JAVA EE COURSE

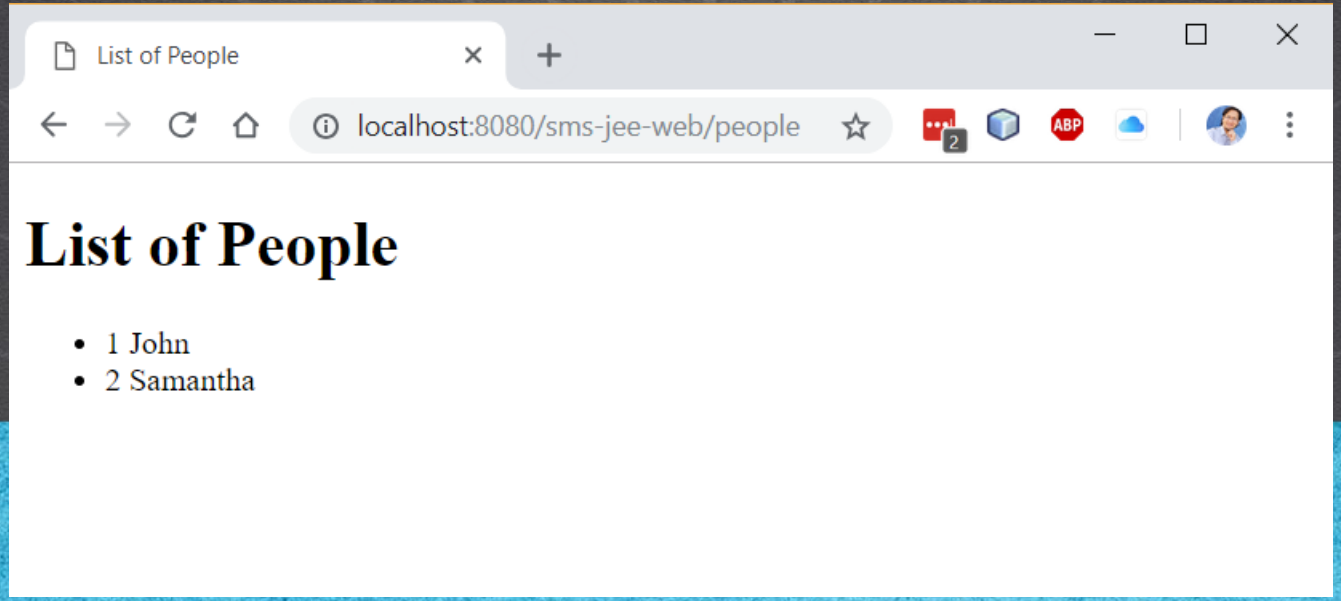
EXERCISE

SMS SYSTEM LOCAL EJB

EXERCISE OBJECTIVE

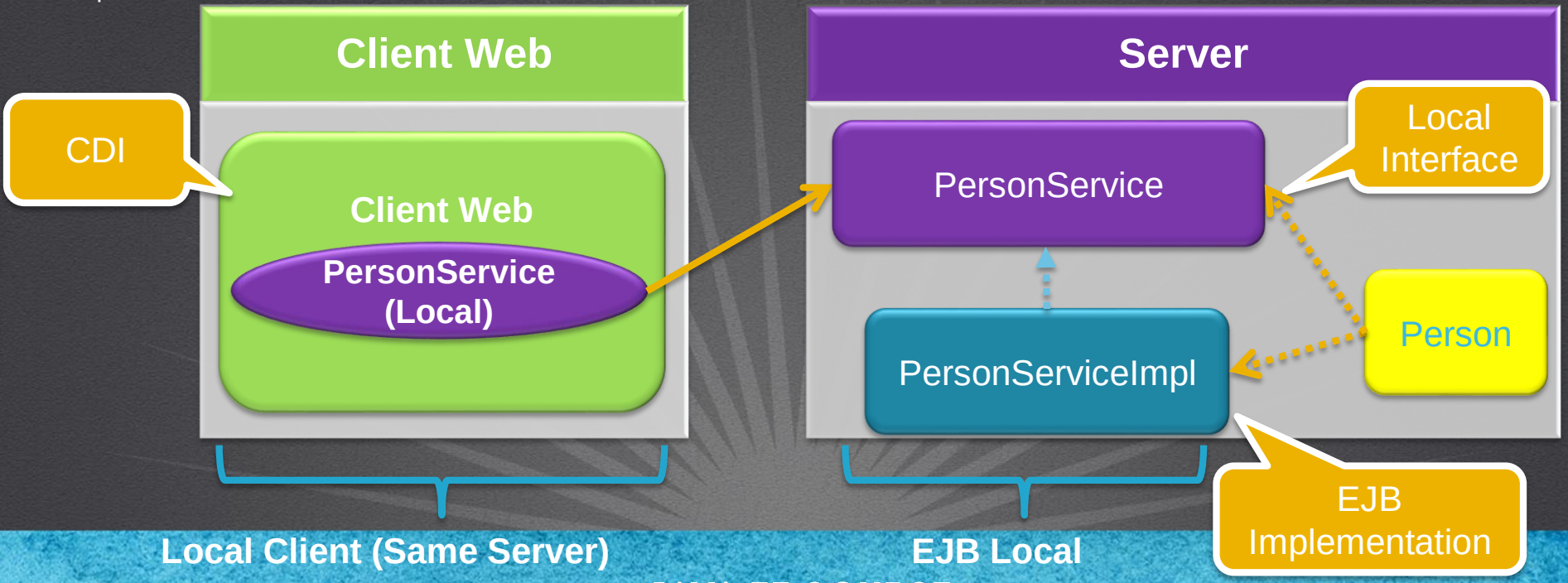
- The objective of the exercise is to add a Local EJB to our SMS project (Student Management System).

- At the end we must observe the following result:



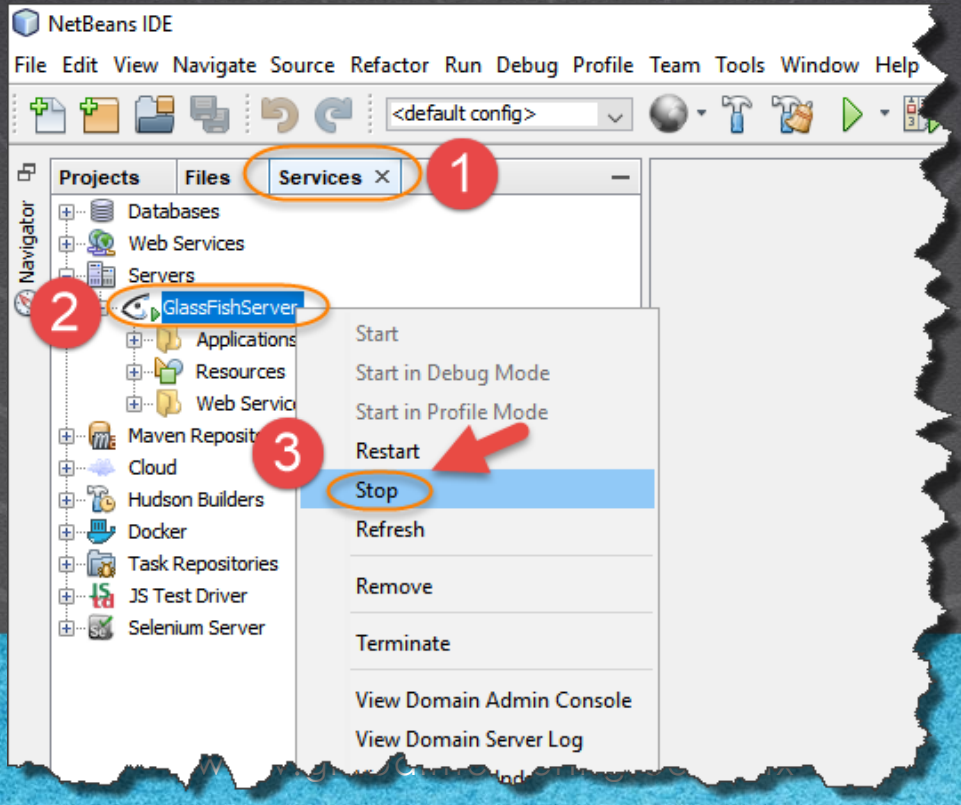
JAVA EE ARCHITECTURE

•In our architecture, we will add the Local interface of our EJB, since our Web components that we will create later will be in the same server, in this way we will avoid unnecessary remote calls. Both the remote interface and the local interface will expose the same methods:



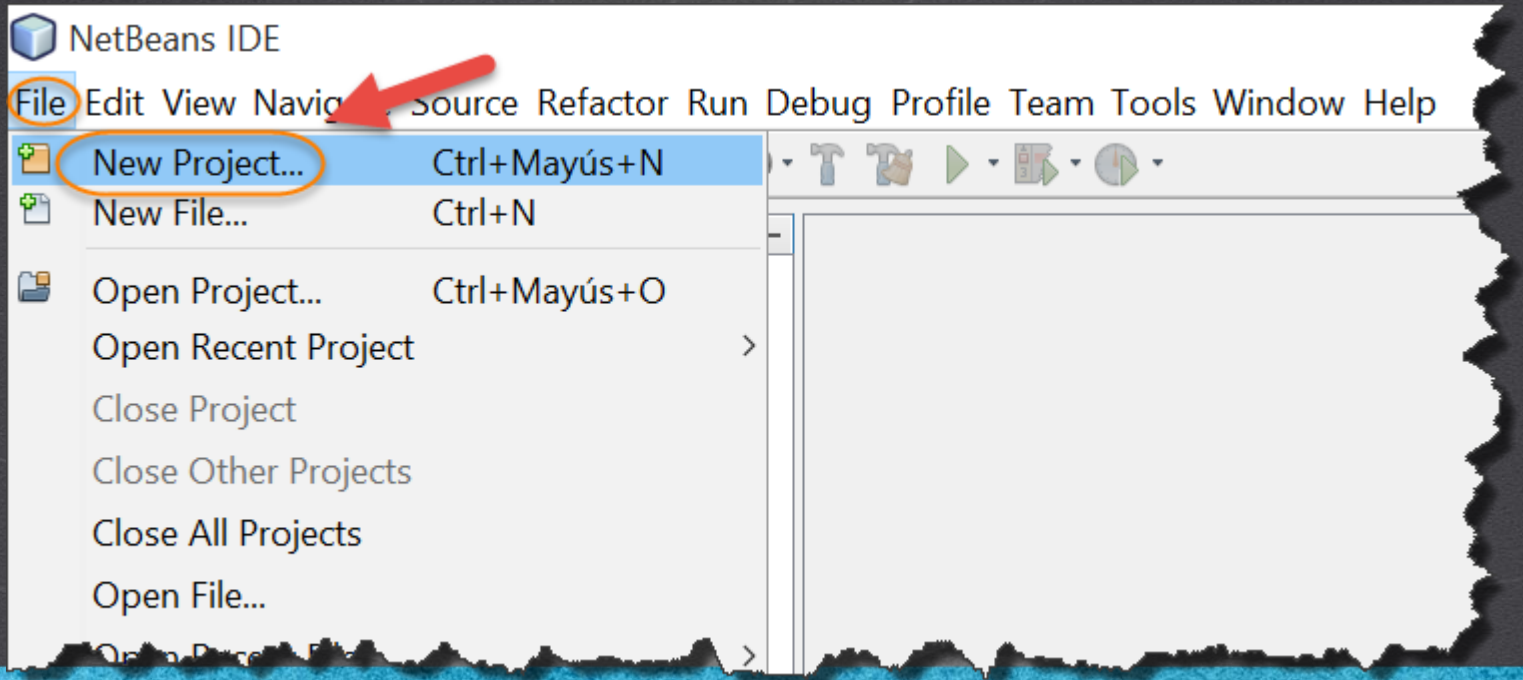
STOP GLASSFISH IF IT IS ACTIVE

Stop the Glassfish server if it was started:



