JAVA EE COURSE

SMS WITH REST WEB SERVICES (JAX-RS)

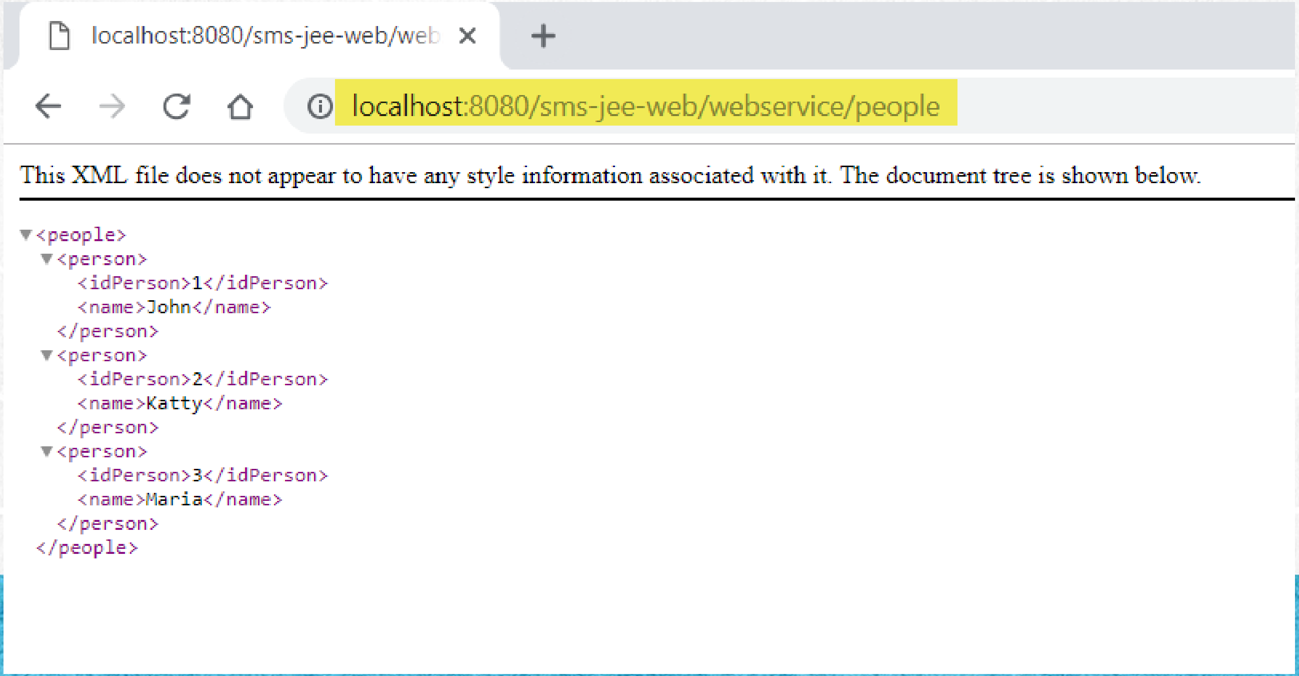
By the expert: Eng. Ubaldo Acosta

JAVA EE COURSE

www.globalmentoring.com.mx

EXERCISE OBJECTIVE

The objective of the exercise is to expose the methods listPeople, addPerson, modifyPerson, deletePerson of the EJB of the SMS project using Restful Web Services with the help of the JAX-RS API. The result is shown below:

