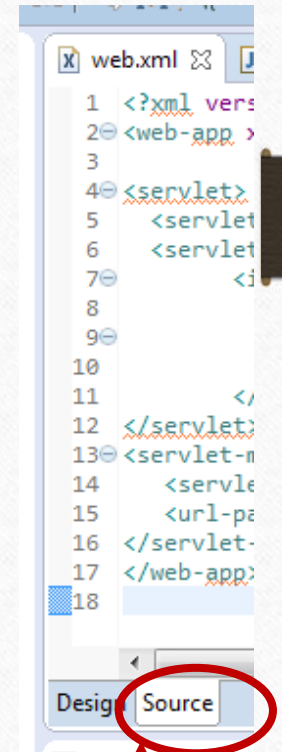
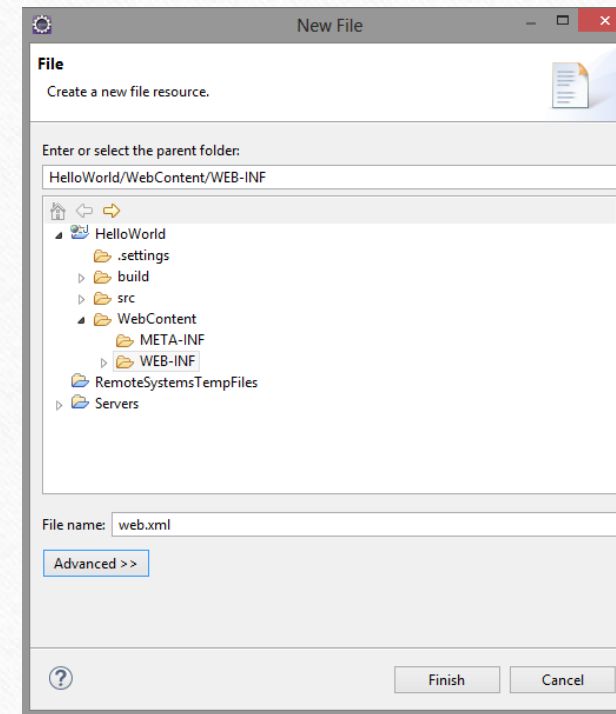
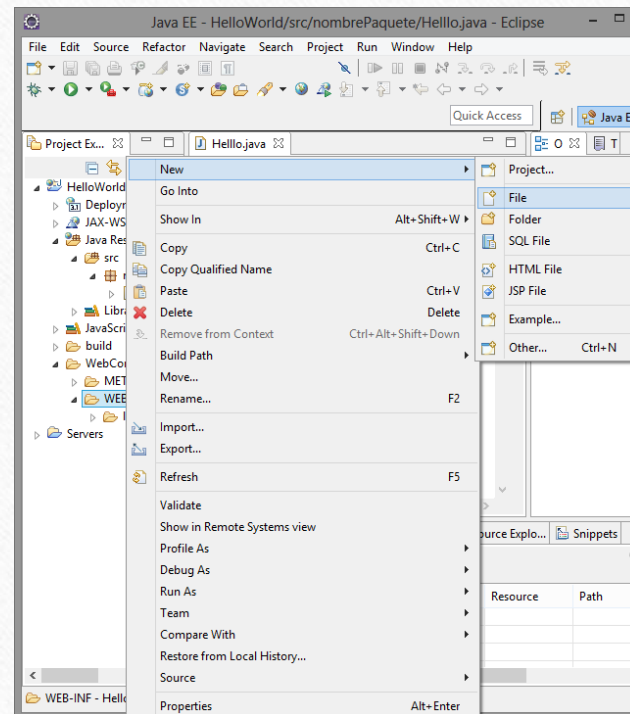
# Contents

---

1. Creation of a Tomcat Server in Eclipse
2. Dynamic Web project creation
3. Jersey library inclusion
4. Package and class creation
- 5. web.xml file creation**
6. Service deployment in the service
7. Testing it from Postman

# 5. web.xml File Creation (i)

- We create web.xml file in WebContent/WEB-INF
- The XML editor opens automatically. Click on “Source” in the tab below to see it in text mode.
- We copy the code in the following slide and we save it.