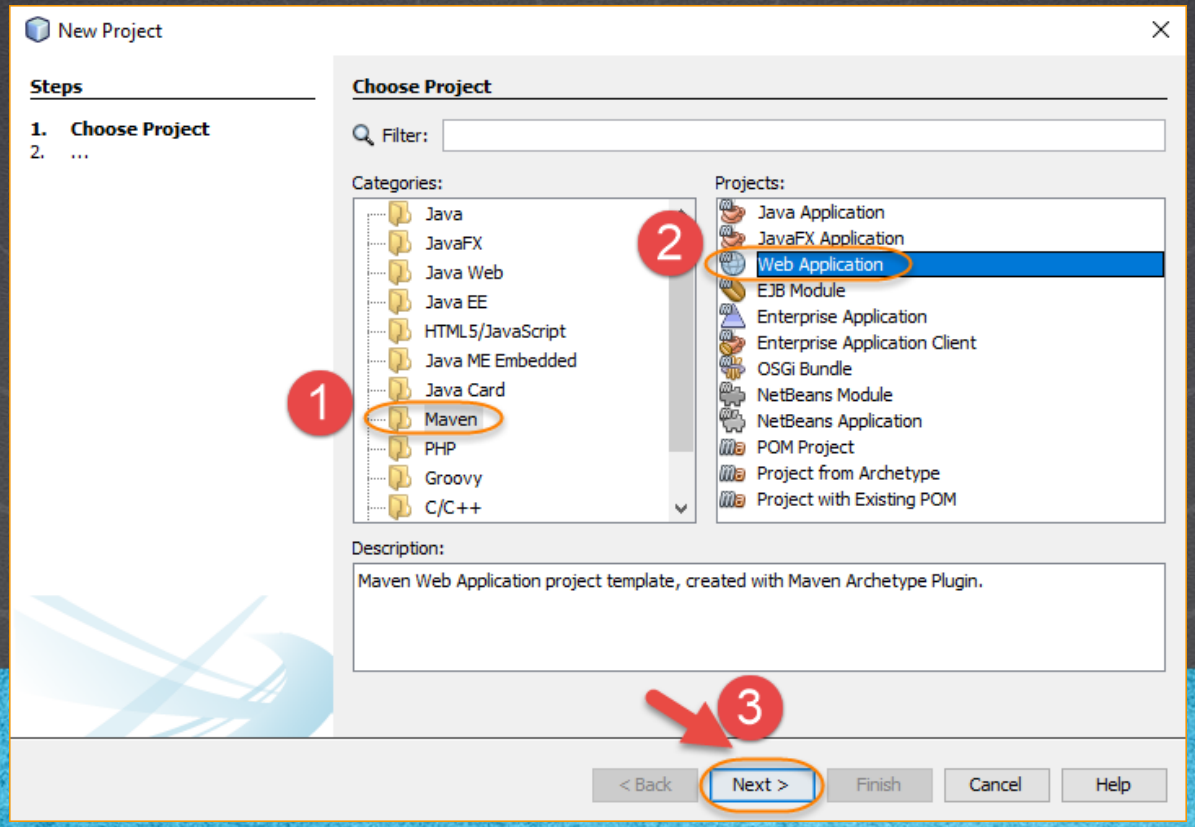
1. CREATE A NEW PROJECT

We create the sms-jee-web project:



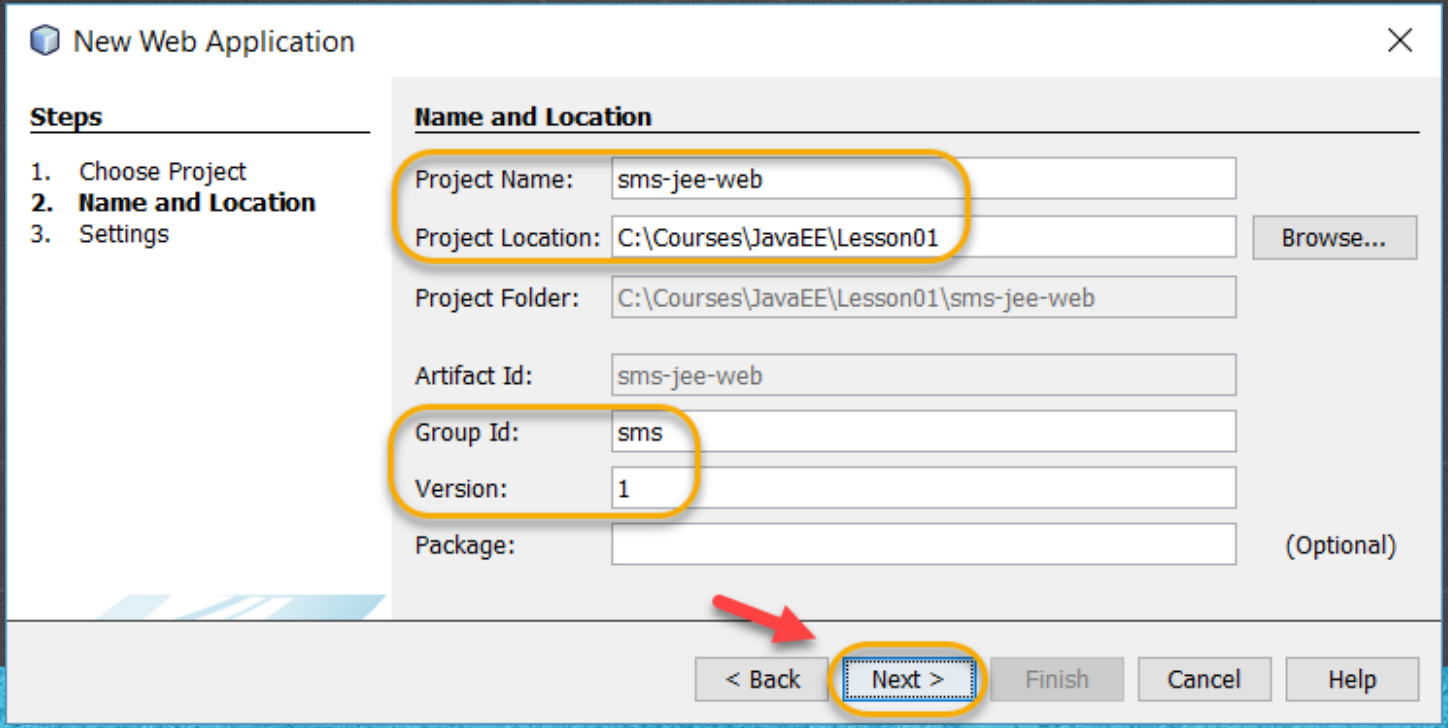
1. CREATE A NEW PROJECT

We create the sms-jee-web project:



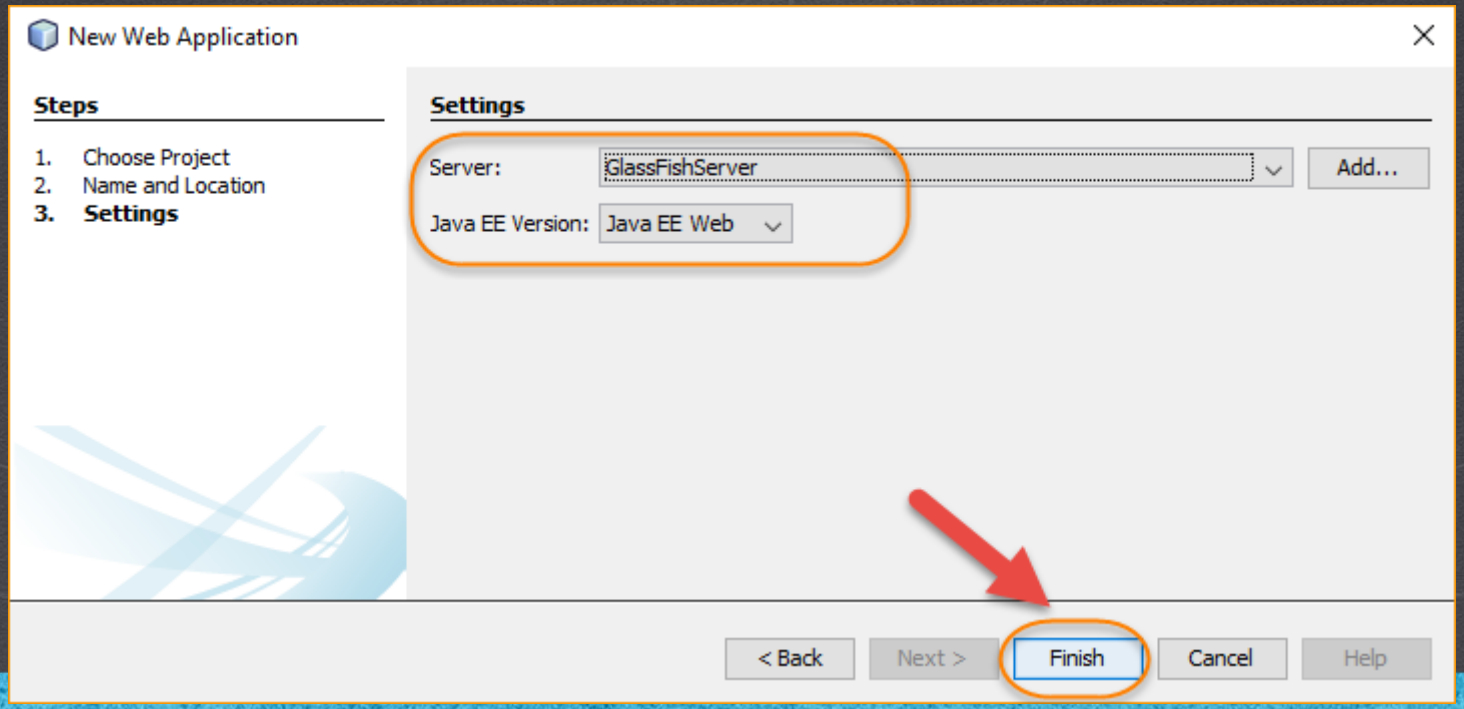
1. CREATE A NEW PROJECT

We create the sms-jee-web project:



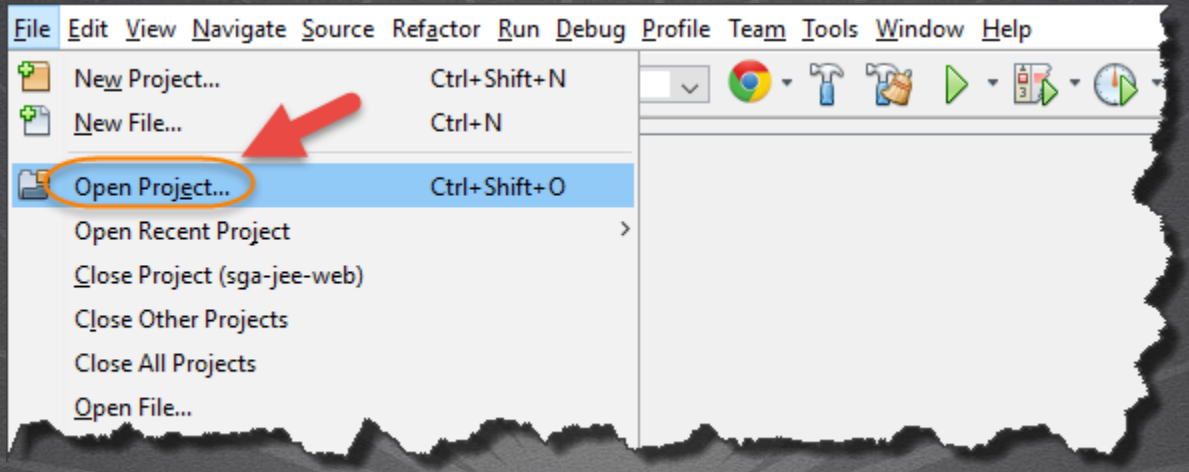
1. CREATE A NEW PROJECT

We create the sms-jee-web project:



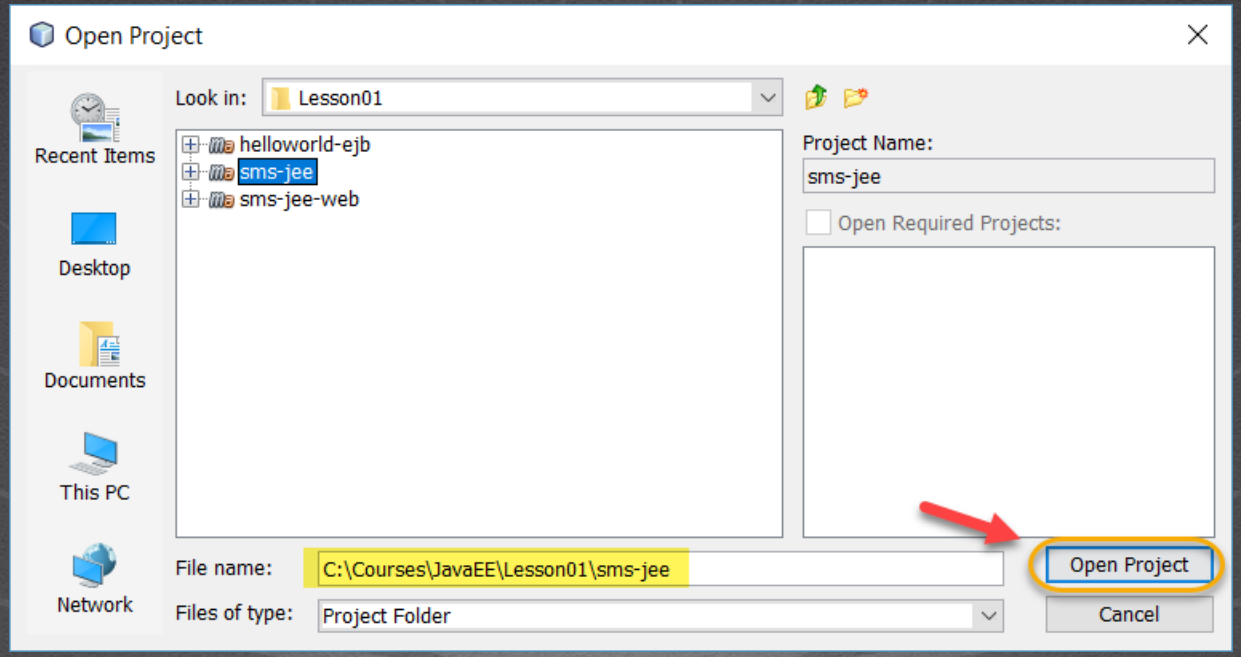
2. OPEN THE PREVIOUS PROJECT

We opened the sms-jee project created in the previous exercise. If it is already open, we omit this step, and we will use it:



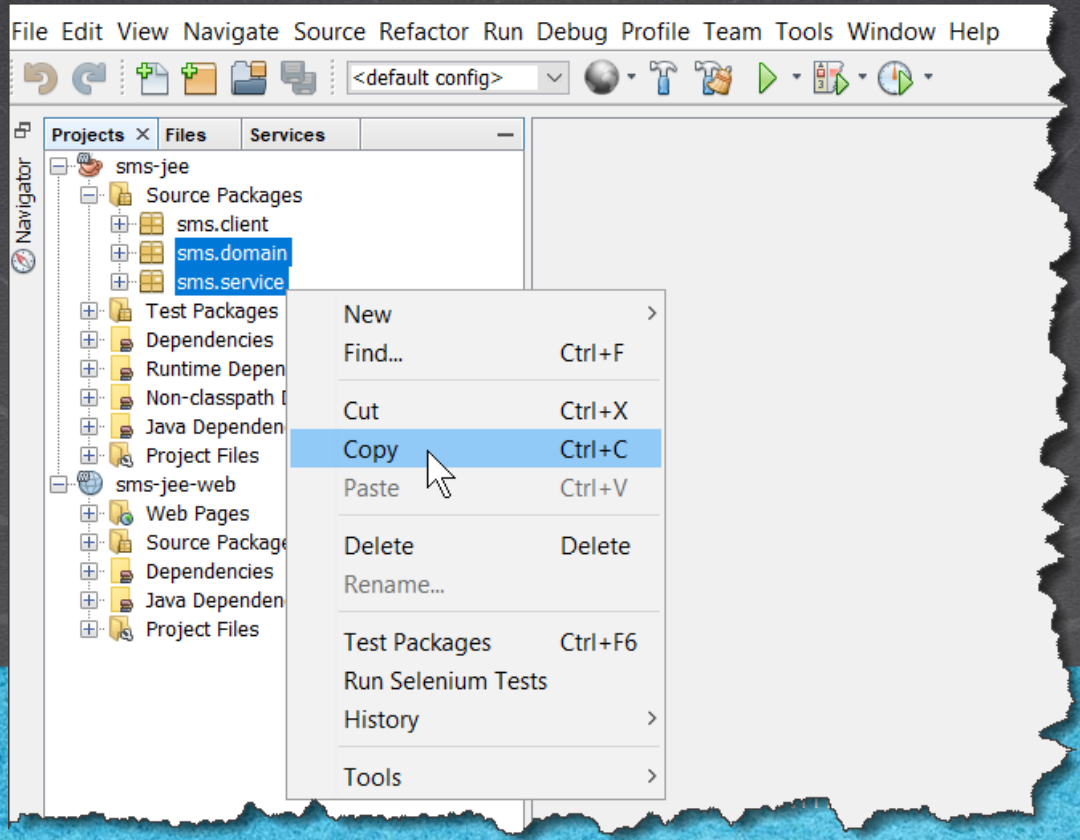
2. OPEN THE PREVIOUS PROJECT

We opened the sms-jee project created in the previous exercise:



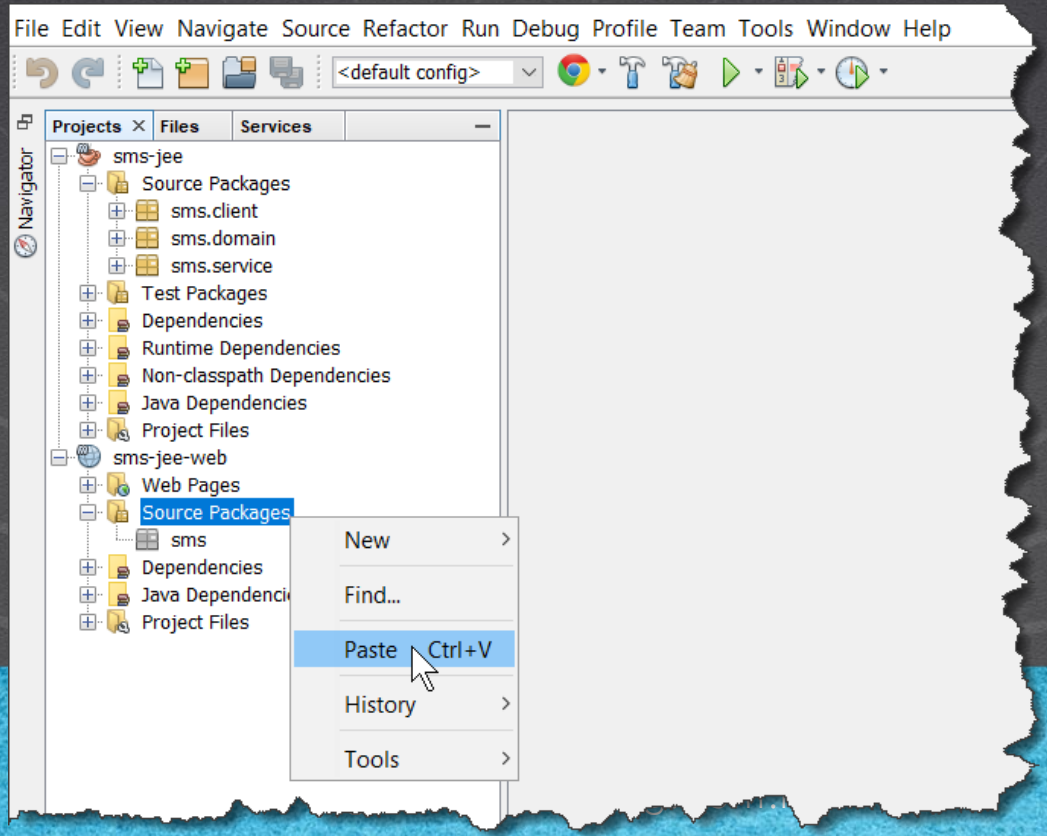
3. COPY THE FOLLOWING CLASSES

We copy the classes of the domain and service packages of the sms-jee project:



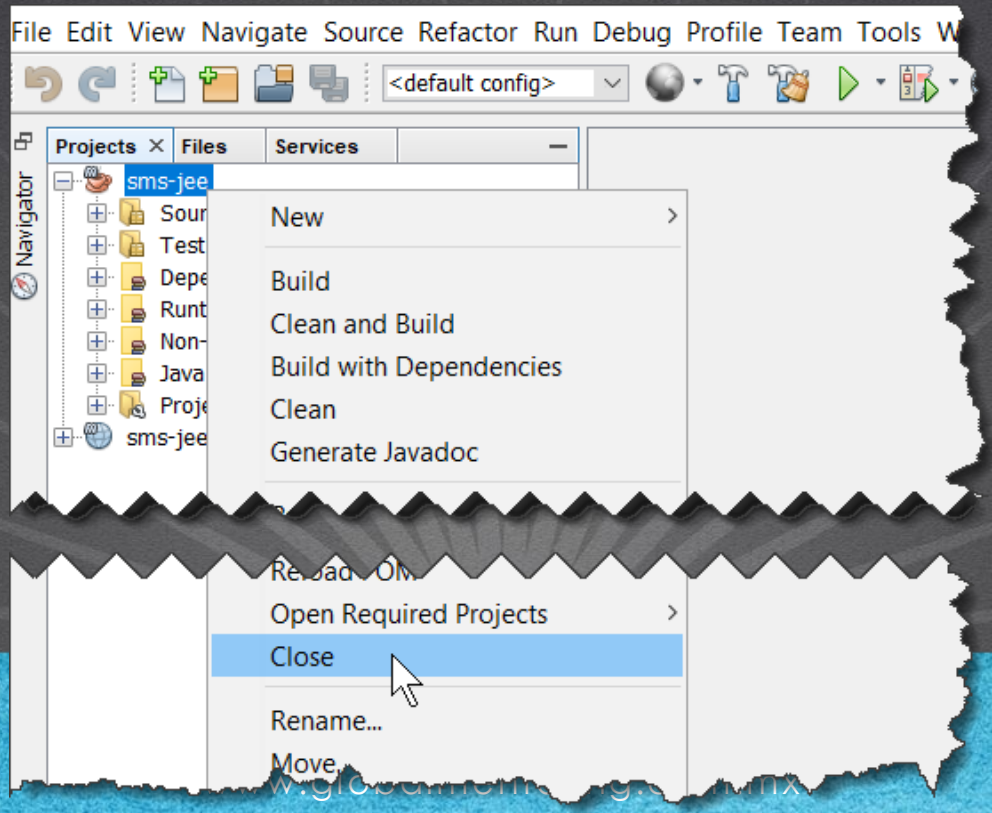
3. COPY THE FOLLOWING CLASSES

We paste the classes of the domain and service packages to the new project as shown:



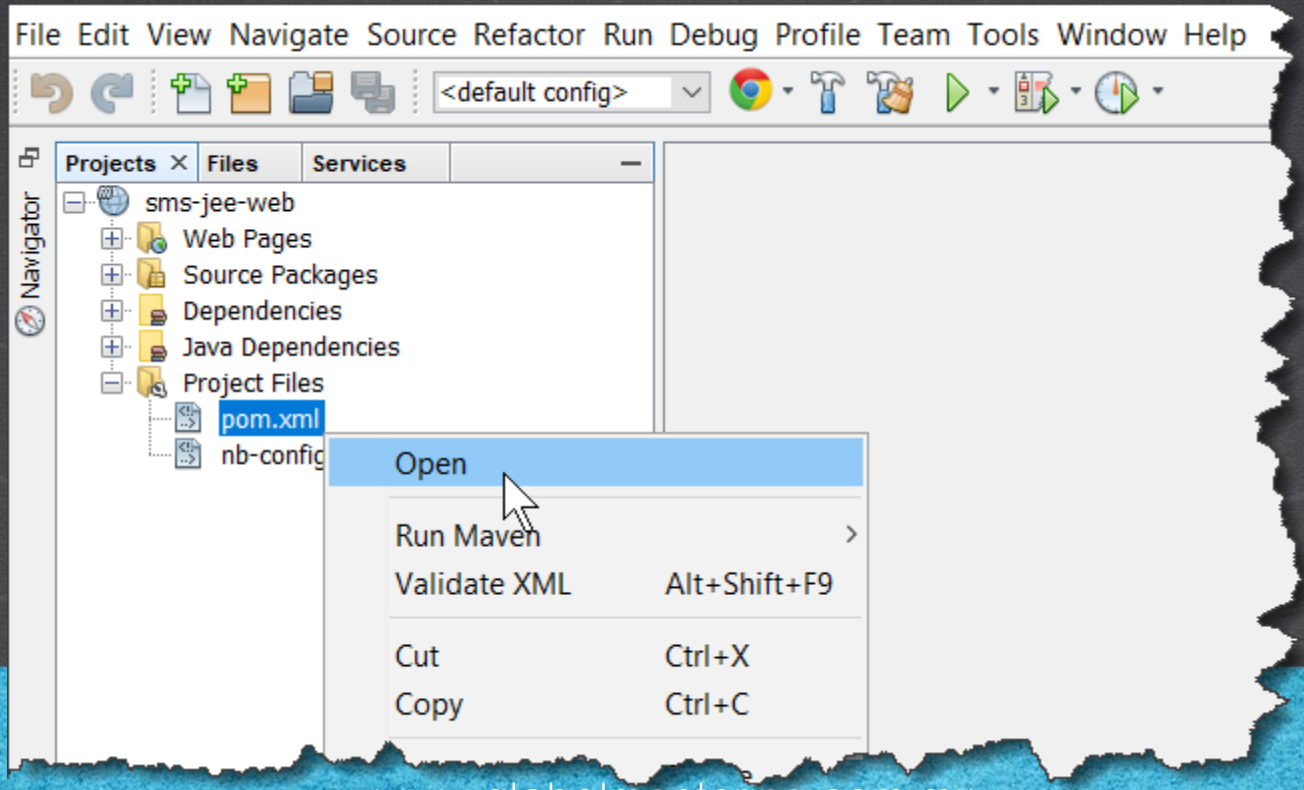
4. CLOSE THE PROJECT THAT WE NO LONGER USE

We closed the sms-jee project that we will no longer use:



5. MODIFY THE POM.XML FILE

Modify the pom.xml file:



5. MODIFY THE CODE

pom.xml:

Click to download

```
<?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>
    </dependencies>
```

5. MODIFY THE CODE

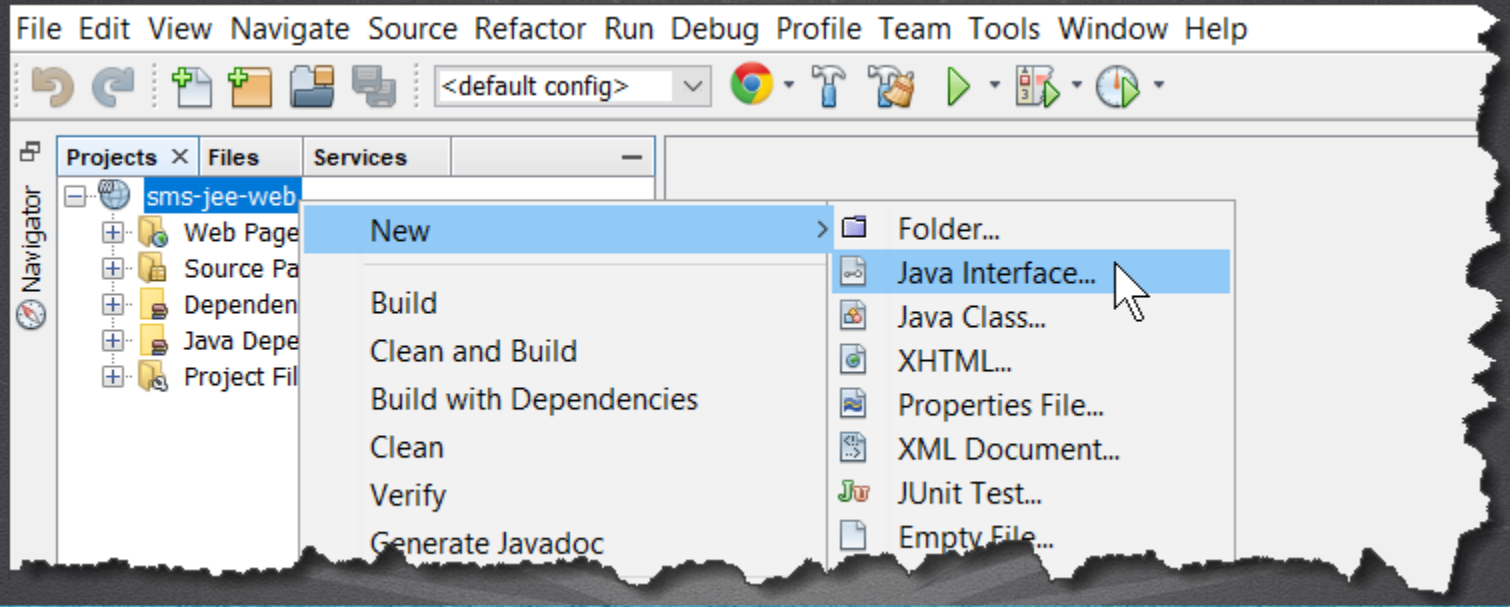
pom.xml:

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>
```

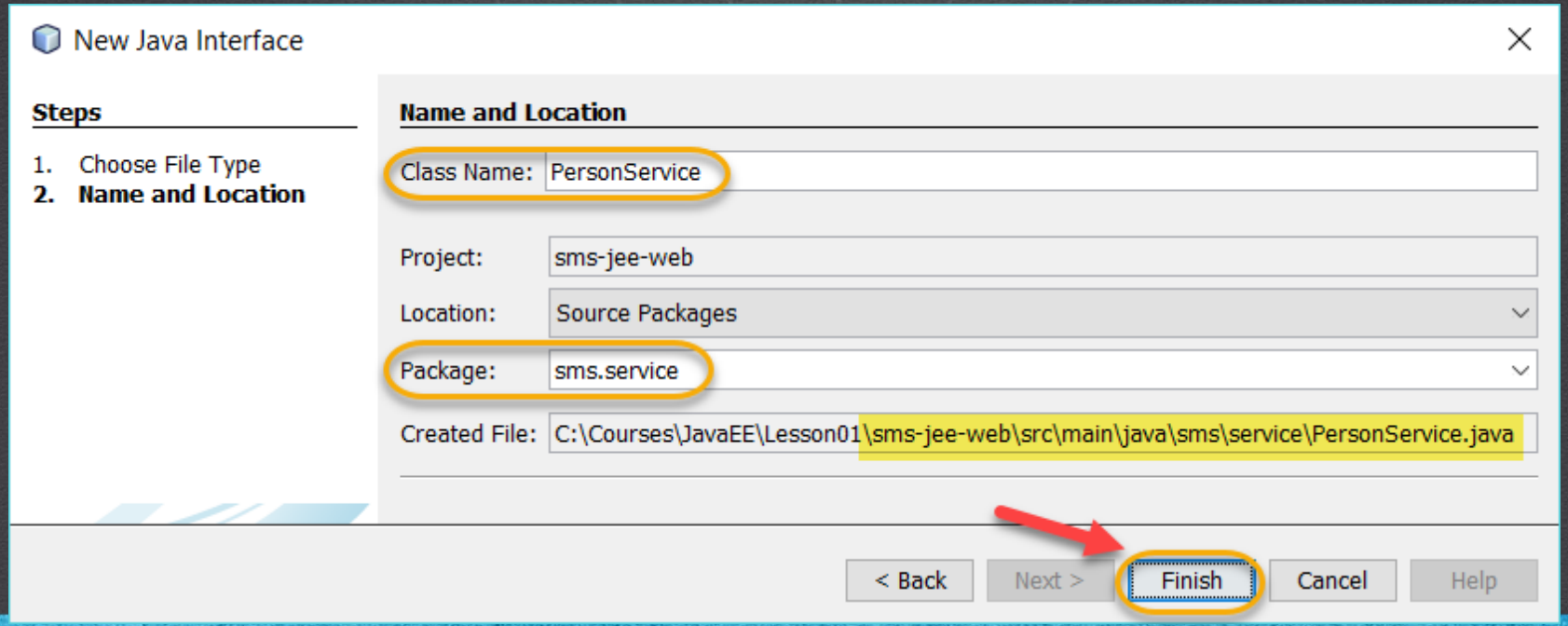
6. CREATE AN INTERFACE

We create the PersonService.java interface:



6. CREATE AN INTERFACE

We create the PersonService.java interface:



7. MODIFY THE FILE

PersonService.java:

Click to download

```
package sms.service;

import java.util.List;
import javax.ejb.Local;
import sms.domain.Person;

@Local
public interface PersonService {

    public List<Person> listPeople();

    public Person findPerson(Person person);

    public void addPerson(Person person);

    public void modifyPerson(Person person);

    public void deletePerson(Person person);
}
```

8. MODIFY THE JAVA CLASS

Modify the PersonServiceImpl.java class:

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config>

Navigator

Projects Files Services

- sms-jee-web
  - Web Pages
  - Source Packages
    - sms.domain
    - sms.service
      - PersonService.java
      - PersonServiceImpl.java
      - PersonServiceRemote.j
  - Test Packages
  - Dependencies
  - Java Dependencies
  - Project Files

Open	
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Compile File	F9
Run File	Shift+F6

8. MODIFY THE FILE

PersonServiceImpl.java:

Click to download

```
package sms.service;

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;
import sms.domain.Person;

@Stateless
public class PersonServiceImpl implements PersonServiceRemote, PersonService {

    @Override
    public List<Person> listPeople() {
        List<Person> people = new ArrayList<>();
        people.add(new Person(1, "John"));
        people.add(new Person(2, "Samantha"));
        return people;
    }

    @Override
    public Person findPerson(Person person) {
        return null;
    }
```

8. MODIFY THE FILE

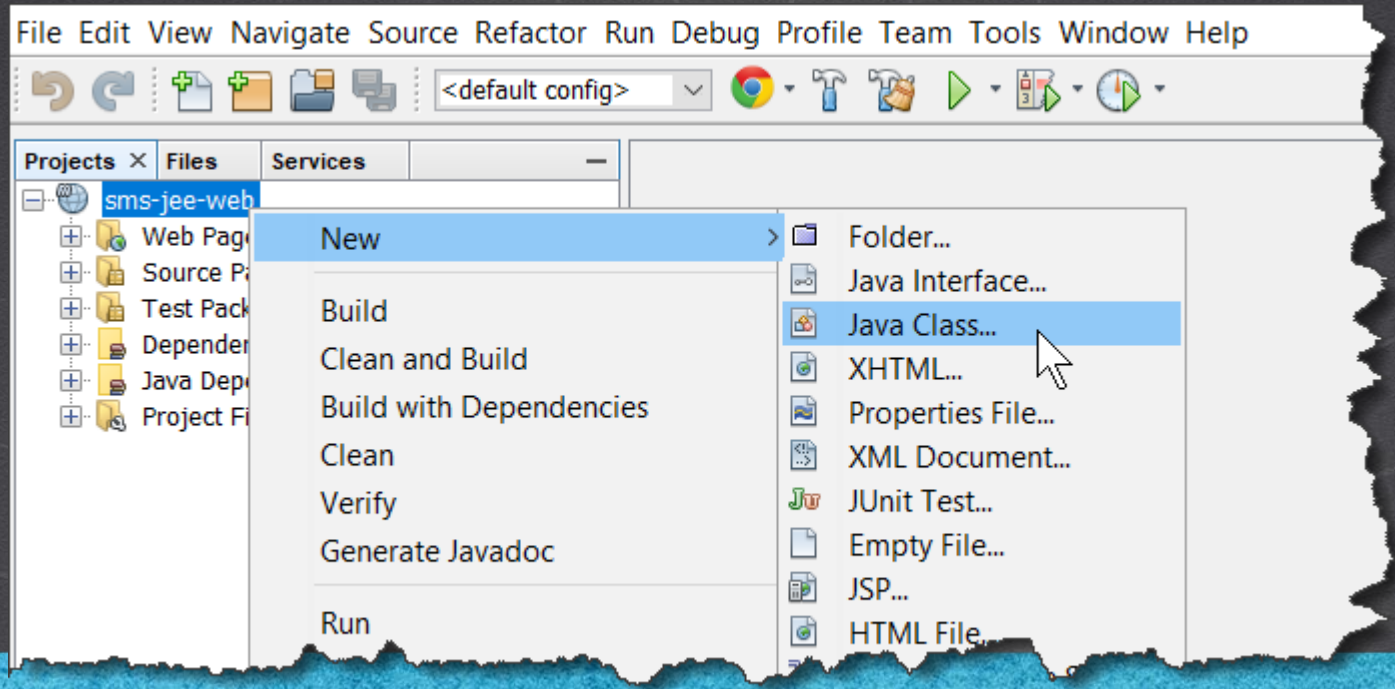
PersonServiceImpl.java:

Click to download

```
    @Override
    public void addPerson(Person person) {}

    @Override
    public void modifyPerson(Person person) {}

    @Override
    public void deletePerson(Person person) {}
}
```

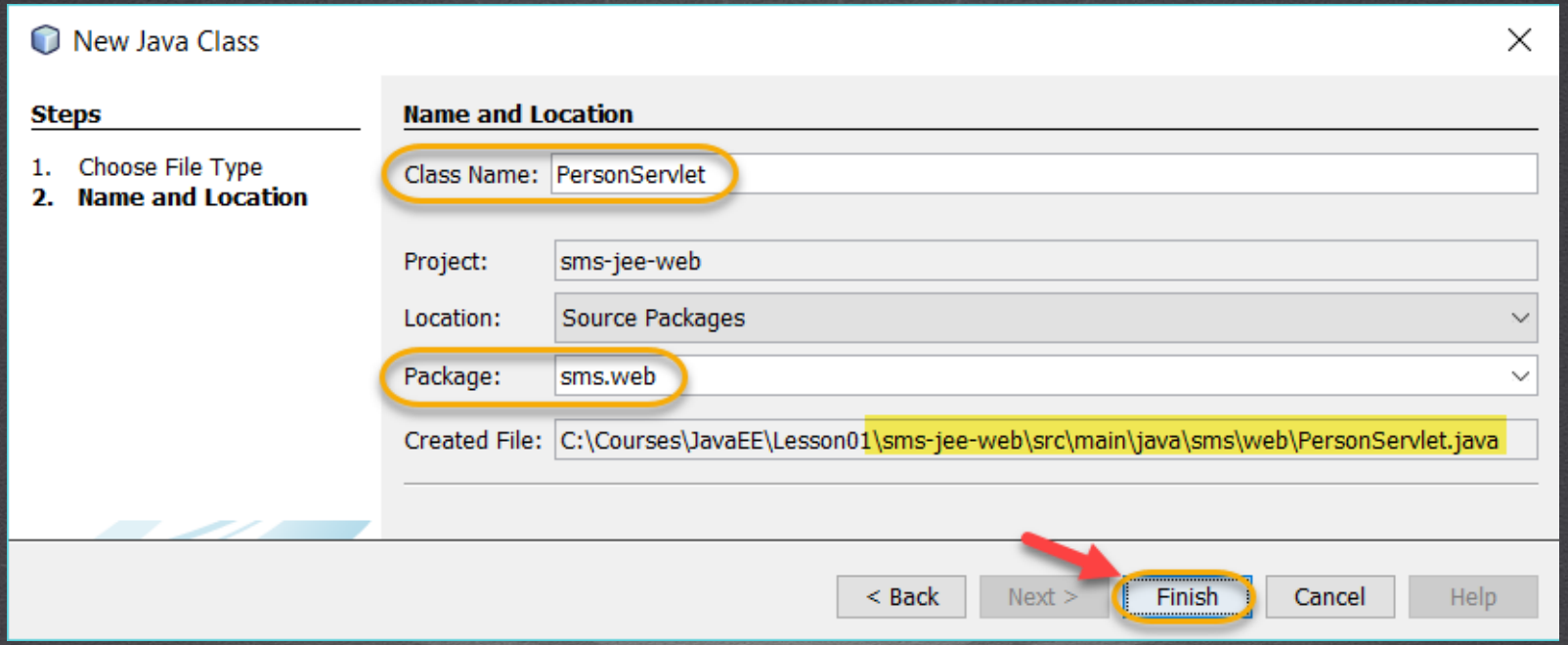
9. CREATE A JAVA CLASS

We create a Servlet called PersonServlet:



9. CREATE A JAVA CLASS

We create a Servlet called PersonServlet:



10. MODIFY THE CODE

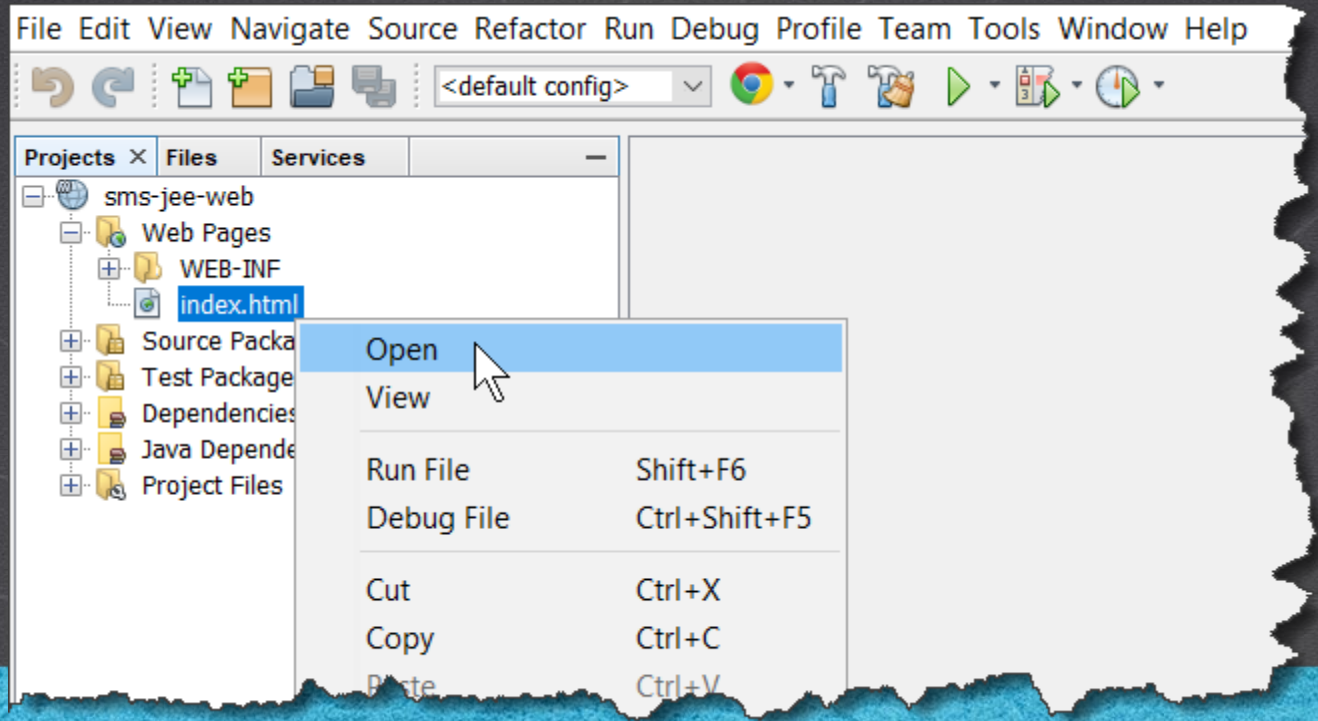
PersonServlet.java:

Click to download