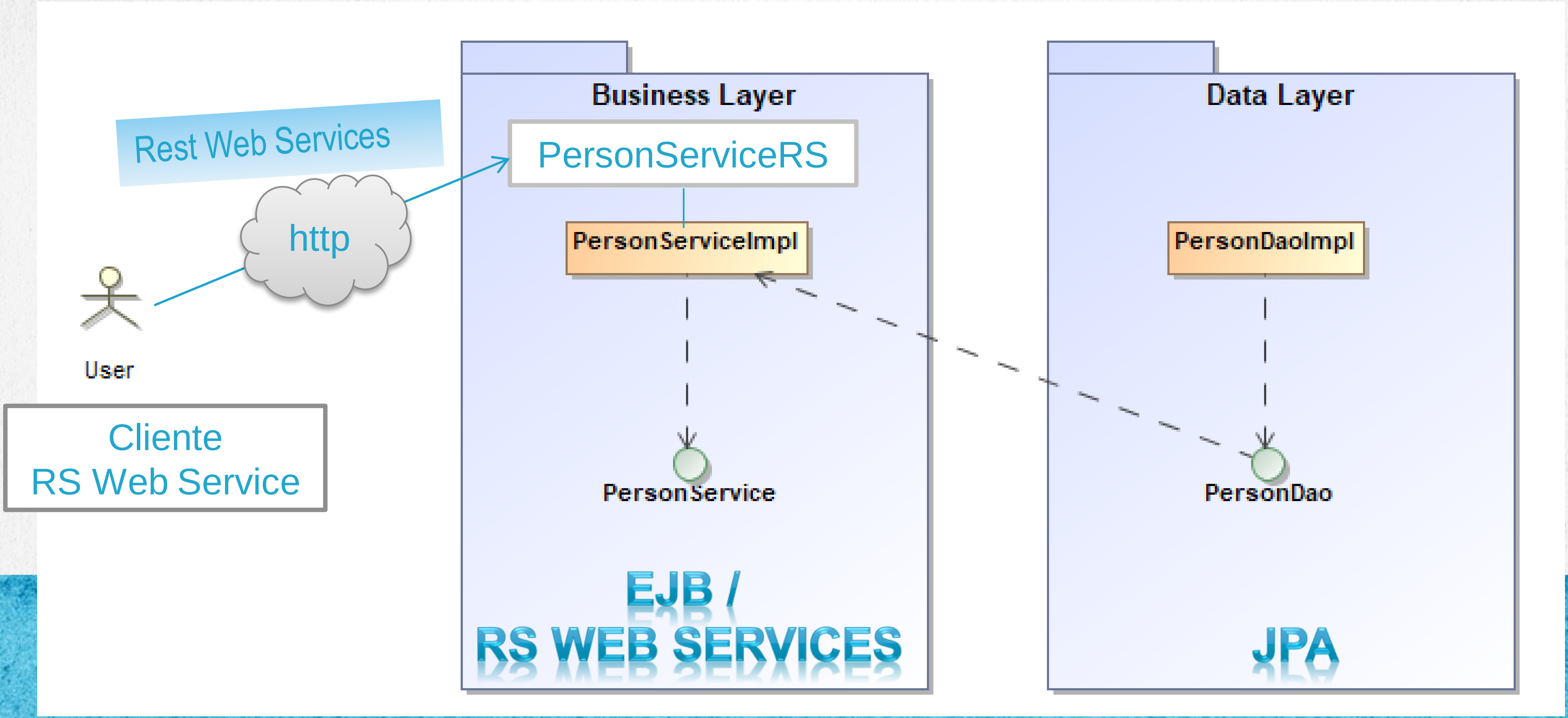

localhost:8080/sms-jee-web/web
localhost:8080/sms-jee-web/webservice/people

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<people>
  ▼<person>
     <idPerson>1</idPerson>
     <name>John</name>
   </person>
  ▼<person>
     <idPerson>2</idPerson>
     <name>Katty</name>
   </person>
  ▼<person>
     <idPerson>3</idPerson>
     <name>Maria</name>
   </person>
 </people>
```

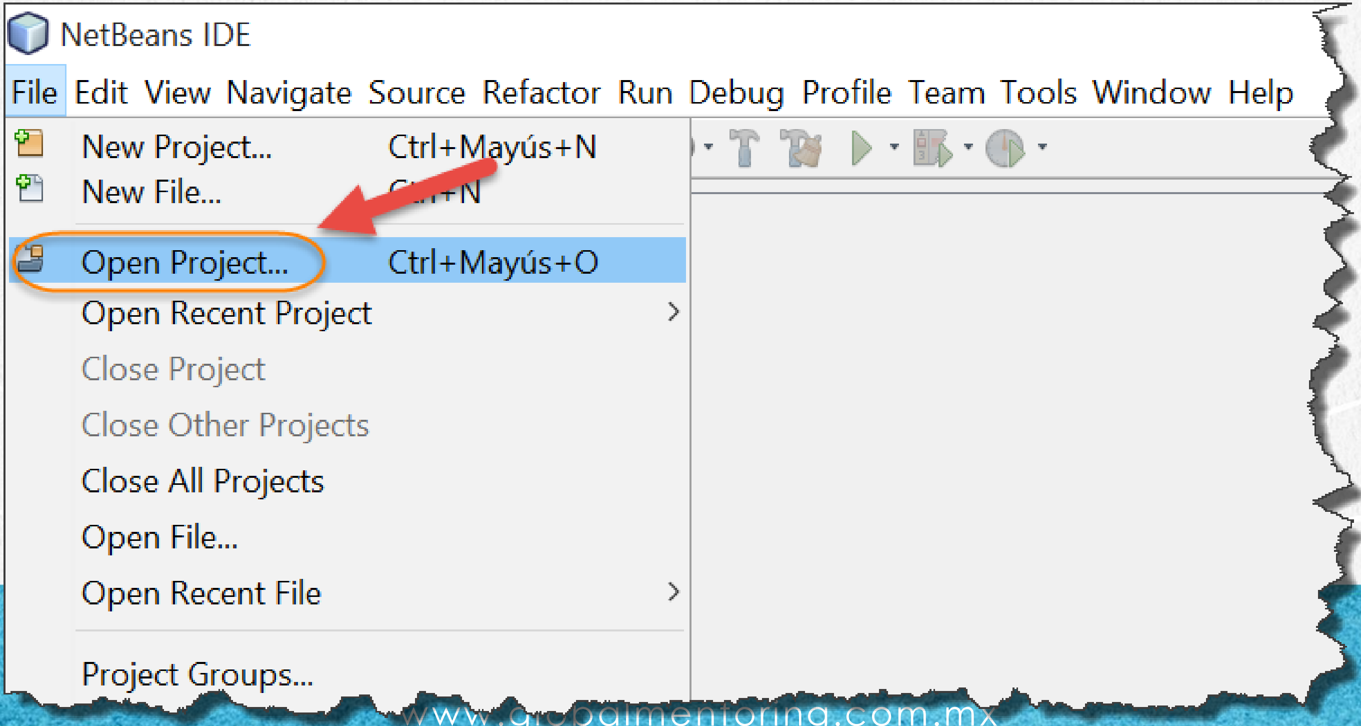
SMS ARCHITECTURE WITH WEB SERVICES

This is the Exercise Class Diagram, where you can see the Architecture of our System:



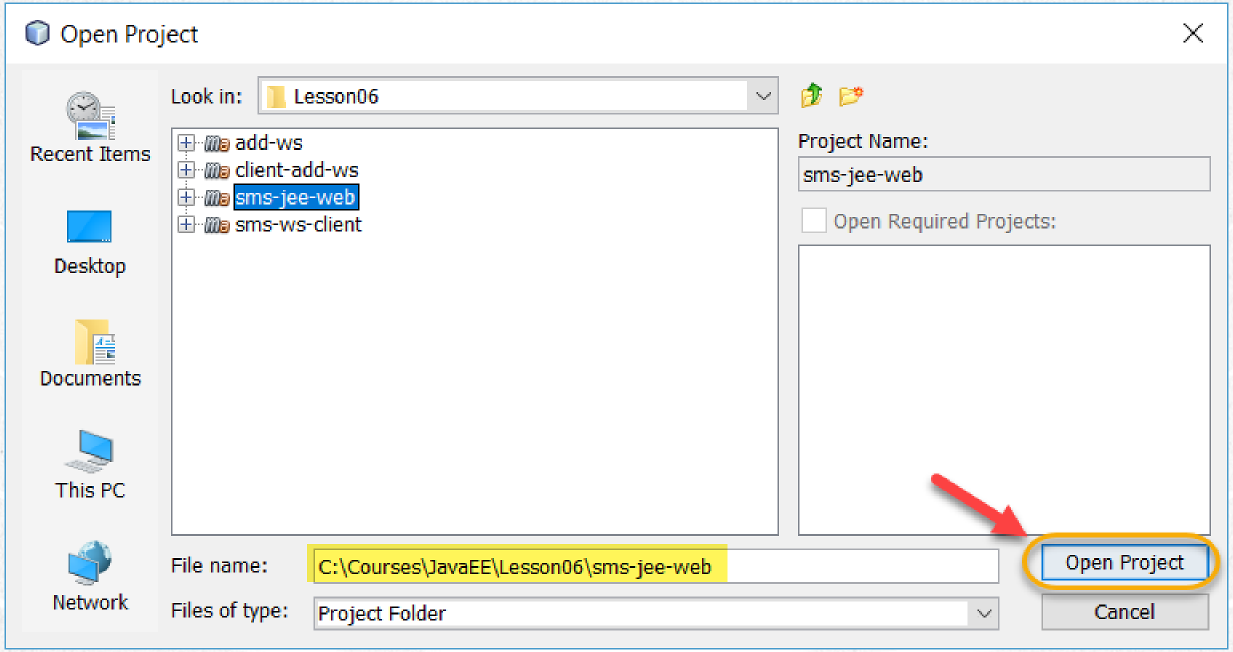
1. OPEN THE PROJECT

In case we do not have open the sms-jee-web project we open it:



1. OPEN THE PROJECT

In case we do not have open the sms-jee-web project we open it:



Look in: Lesson06

- add-ws
- client-add-ws
- sms-jee-web
- sms-ws-client

Project Name: sms-jee-web

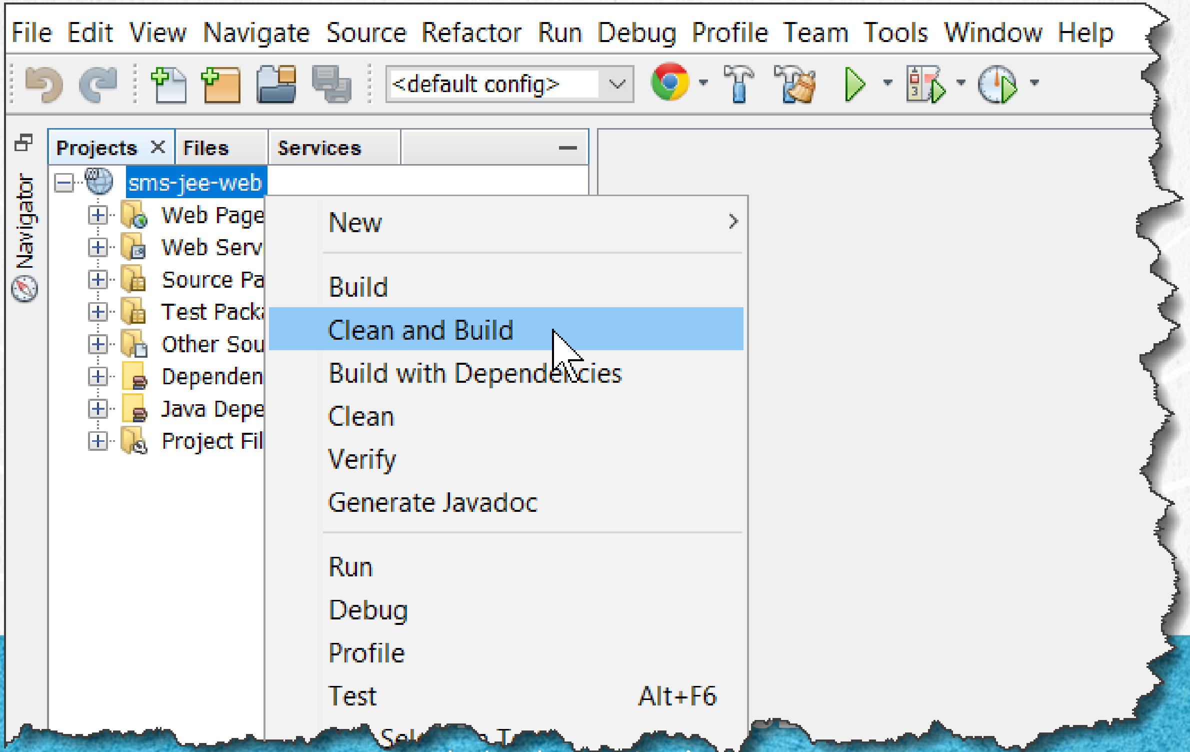
Open Required Projects:

File name: C:\Courses\JavaEE\Lesson06\sms-jee-web

Files of type: Project Folder

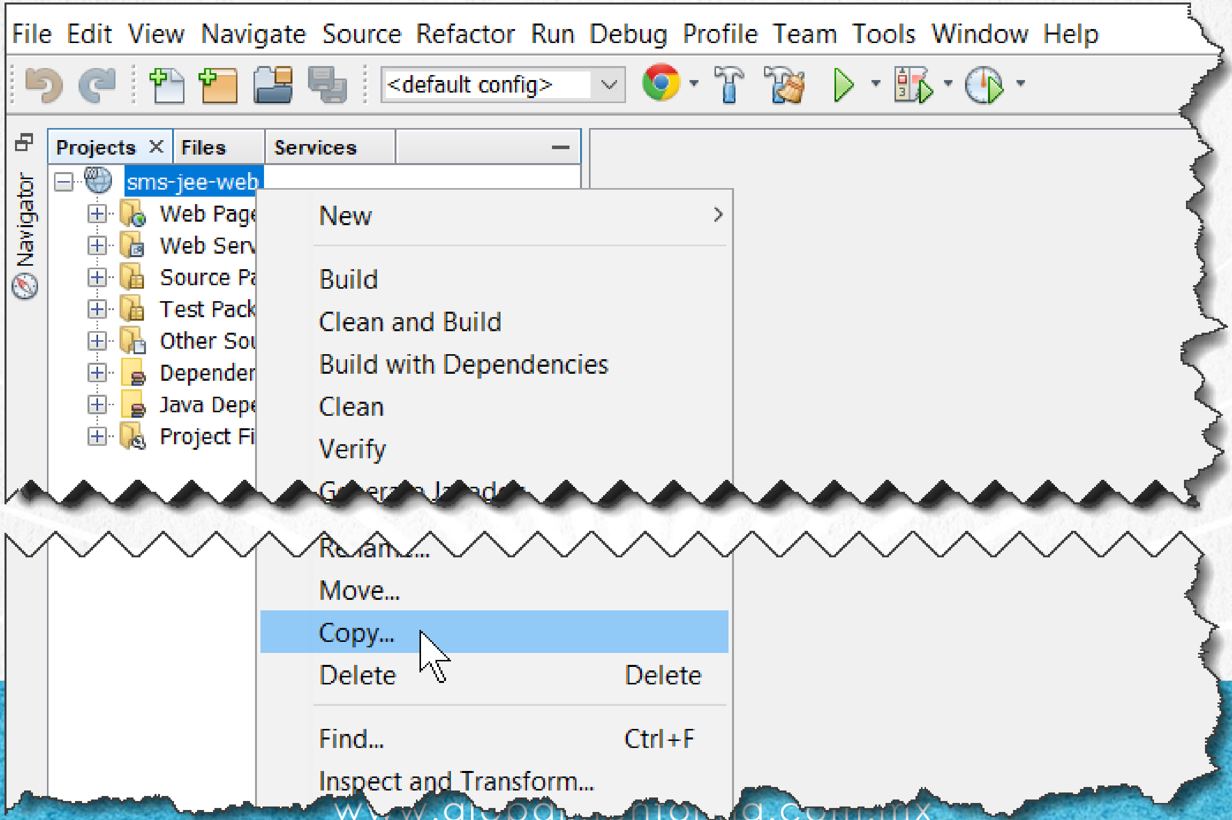
1. OPEN THE PROJECT

We wait for you to fully load the project. In case the project makes a mistake, we make a Clean & Build so that all the files are shown, this step is optional:



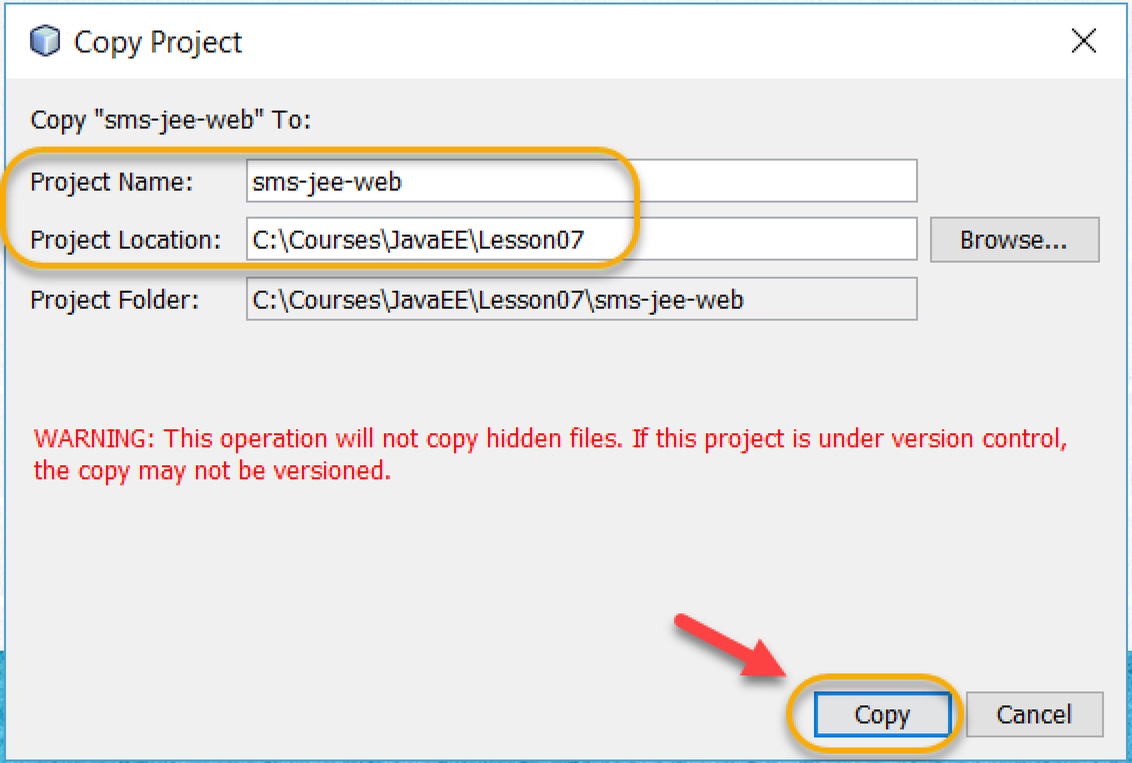
2. COPY THE PROJECT

We copy the project to put it in the new path:



2. COPY THE PROJECT

We copy the project to put it in the new path:



