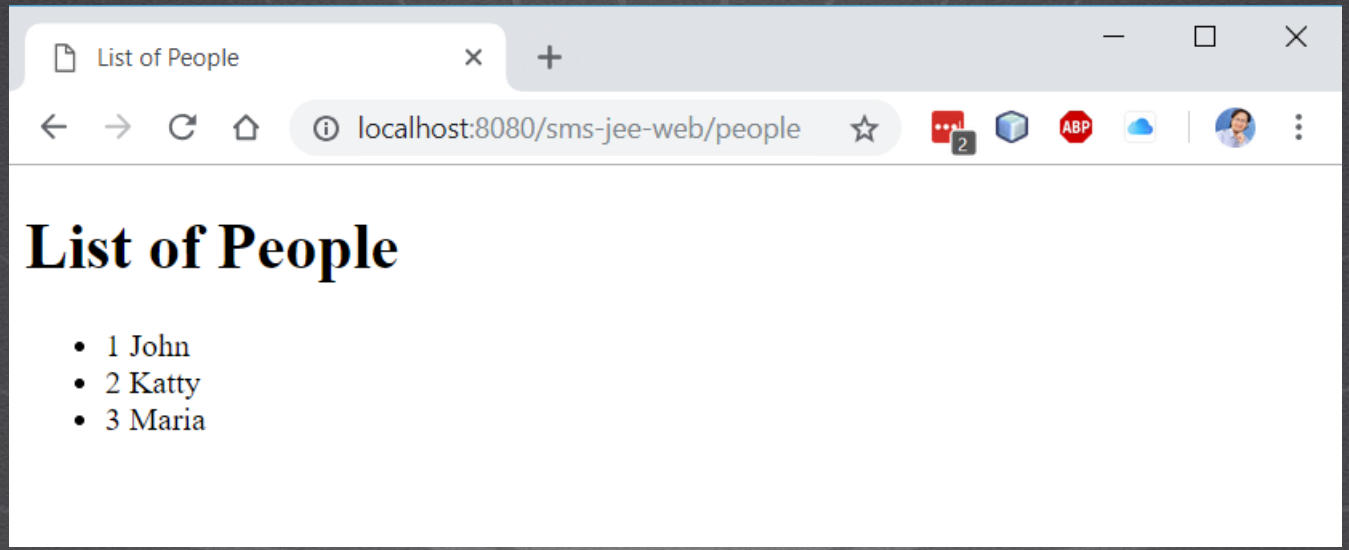
EXERCISE

SMS SYSTEM WITH JPA

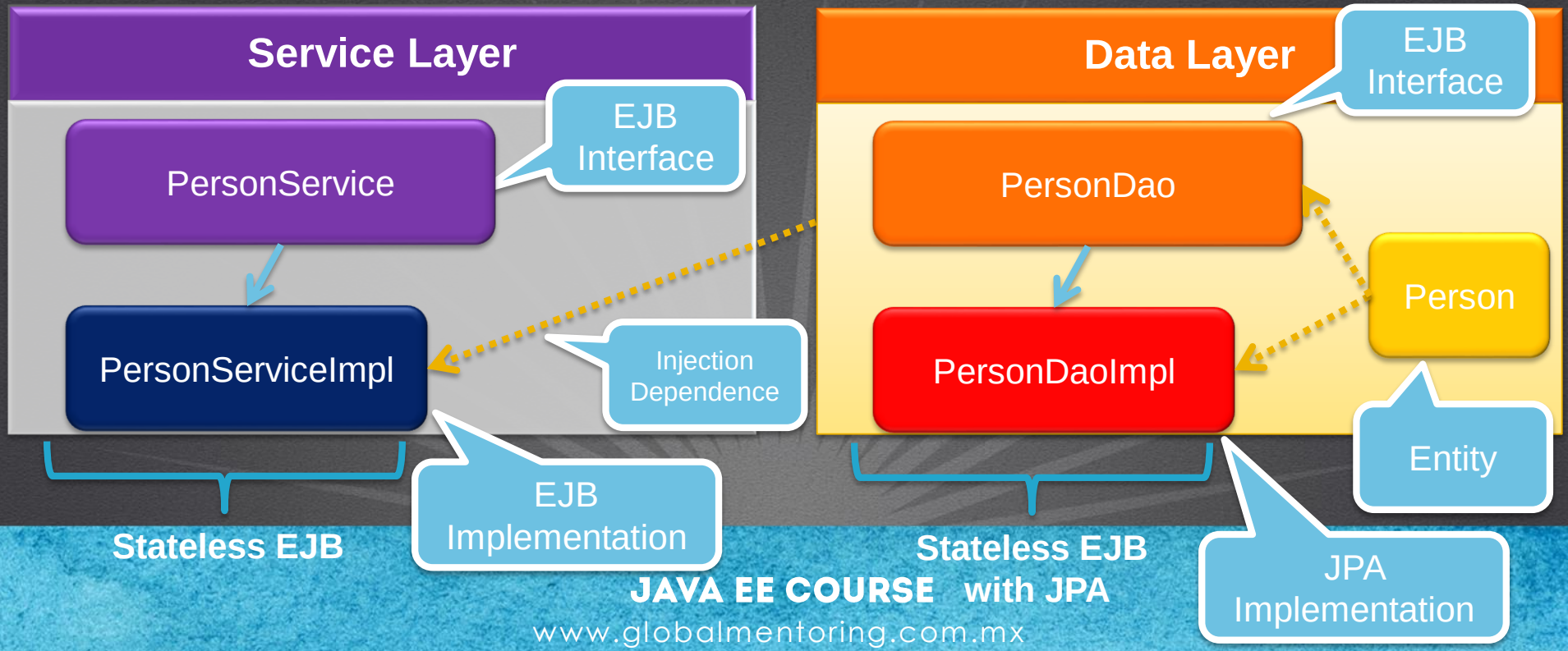
EXERCISE OBJECTIVE

•The objective of the exercise is to add persistence with JPA to our SMS (Student Management System) project. At the end we must observe the following result:



JAVA EE ARCHITECTURE

•We will convert our Person class into an Entity class, in turn we will add