## 5. web.xml File Creation (ii)

---

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance" xmlns="http://java.sun.com/xml/ns/javaee"  
xmlns:web="http://java.sun.com/xml/ns/javaee/web-  
app_2_5.xsd"  
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"  
id="WebApp_ID" version="3.0">
```

## 5. web.xml File Creation (iii)

---

```
<servlet>
  <servlet-name>My REST service </servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-
class>
  <init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
    <param-value>
      packageName, com.fasterxml.jackson.jaxrs.json</param-value>
  </init-param>
</servlet>
```

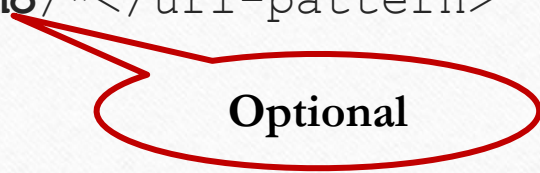
**Replace by  
your package  
name**



## 5. web.xml File Creation (iv)

---

```
<servlet-mapping>  
  <servlet-name>My REST service </servlet-name>  
  <url-pattern>/demo/*</url-pattern>  
</servlet-mapping>  
</web-app>
```



**Optional**

**NOTE:** depending on the operating system the quotation marks are copied wrongly. Revise them when you get an error.

**NOTE:** You can create first the class and then the web.xml or vice-versa.

# Contents

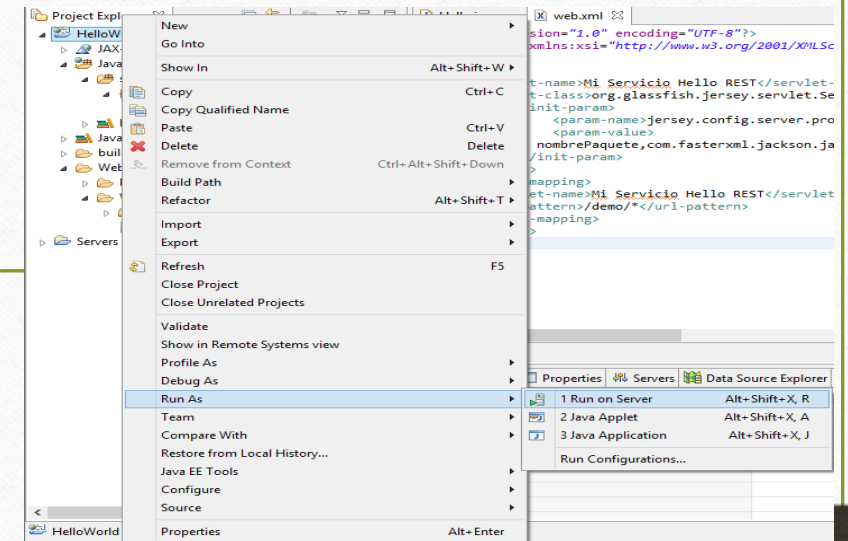
---

1. Creation of a Tomcat Server in Eclipse
2. Dynamic Web project creation
3. Jersey library inclusion
4. Package and class creation
5. web.xml file creation
- 6. Service API deployment**
7. Testing it from Postman



# 6. Service API Deployment

- We deploy the service in the Tomcat server previously created: Right click on the project → Run as → Run on Server → We select the server created → Finish
- We will be asked to restart the server → we click on OK
- If everything worked fine in the deployment, the service will appear as synchronized.
- Otherwise, we will have to click in the console to see the error.
- Do not worry if you get a 404 error in the browser.
- NOTE: To see the console window, problems or error log: Window → Show View → General → *corresponding option*



# Contents

---

1. Creation of a Tomcat Server in Eclipse
2. Dynamic Web project creation
3. Jersey library inclusion
4. Package and class creation
5. web.xml file creation
6. Service deployment in the service
7. Testing it from Postman



# 7. Testing it from Postman

- We can use any rest client, for instance Postman

- Fill in the address to invoke:

http://localhost:8080/HelloWorld/demo/hello

Included in web.xml

Class annotation

Your project name

The screenshot shows the Postman interface for a REST client. The top bar displays the request method as 'GET' and the URL as 'http://localhost:8080/HelloWorld/demo/hello'. The 'Send' button is visible in the top right corner. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is selected, and it shows a message: 'This request does not have a body'. Below the body tab, there are radio buttons for 'none', 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', and 'GraphQL'. The bottom section of the interface shows the response details, including 'Status: 200 OK', 'Time: 2.56 s', and 'Size: 161 B'. The response body is displayed as '1 Hello Plain'.

# Support Bibliography and References

---

- Developing RESTful Services with JAX-RS 2.0, WebSockets, and JSON. By: Masoud Kalali; Bhakti Mehta. Publisher: Packt Publishing Pub. Date: October 15, 2013. Print ISBN-13: 978-1-78217-812-5
- REST with Java (JAX-RS) using Jersey - Tutorial Lars Voguel  
<https://www.vogella.com/tutorials/REST/article.html>
- Postman API Client - Postman Inc.  
<https://www.postman.com/product/api-client/>