Copy Project

Copy "sms-jee-web" To:

Project Name: sms-jee-web

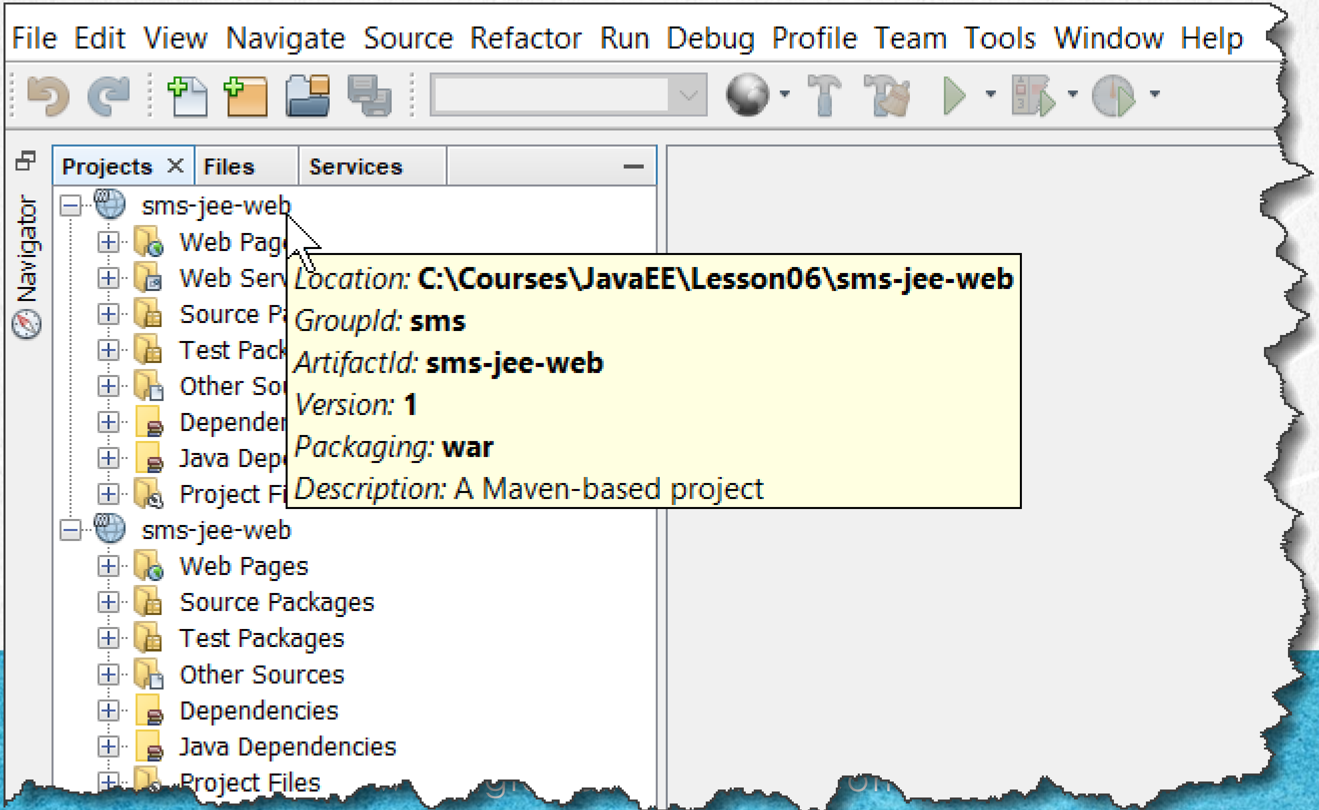
Project Location: C:\Courses\JavaEE\Lesson07

Project Folder: C:\Courses\JavaEE\Lesson07\sms-jee-web

WARNING: This operation will not copy hidden files. If this project is under version control, the copy may not be versioned.

3. CLOSE THE PROJECT

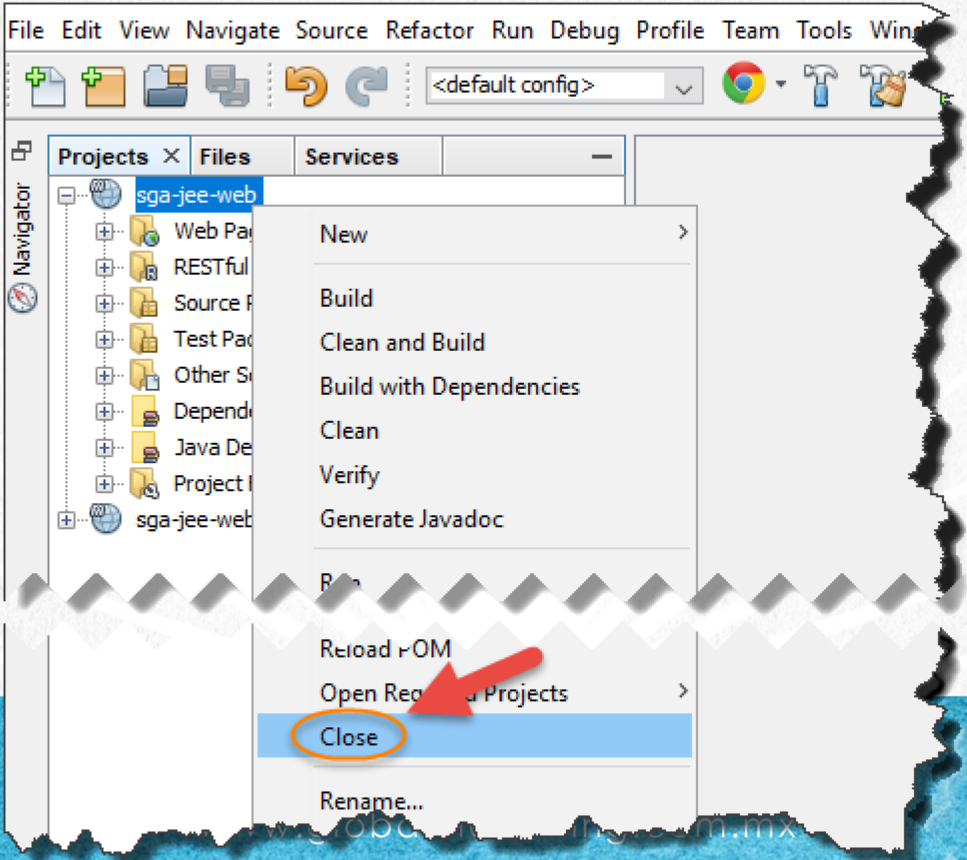
We closed the previous project, we identified it by positioning ourselves on the project:



Location: **C:\Courses\JavaEE\Lesson06\sms-jee-web**
GroupId: **sms**
ArtifactId: **sms-jee-web**
Version: **1**
Packaging: **war**
Description: A Maven-based project

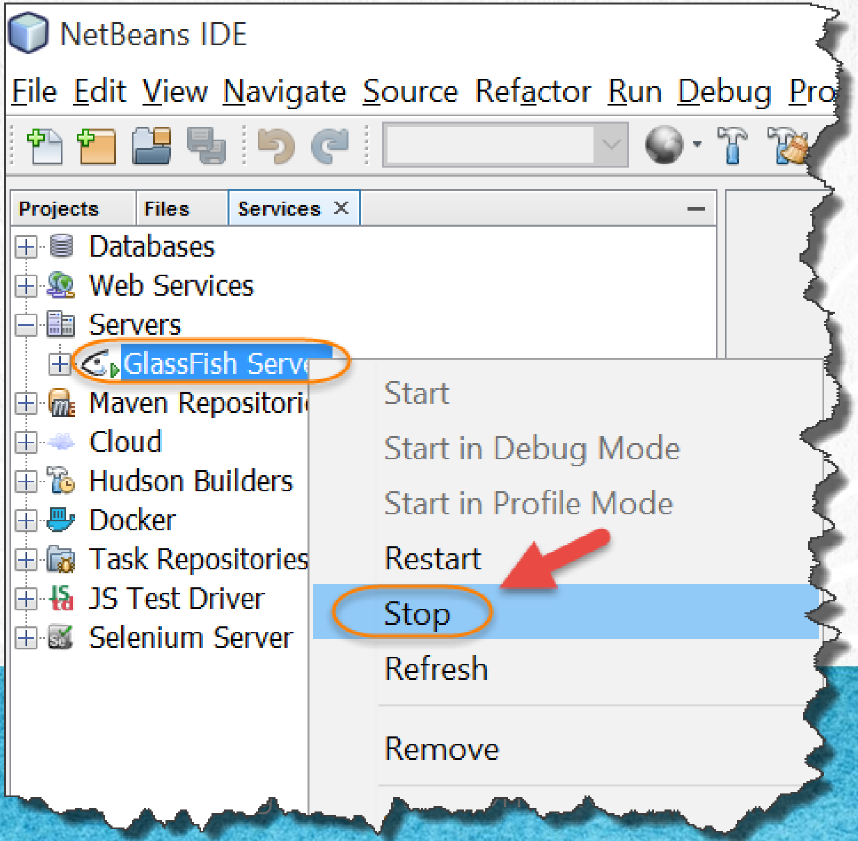
3. CLOSE THE PROJECT

We closed the previous project and left only the new one:



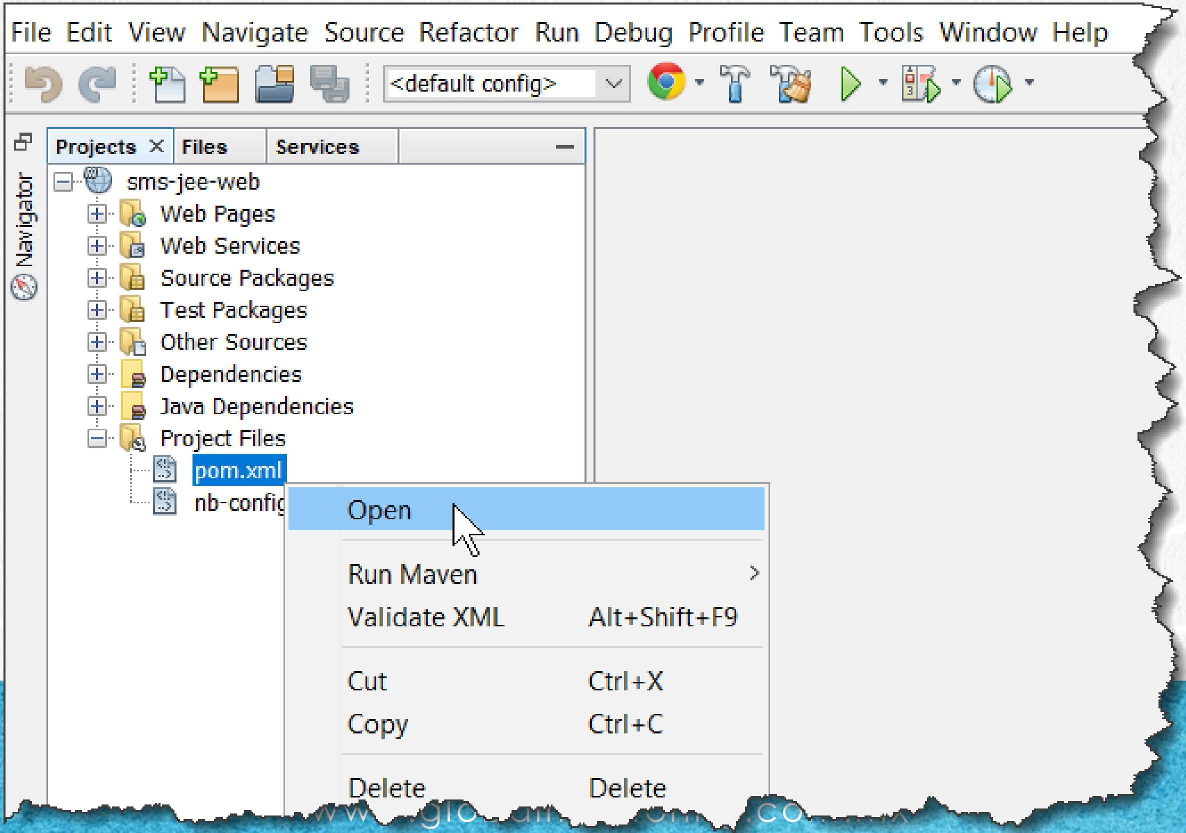
4. STOP GLASSFISH

We stop the Glassfish server:



5. MODIFY THE POM.XML FILE

We add the jersey-client.jar library to the pom.xml file:



5. MODIFY THE FILE

pom.xml:

Click to download

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>sms</groupId>
    <artifactId>sms-jee-web</artifactId>
    <version>1</version>
    <packaging>war</packaging>
    <name>sms-jee-web</name>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>8.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.primefaces</groupId>
            <artifactId>primefaces</artifactId>
            <version>6.2</version>
        </dependency>
```