```
package sms.web;

import java.io.IOException;
import java.util.List;
import javax.inject.Inject;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import sms.domain.Person;
import sms.service.PersonService;

@WebServlet("/people")
public class PersonServlet extends HttpServlet {

    @Inject
    PersonService personServiceLocalEjb;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        List<Person> people = personServiceLocalEjb.listPeople();
        System.out.println("people:" + people);
        request.setAttribute("people", people);
        request.getRequestDispatcher("/listPeople.jsp").forward(request, response);
    }
}
```

11. MODIFY THE INDEX.HTML FILE

Modify the index.html file. If it does not exist, we create it:



11. MODIFY THE FILE

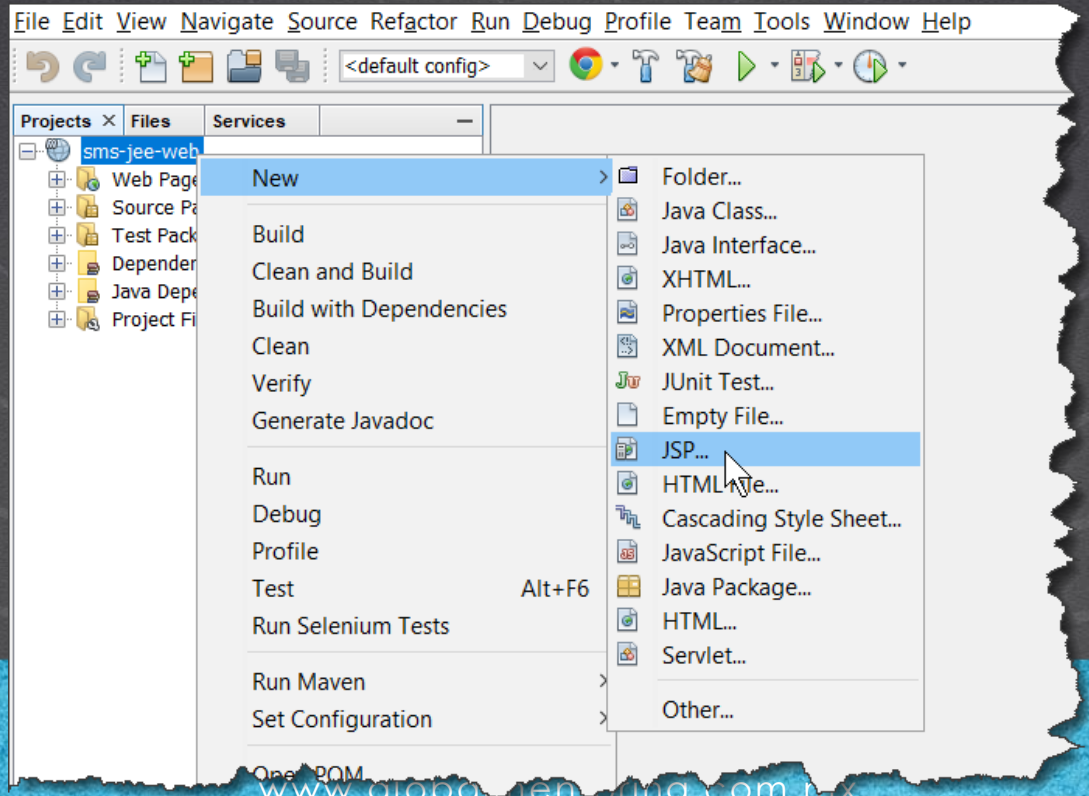
index.html:

Click to download

```
<!DOCTYPE html>
<html>
    <head>
        <title>Index</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <a href="people">Go to the list of people</a>
    </body>
</html>
```

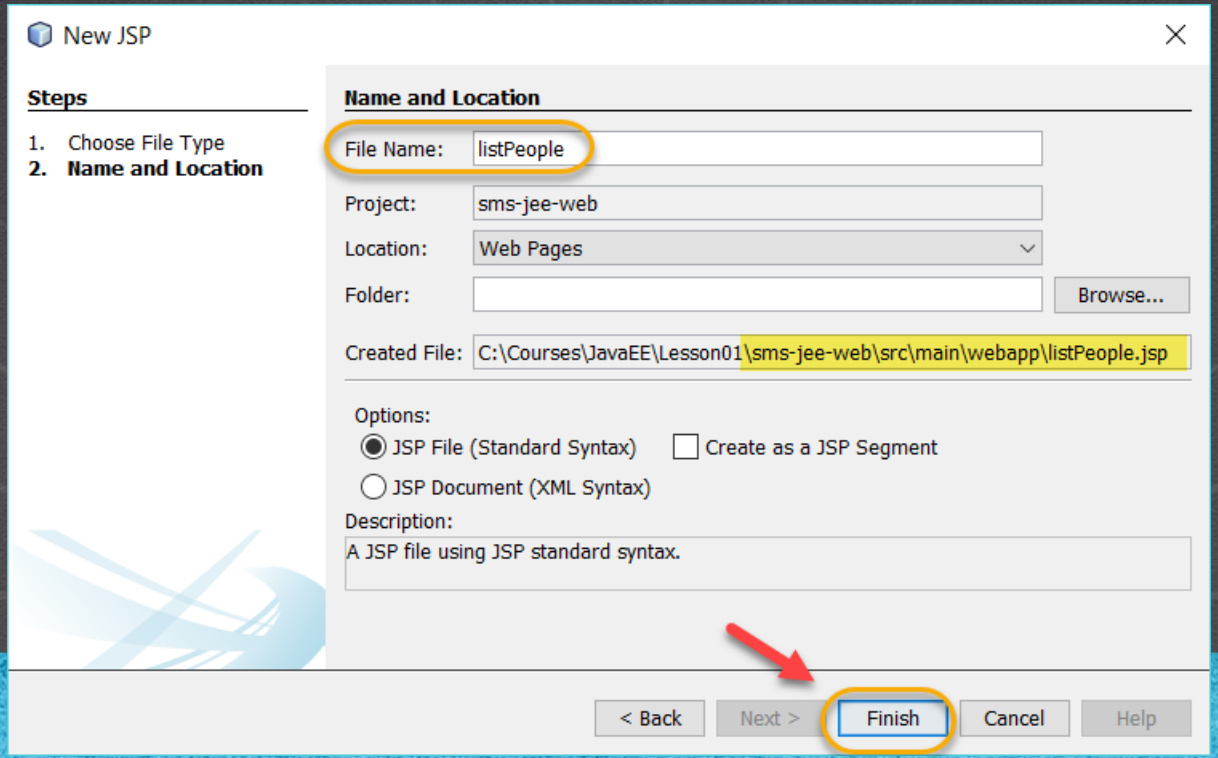
12. CREATE A JSP

We create a file called listPeople.jsp:



12. CREATE A JSP

We create a file called listPeople.jsp:



13. MODIFY THE CODE

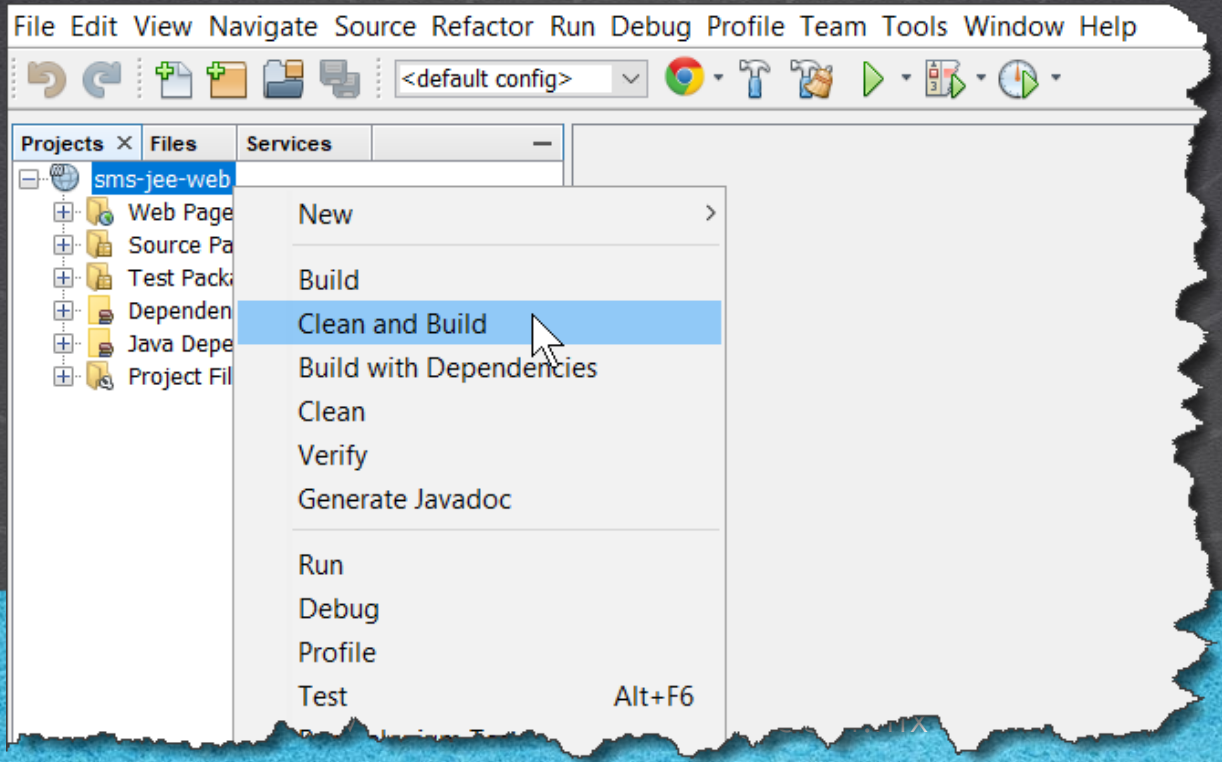
listPeople.jsp:

Click to download

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>
<!DOCTYPE html>
<html>
    <head>
        <title>List of People</title>
    </head>
    <body>
        <h1>List of People</h1>
        <ul>
            <c:forEach items="${people}" var="person">
                <li>${person.idPerson} ${person.name}</li>
            </c:forEach>
        </ul>
    </body>
</html>
```