5. MODIFY THE FILE

pom.xml:

Click to download

```
        <dependency>
            <groupId>org.primefaces.themes</groupId>
            <artifactId>all-themes</artifactId>
            <version>1.0.10</version>
        </dependency>
         <dependency>
            <groupId>org.glassfish.jersey.core</groupId>
            <artifactId>jersey-client</artifactId>
            <version>2.27</version>
        </dependency>
    </dependencies>
    <repositories>
        <repository>
            <id>prime-repo</id>
            <name>PrimeFaces Maven Repository</name>
            <url>http://repository.primefaces.org</url>
            <layout>default</layout>
        </repository>
    </repositories>
```

5. MODIFY THE FILE

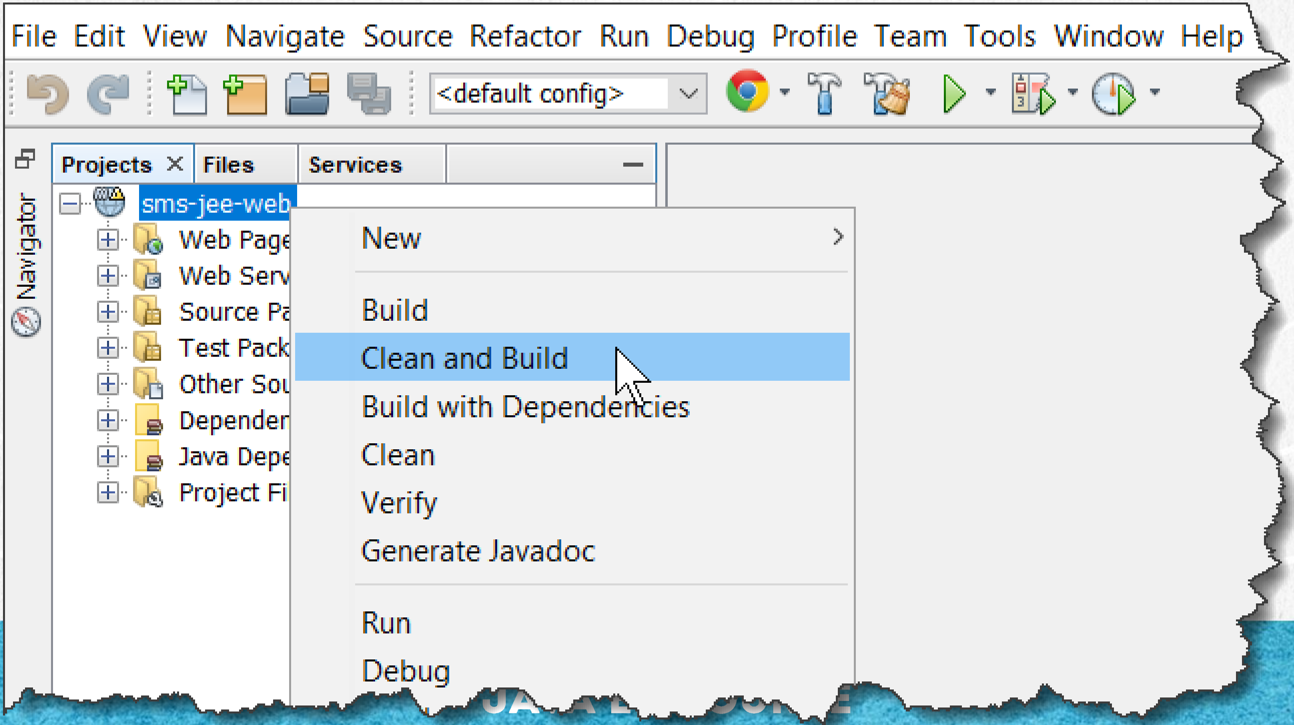
pom.xml:

Click to download

```
     <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.6</version>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

5. EXECUTE CLEAN & BUILD

We do a clean & build to the project to download the pending libraries:



6. MODIFY A JAVA FILE

- We modify the Person domain class, adding the following annotation to the beginning of the class (Note: In case you don’t need XML data but only JSON data, don’t add the @XmlRoolElement annotation):

```
@XmlRootElement
```

- And add a new constructor to accept the idPerson.

6. MODIFY THE FILE

Person.java:

Click to download

```
package sms.domain;

import java.io.Serializable;
import javax.persistence.*;
import javax.xml.bind.annotation.*;

@Entity
@NamedQueries({
    @NamedQuery(name = "Person.findAll", query = "SELECT p FROM Person p ORDER BY p.idPerson")})
@Table(name = "person")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement
public class Person implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_person")
    private int idPerson;

    private String name;

    public Person() {
    }

    public Person(int idPerson) {
        this.idPerson = idPerson;
    }
```

6. MODIFY THE FILE

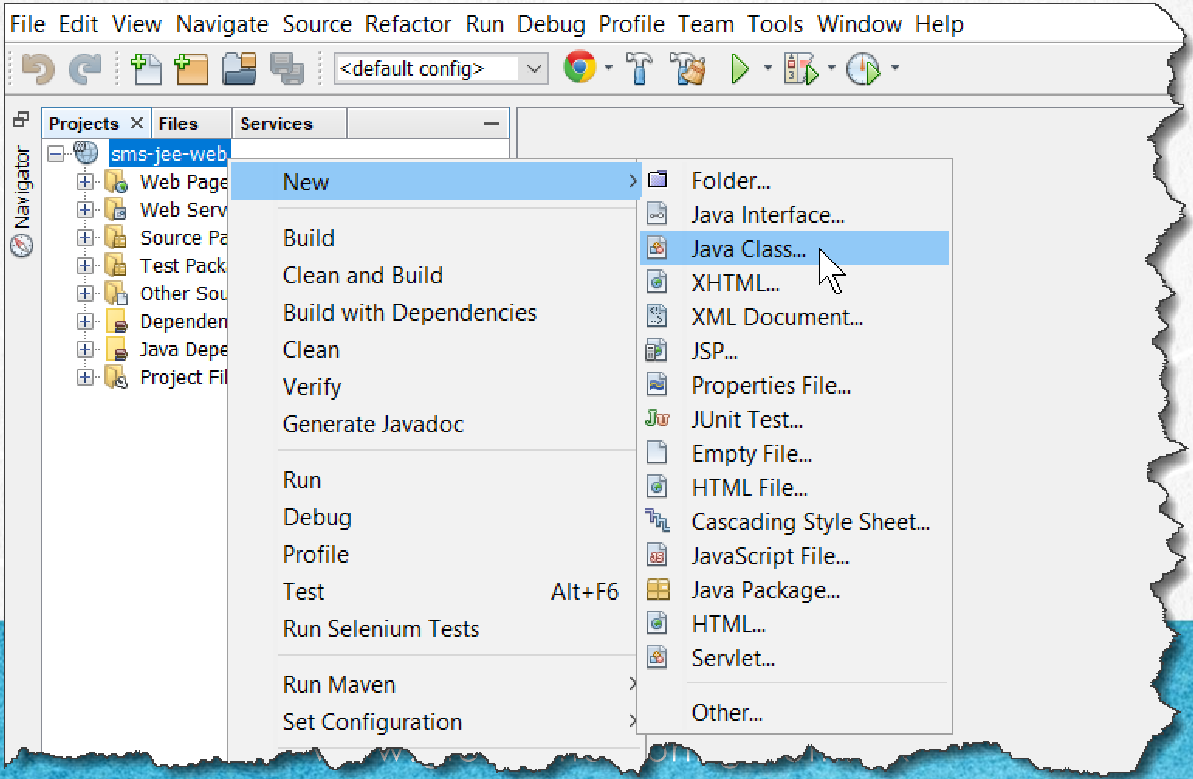
Person.java:

Click to download

```
    public Person(int idPersona, String name) {
        this.idPerson = idPersona;
        this.name = name;
    }

    public int getIdPerson() {
        return idPerson;
    }

    public void setIdPerson(int idPerson) {
        this.idPerson = idPerson;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Person{" + "idPerson=" + idPerson + ", name=" + name + '}';
    }
}
```

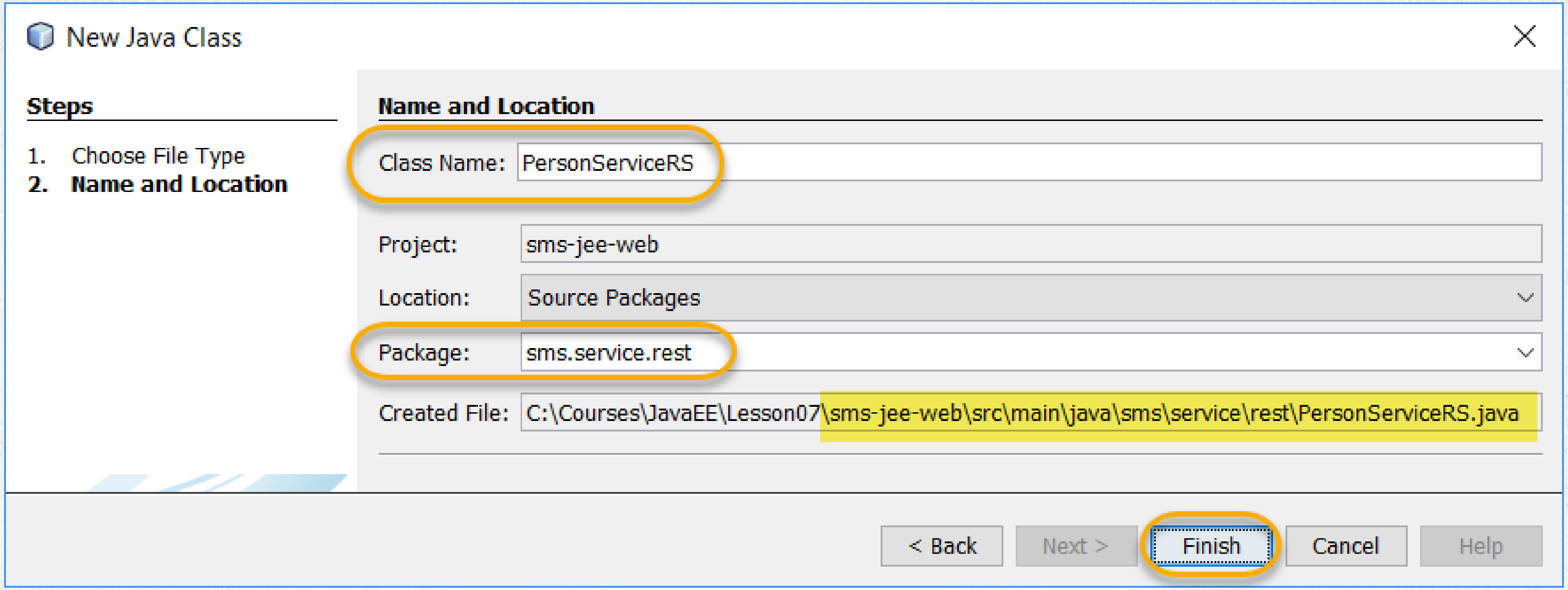
7. CREATE A JAVA FILE

We create the PersonServiceRS.java class to expose the methods of listing, adding, modifying and deleting People via Rest Web Services:



7. CREATE A JAVA FILE

We create the PersonServiceRS.java class:



New Java Class

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: PersonServiceRS

Project: sms-jee-web

Location: Source Packages

Package: sms.service.rest

Created File: C:\Courses\JavaEE\Lesson07\sms-jee-web\src\main\java\sms\service\rest\PersonServiceRS.java

8. MODIFY THE CODE

PersonServiceRS.java:

Click to download

```
package sms.service.rest;

import java.util.List;
import javax.ejb.Stateless;
import javax.inject.Inject;
import javax.ws.rs.*;
import javax.ws.rs.core.*;
import javax.ws.rs.core.Response.Status;
import sms.domain.Person;
import sms.service.PersonService;

@Path("/people")
@Stateless
public class PersonServiceRS {

    @Inject
    private PersonService personService;

    @GET
    @Produces(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Person> listPeople() {
        return personService.listPeople();
    }
```

8. MODIFY THE CODE

PersonServiceRS.java:

Click to download

```
    @GET
    @Produces(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Path("{id}") //refers to /people/{id}
    public Person findPerson(@PathParam("id") int id) {
        return personService.findPerson(new Person(id));
    }

    @POST
    @Consumes(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Produces(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Response addPerson(Person person) {
        try {
            personService.addPerson(person);
            return Response.ok().entity(person).build();
        } catch (Exception e) {
            System.out.println("Error:" + e.getMessage());
            return Response.status(Status.INTERNAL_SERVER_ERROR).build();
        }
    }
```

8. MODIFY THE CODE

PersonServiceRS.java:

Click to download

```
@PUT
@Consumes(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
@Produces(value={MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
@Path("{id}")
public Response modifyPerson(@PathParam("id") int id, Person modifiedPerson) {
    try {
        Person person = personService.findPerson(new Person(id));
        if (person != null) {
            personService.modifyPerson(modifiedPerson);
            return Response.ok().entity(modifiedPerson).build();
        } else {
            return Response.status(Status.NOT_FOUND).build();
        }
    } catch (Exception e) {
        System.out.println("Error:" + e.getMessage());
        return Response.status(Status.INTERNAL_SERVER_ERROR).build();
    }
}
```

8. MODIFY THE CODE

PersonServiceRS.java:

Click to download

```
    @DELETE
    @Path("{id}")
    public Response deletePerson(@PathParam("id") int id) {
        try {
            personService.deletePerson(new Person(id));
            return Response.ok().build();
        } catch (Exception e) {
            System.out.println("Error:" + e.getMessage());
            return Response.status(404).build();
        }
    }
}
```

9. MODIFY A JAVA FILE

We modified the web.xml file. We configure the Jersey Servlet, adding the following configuration:

```
    <servlet>
        <servlet-name>JerseyWebApplication</servlet-name>
        <servlet-class>
            org.glassfish.jersey.servlet.ServletContainer
        </servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>sms.service.rest</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>JerseyWebApplication</servlet-name>
        <url-pattern>/webservice/*</url-pattern>
    </servlet-mapping>
```

9. MODIFY THE FILE

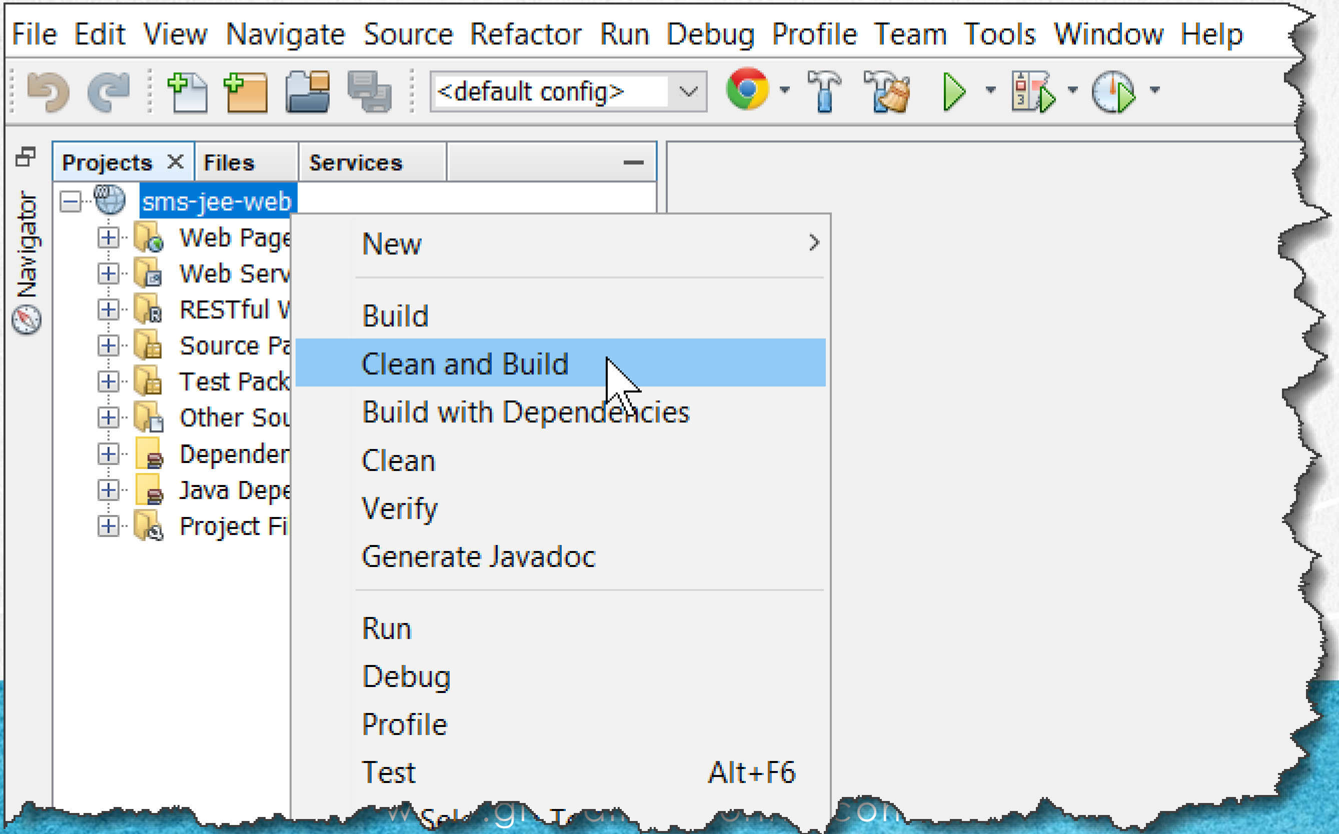
web.xml:

Click to dowload