14. EXECUTE CLEAN & BUILD

We do a Clean & Build application to have all the most recent files (Glassfish must be stopped):



14. EXECUTE CLEAN & BUILD

We observe the result, it must be similar to the one shown:

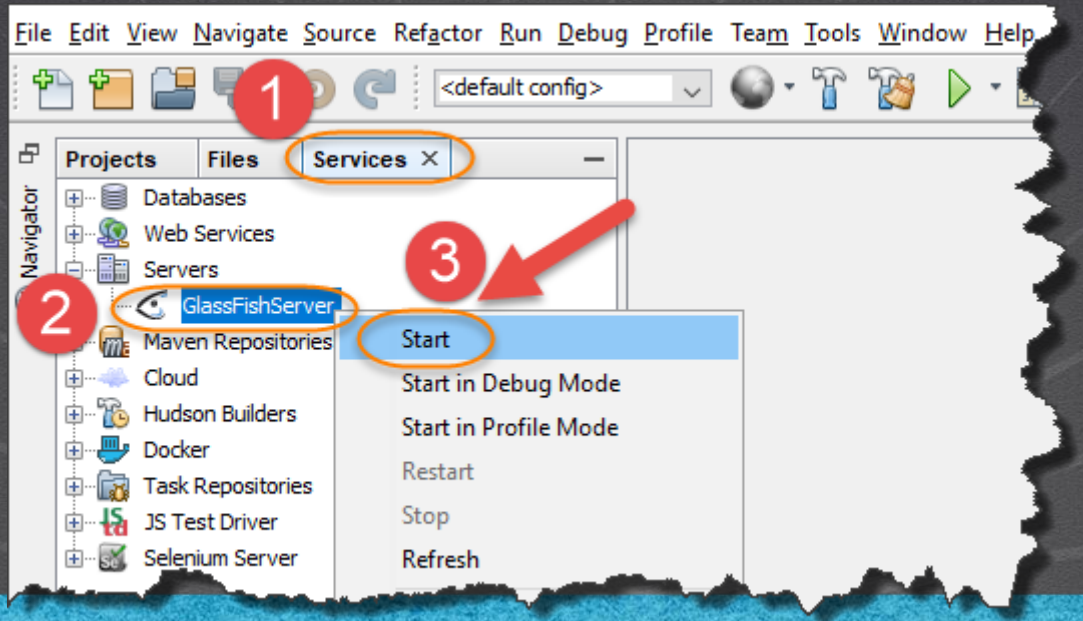
Output

Java DB Database Process | GlassFish Server | Build (sms-jee-web)

```
Packaging webapp
Assembling webapp [sms-jee-web] in [C:\Courses\JavaEE\Lesson01\sms-jee-web\target\sms-jee-web-1]
Processing war project
Copying webapp resources [C:\Courses\JavaEE\Lesson01\sms-jee-web\src\main\webapp]
Webapp assembled in [90 msecs]
Building war: C:\Courses\JavaEE\Lesson01\sms-jee-web\target\sms-jee-web-1.war

--- maven-install-plugin:2.3.1:install (default-install) @ sms-jee-web ---
Installing C:\Courses\JavaEE\Lesson01\sms-jee-web\target\sms-jee-web-1.war to C:\Users\user\.m2\repository\sms\sms-jee-web\1\sms-jee-web-1.war
Installing C:\Courses\JavaEE\Lesson01\sms-jee-web\pom.xml to C:\Users\user\.m2\repository\sms\sms-jee-web\1\sms-jee-web-1.pom
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 14.523s
Finished at: Tue Oct 16 17:14:56
Final Memory: 21M/287M
------------------------------------------------------------------------
```

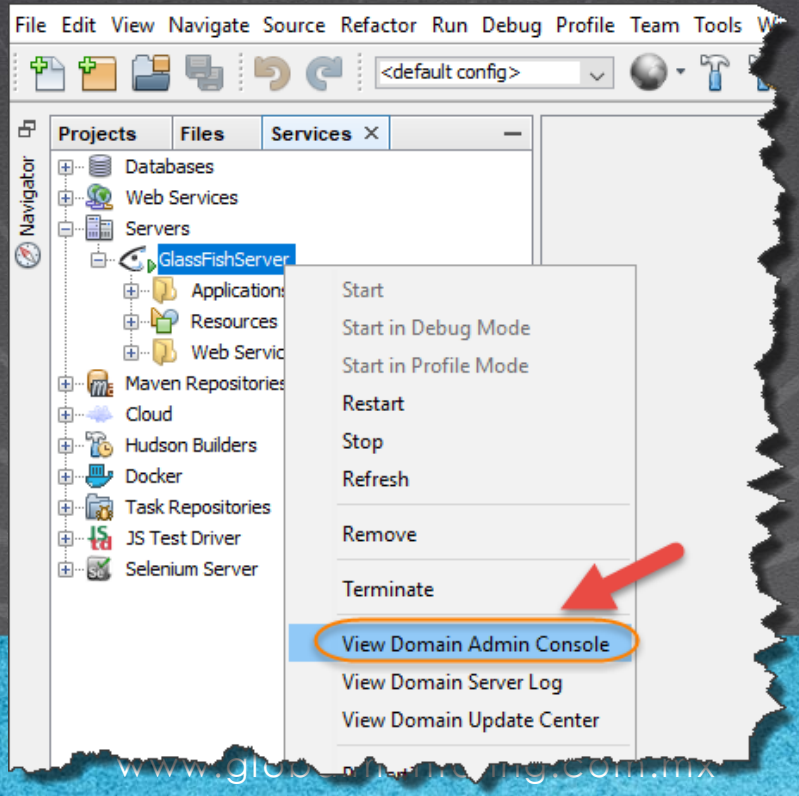
15. START UP GLASSFISH

We raise Glassfish to eliminate the deployed applications:



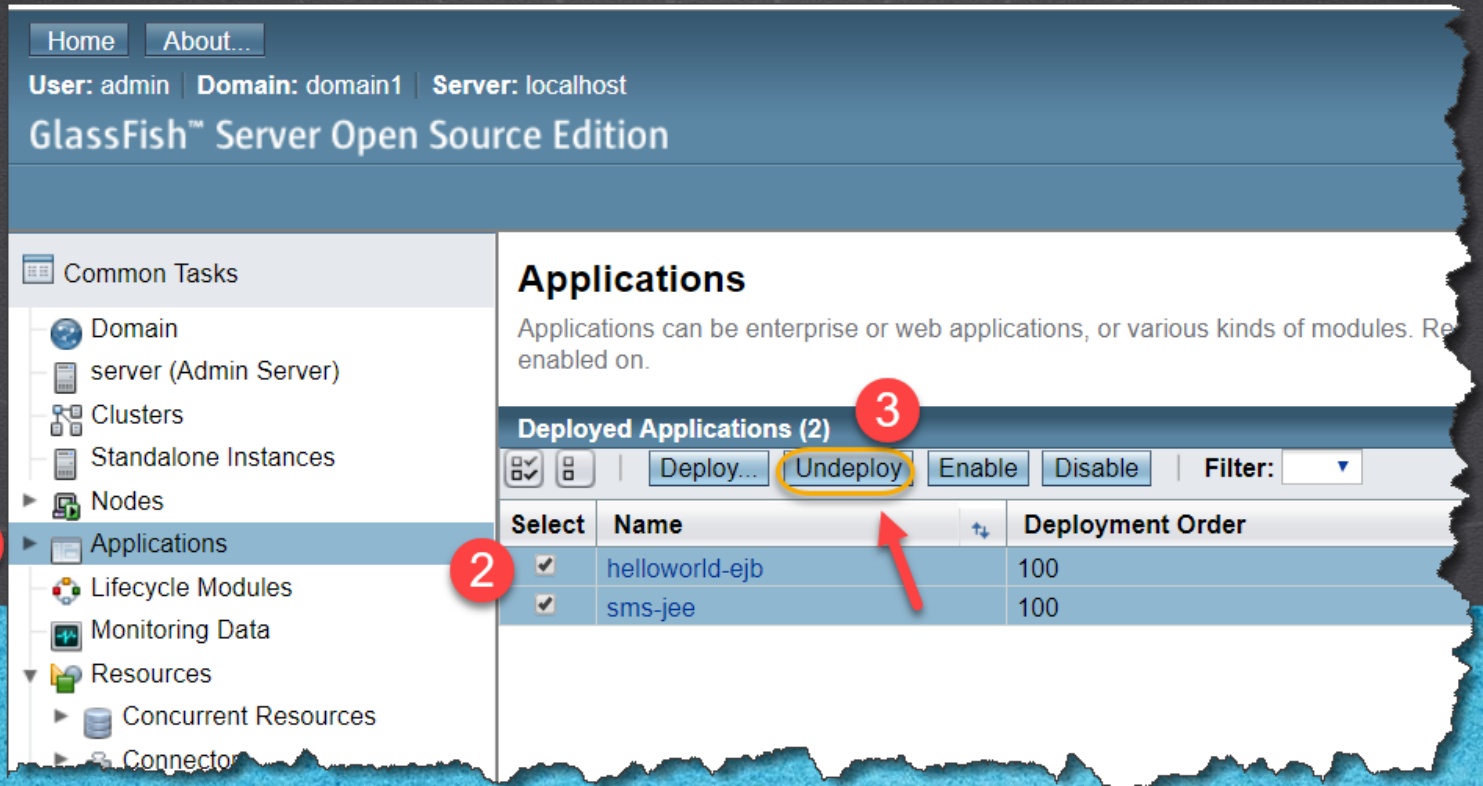
15. START UP GLASSFISH

After several seconds, we wait for you to lift, and once up, we enter the Glassfish administration console:



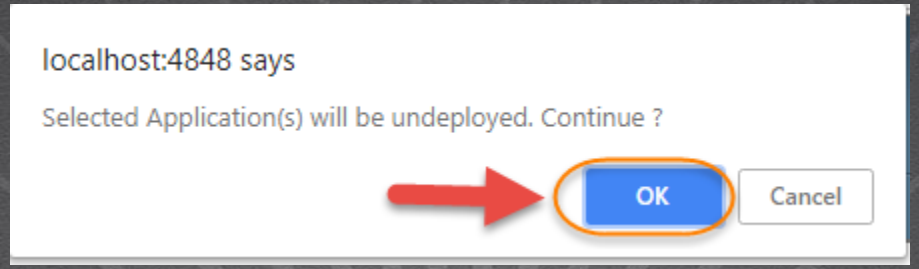
16. DELETE ALL THE APPLICATIONS

We go to Applications, select all the deployed applications and make Undeploy as shown:



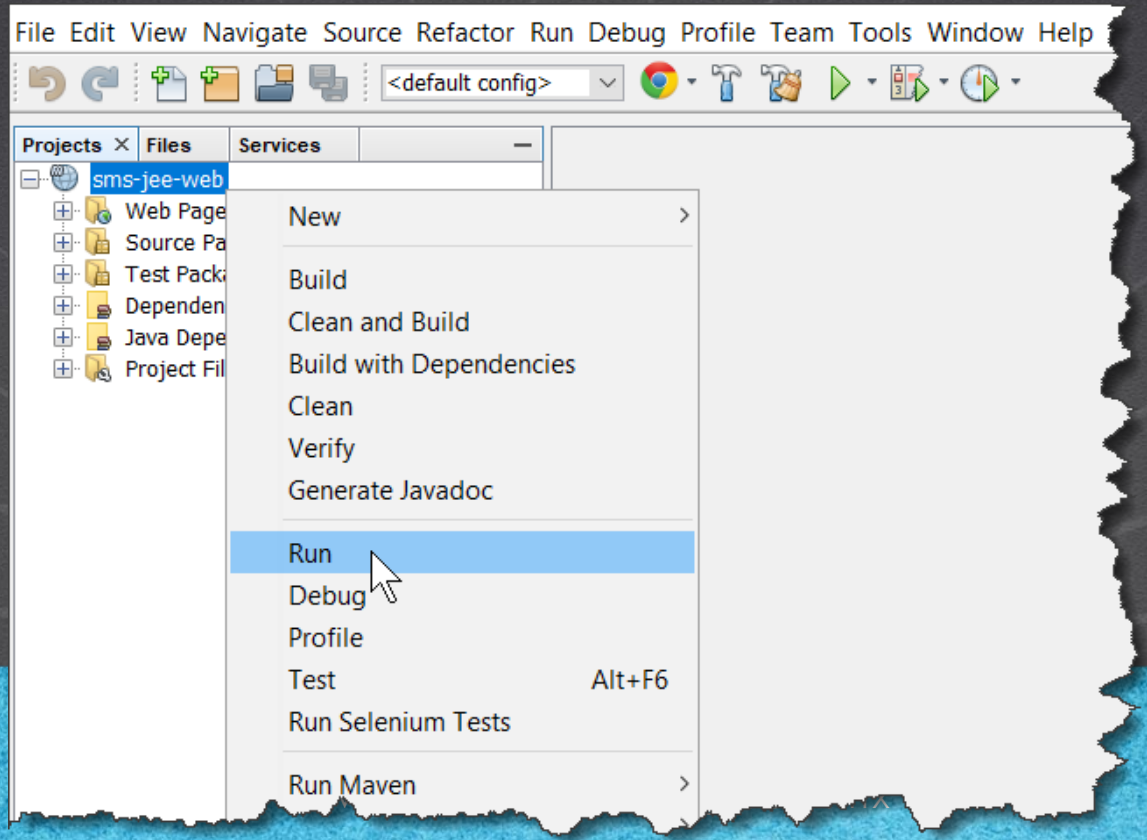
16. DELETE THE APPLICATIONS

We accept to undeploy the applications (they will be removed only from Glassfish):



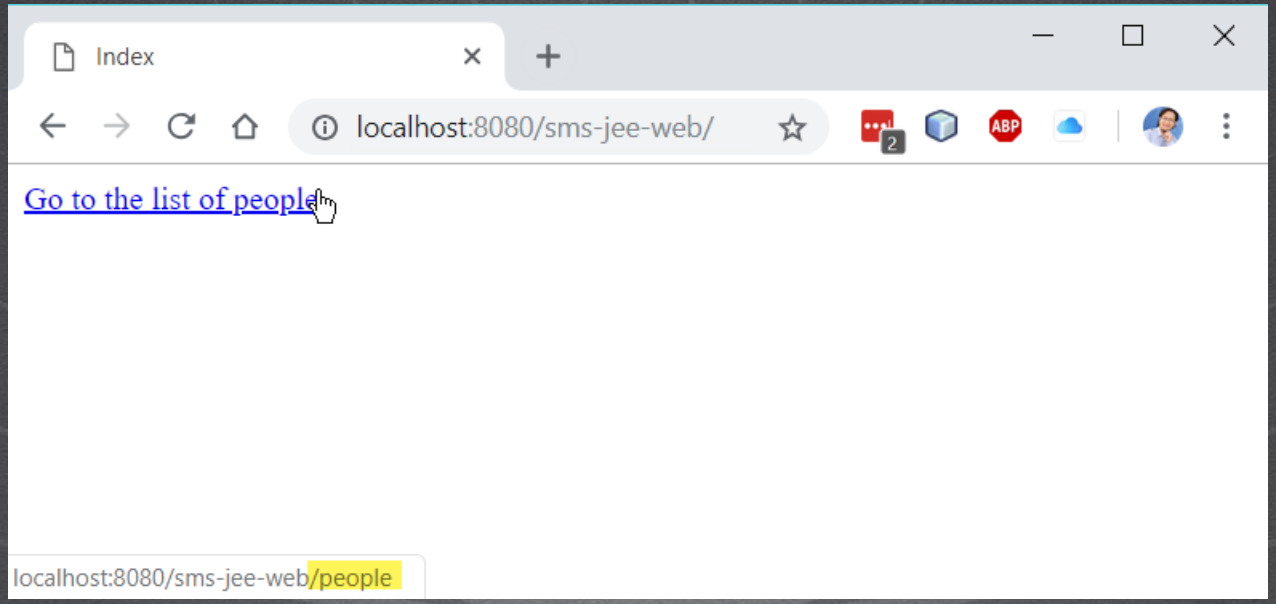
17. EXECUTE THE APPLICATION

Execute the application:



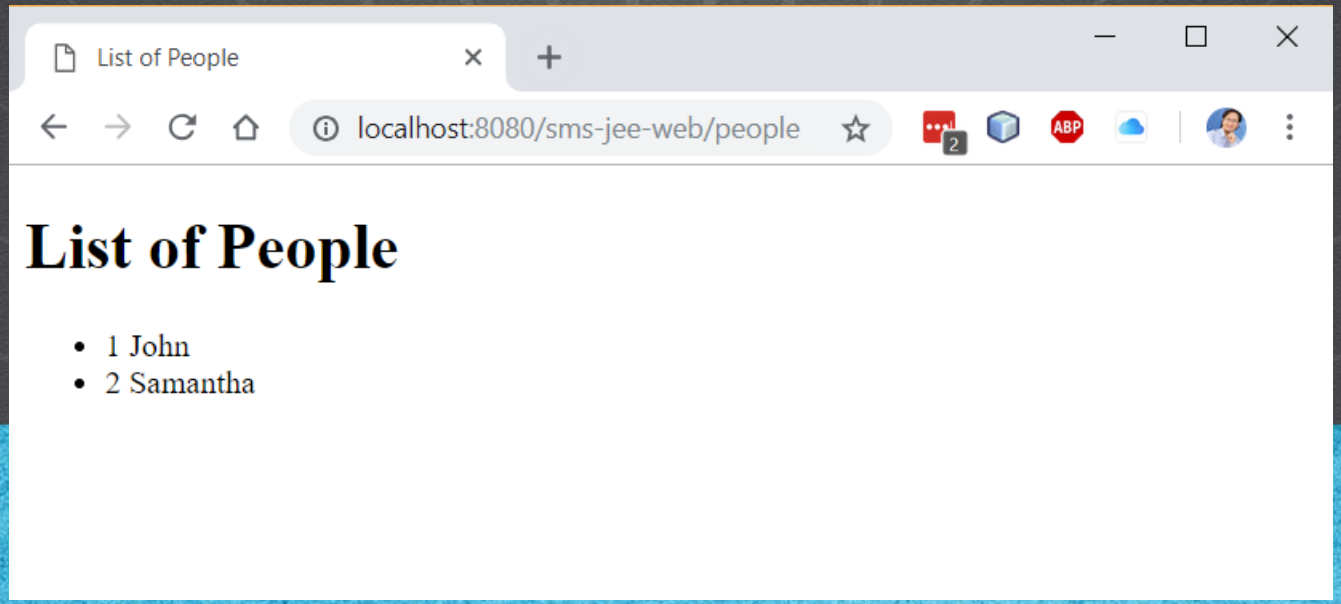
17. EXECUTE THE APPLICATION

The result of index.html is shown, we click to take it to the Servlet and later to the peopleList.jsp:



17. EXECUTE THE APPLICATION

We observe the result of executing the list of people. And we can verify that the Local EJB that we defined in our application has been correctly used. The EJB is injected into the Servlet using the @Inject annotation that is part of the Java EE CDI API. From the Servlet we call the method listPeople of the EJB. This result was shared in the scope of request from the Servlet and when forwarding to the peopleList.jsp, we went through this list of Person objects using JSTL and EL.



OBSERVATIONS IN CASE OF PROBLEMS

If for some reason the execution of the project does not work, we recommend you to carry out the following actions :

1) The Glassfish server must be running in order to access the local EJB and the Web application.
2) In this case any change in the EJB or the interface is displayed automatically, however if it fails, stop Glassfish, make the respective changes, return to Clean & Build the project and "run" the project so that the changes in the Java server.
3) If none of this works, try loading the resolved project, which is 100% functional and contains all the classes and changes described in the exercise.

EXERCISE CONCLUSION

- With this exercise we have implemented the call to a Local EJB and we tested the EJB by creating a small Web application with Servlets and JSPs.

- We could observe how simple it is to inject the dependency of an EJB of Stateless type simply using the name of the local interface, and thus avoid unnecessary remote calls, as long as the EJB is deployed in the same Web server, otherwise we will have to use the remote interface instead of the local interface.

- We applied the concept of CDI (Context and Dependency Injection) to inject the dependency of the EJB using the annotation of @Inject, it was that simple to use the EJB in the Servlet class.

ONLINE COURSE

JAVA EE
JAKARTA EE

By: Eng. Ubaldo Acosta