```
<?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">
    <context-param>
        <param-name>primefaces.THEME</param-name>
        <param-value>cupertino</param-value>
    </context-param>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>JerseyWebApplication</servlet-name>
        <servlet-class>
            org.glassfish.jersey.servlet.ServletContainer
        </servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>sms.service.rest</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>JerseyWebApplication</servlet-name>
        <url-pattern>/webservice/*</url-pattern>
    </servlet-mapping>
</web-app>
```

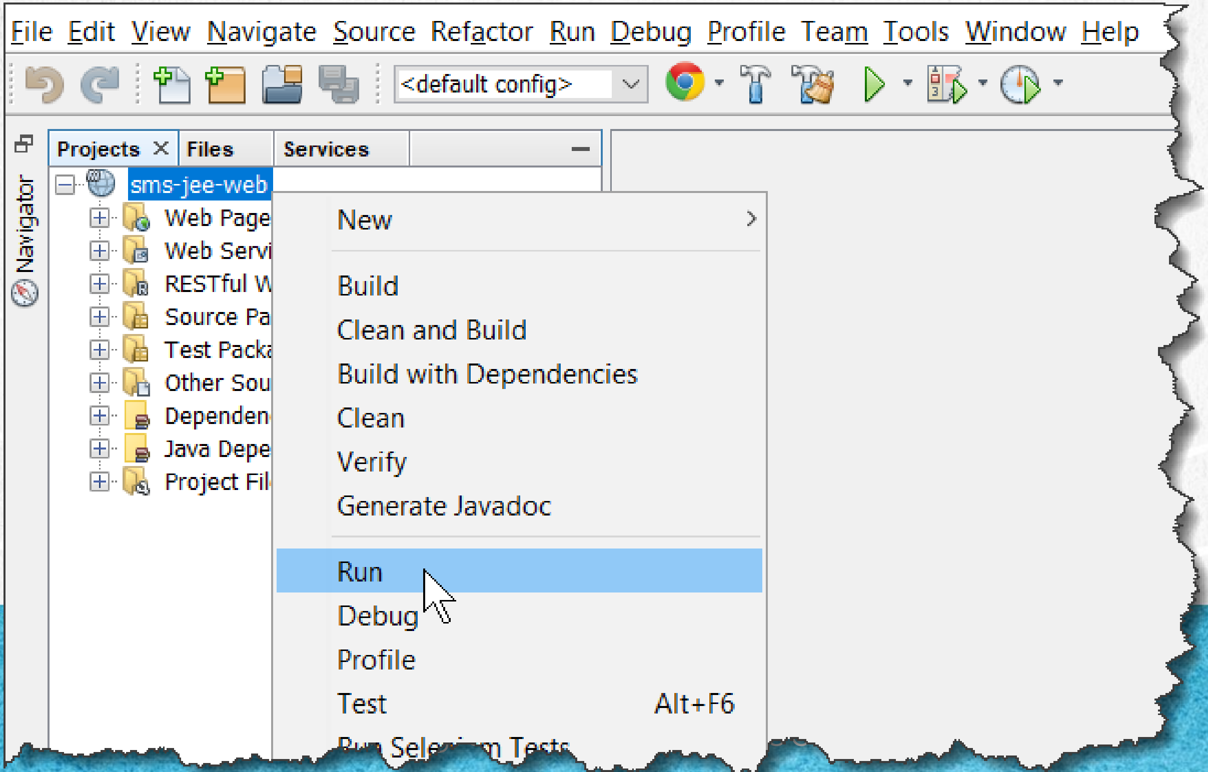
10. EXECUTE CLEAN /& BUILD

We do a clean & build to have the latest version of each file:



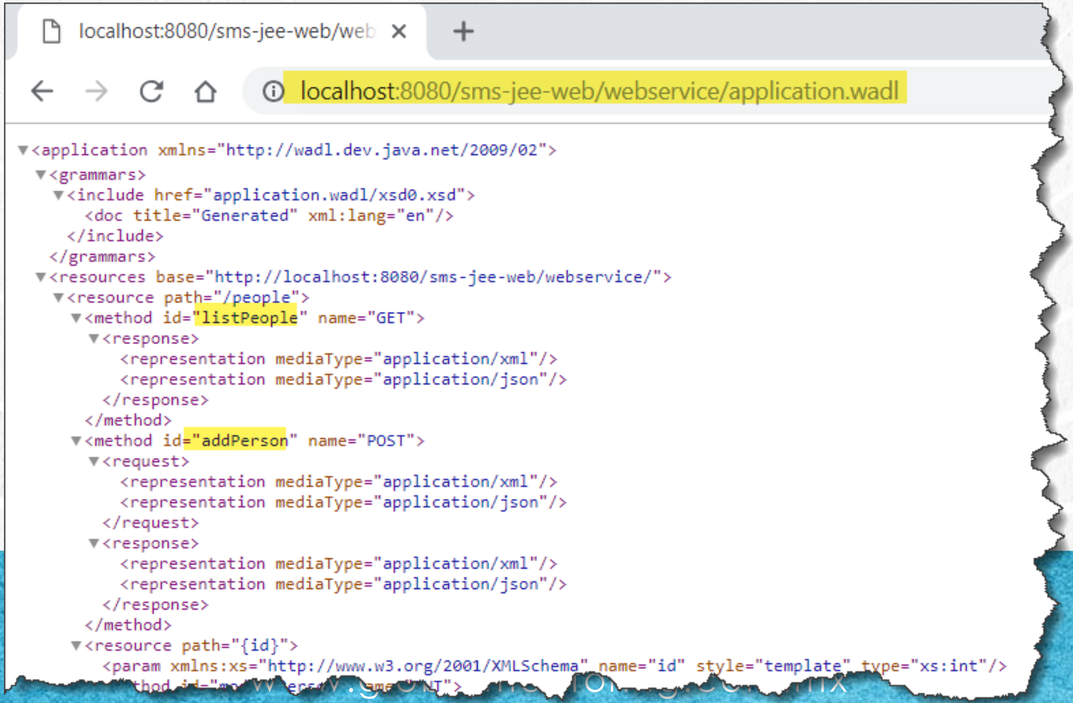
11. DEPLOY ON GLASSFISH

We run the application, and this will automatically deploy the application:



12. REVIEW OF REST WEB SERVICE

We execute the application. We verify that the Rest web service has been deployed as the url as follows:
http://localhost:8080/sms-jee-web/webservice/application.wadl

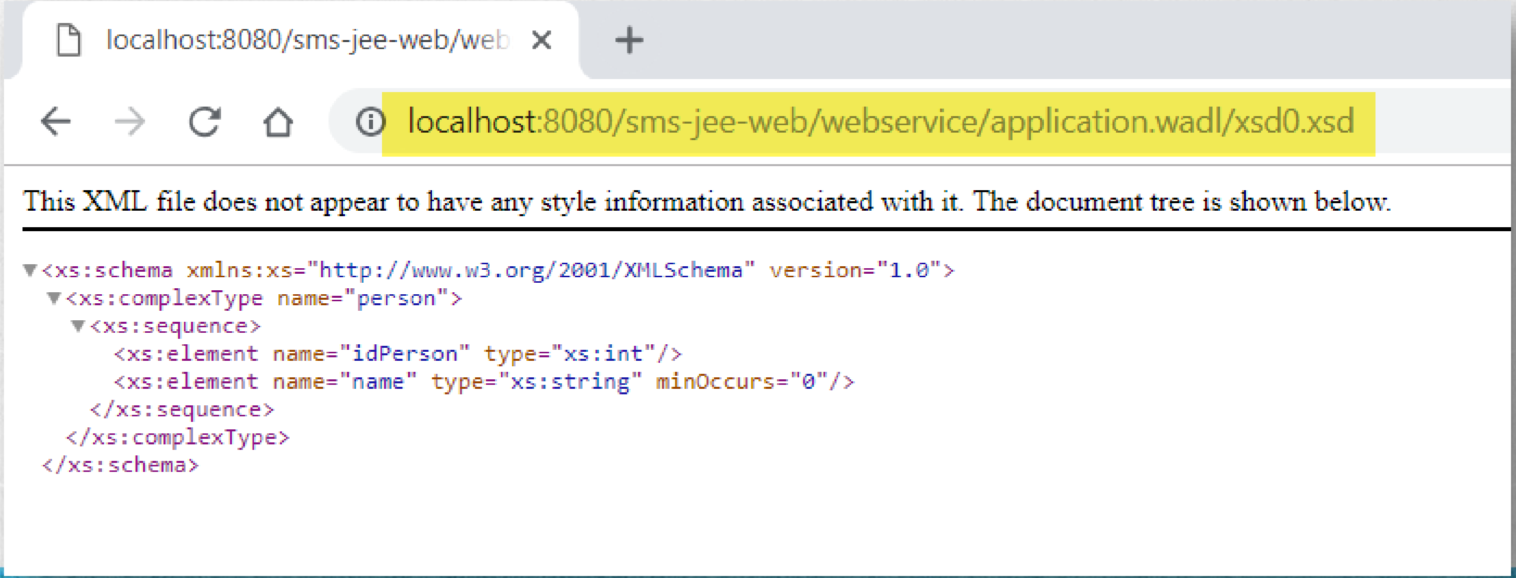

localhost:8080/sms-jee-web/web
localhost:8080/sms-jee-web/webservice/application.wadl

```
▼<application xmlns="http://wadl.dev.java.net/2009/02">
  ▼<grammars>
    ▼<include href="application.wadl/xsd0.xsd">
       <doc title="Generated" xml:lang="en"/>
     </include>
   </grammars>
  ▼<resources base="http://localhost:8080/sms-jee-web/webservice/">
    ▼<resource path="/people">
      ▼<method id="listPeople" name="GET">
        ▼<response>
           <representation mediaType="application/xml"/>
           <representation mediaType="application/json"/>
         </response>
       </method>
      ▼<method id="addPerson" name="POST">
        ▼<request>
           <representation mediaType="application/xml"/>
           <representation mediaType="application/json"/>
         </request>
        ▼<response>
           <representation mediaType="application/xml"/>
           <representation mediaType="application/json"/>
         </response>
       </method>
      ▼<resource path="{id}">
         <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="id" style="template" type="xs:int"/>
```

12. REVIEW OF REST WEB SERVICE

With the following URL we can verify the XSD of the Rest Web Service:

http://localhost:8080/sms-jee-web/webservice/application.wadl/xsd0.xsd

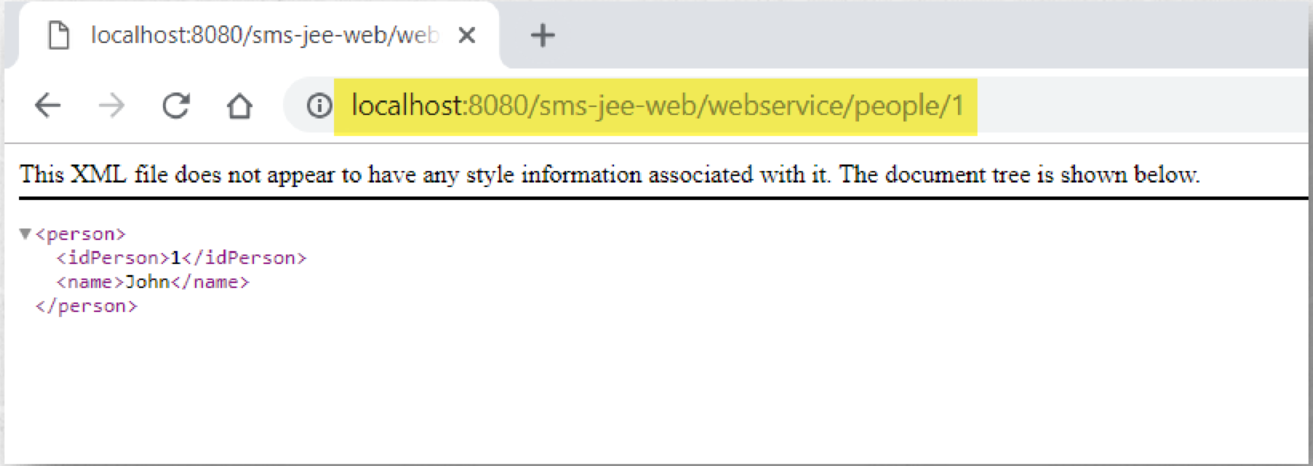

localhost:8080/sms-jee-web/web
localhost:8080/sms-jee-web/webservice/application.wadl/xsd0.xsd

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
  ▼<xs:complexType name="person">
    ▼<xs:sequence>
       <xs:element name="idPerson" type="xs:int"/>
       <xs:element name="name" type="xs:string" minOccurs="0"/>
     </xs:sequence>
   </xs:complexType>
 </xs:schema>
```

12. REVIEW OF REST WEB SERVICE

It is also possible to review directly from the Web browser, any of the published Web services. For example, providing a valid id_person whatever it is:
http://localhost:8080/sms-jee-web/webservice/people/1

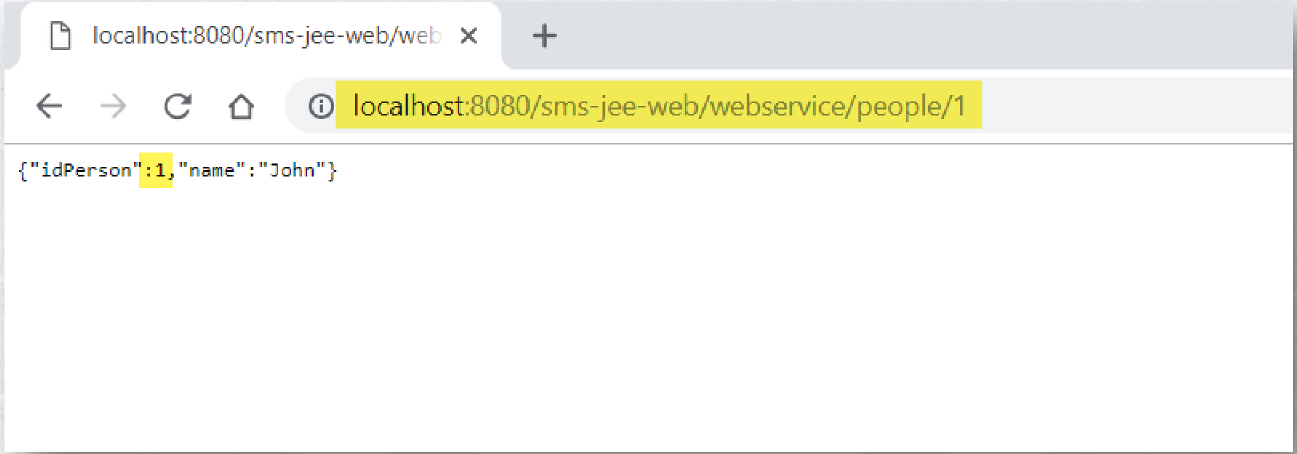

localhost:8080/sms-jee-web/web
localhost:8080/sms-jee-web/webservice/people/1

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<person>
   <idPerson>1</idPerson>
   <name>John</name>
 </person>
```

12. REVIEW OF REST WEB SERVICE

If you didn't add the @XmlRootElement to the entity class, you get JSON data:
http://localhost:8080/sms-jee-web/webservice/people/1

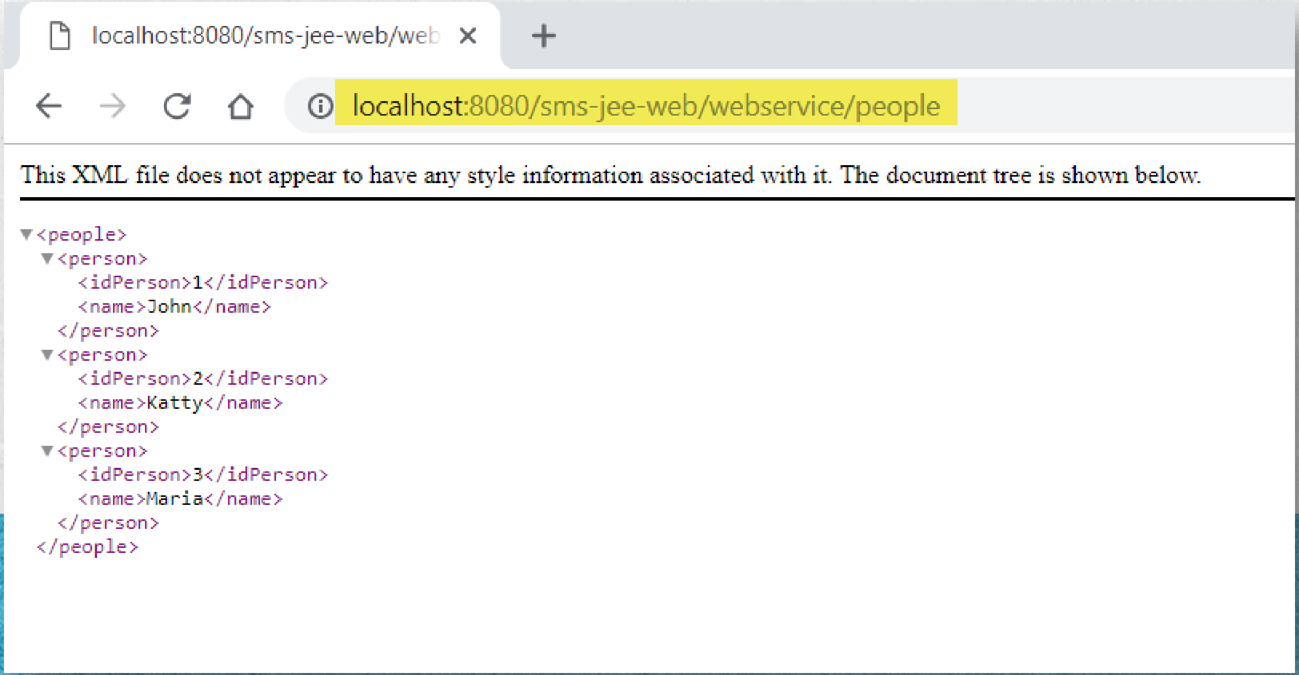

localhost:8080/sms-jee-web/web
localhost:8080/sms-jee-web/webservice/people/1

```
{"idPerson":1,"name":"John"}
```

12. REVIEW OF REST WEB SERVICE

We can also review all list of people using the web service:

http://localhost:8080/sms-jee-web/webservice/people

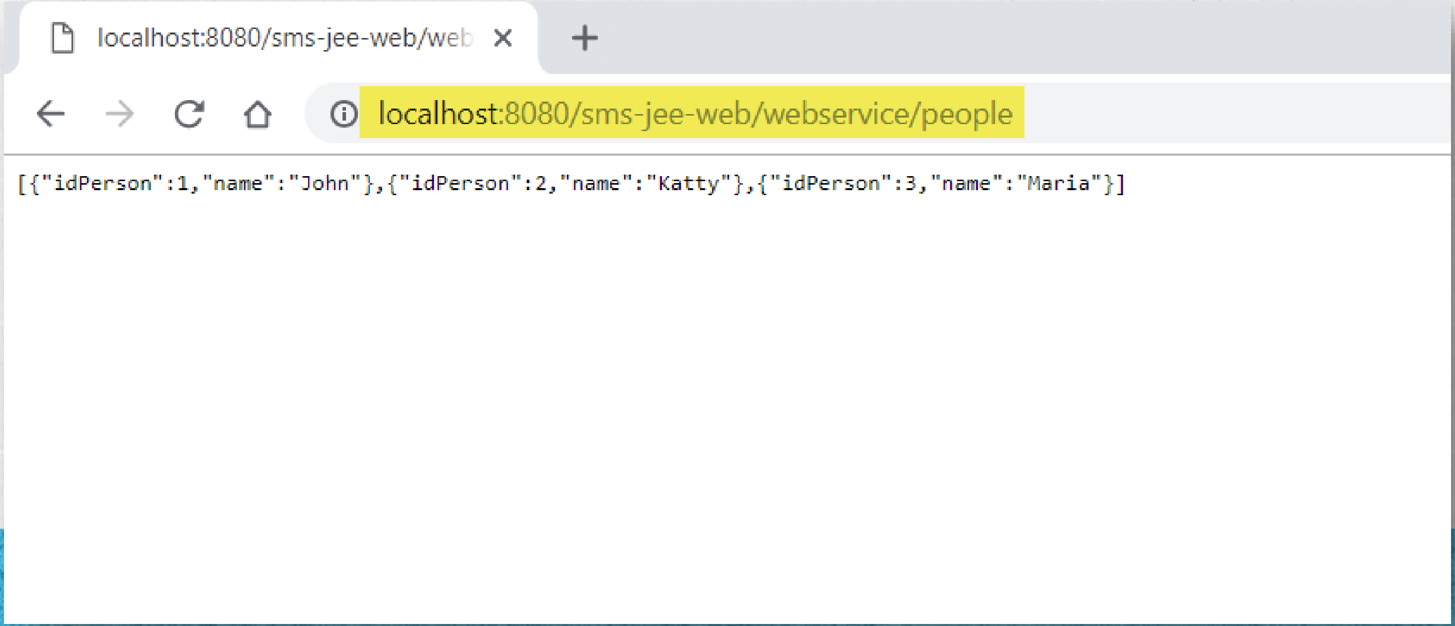

localhost:8080/sms-jee-web/web
localhost:8080/sms-jee-web/webservice/people

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<people>
  ▼<person>
     <idPerson>1</idPerson>
     <name>John</name>
   </person>
  ▼<person>
     <idPerson>2</idPerson>
     <name>Katty</name>
   </person>
  ▼<person>
     <idPerson>3</idPerson>
     <name>Maria</name>
   </person>
 </people>
```

11. REVIEW OF REST WEB SERVICE

If you didn't add the @XmlRootElement to the entity class, you get JSON data:
http://localhost:8080/sms-jee-web/webservice/people


localhost:8080/sms-jee-web/web
localhost:8080/sms-jee-web/webservice/people

```
[{"idPerson":1,"name":"John"},{"idPerson":2,"name":"Katty"},{"idPerson":3,"name":"Maria"}]
```

IN CASE OF PROBLEMS

- Stop and Start Glassfish
- Undeploy any application on Glassfish
- Clean & Build the application
- Run the Application
- Check the URL's to see if the web service is running

- If none of the previous steps worked, you can load the resolved project, which is 100% functional and re-execute the previous steps

EXERCISE CONCLUSION

With this exercise we have published the EJB methods using REST Web Services.

We create the class that exposes the methods, as well as perform the respective modifications, both in the respective Entity class, as well as in the configuration file web.xml

In the following exercise we will create the client that will consume the REST web services exposed in this exercise.

ONLINE COURSE

JAVA EE
JAKARTA EE

By: Eng. Ubaldo Acosta

Global Mentoring

JAVA ONLINE COURSES UNIVERSITY

JAVA EE COURSE

www.globalmentoring.com.mx