the data layer of our SMS System (Student Management System) in order to integrate the persistence with JPA.



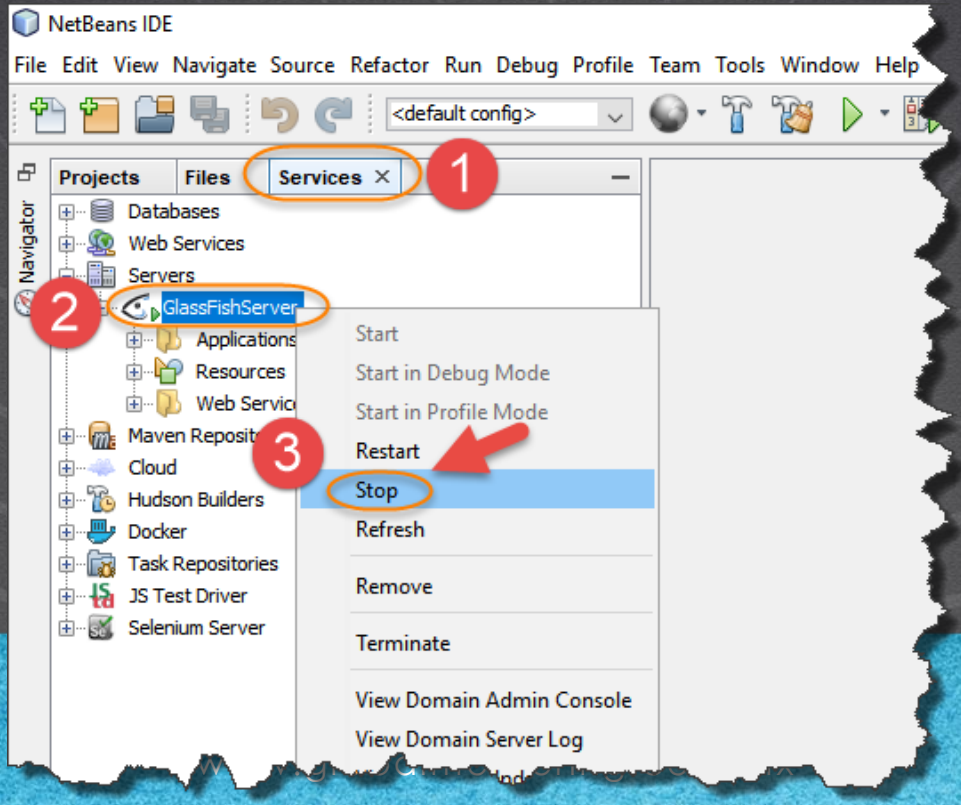
JTA CONFIGURATION

First, we execute the steps in the following guide to configure the pool of connections in Glassfish using JTA (omit this step if it is already done):

http://icursos.net/en/Installations/CJ-B-Exercise-04-JTAGlassfish.pdf

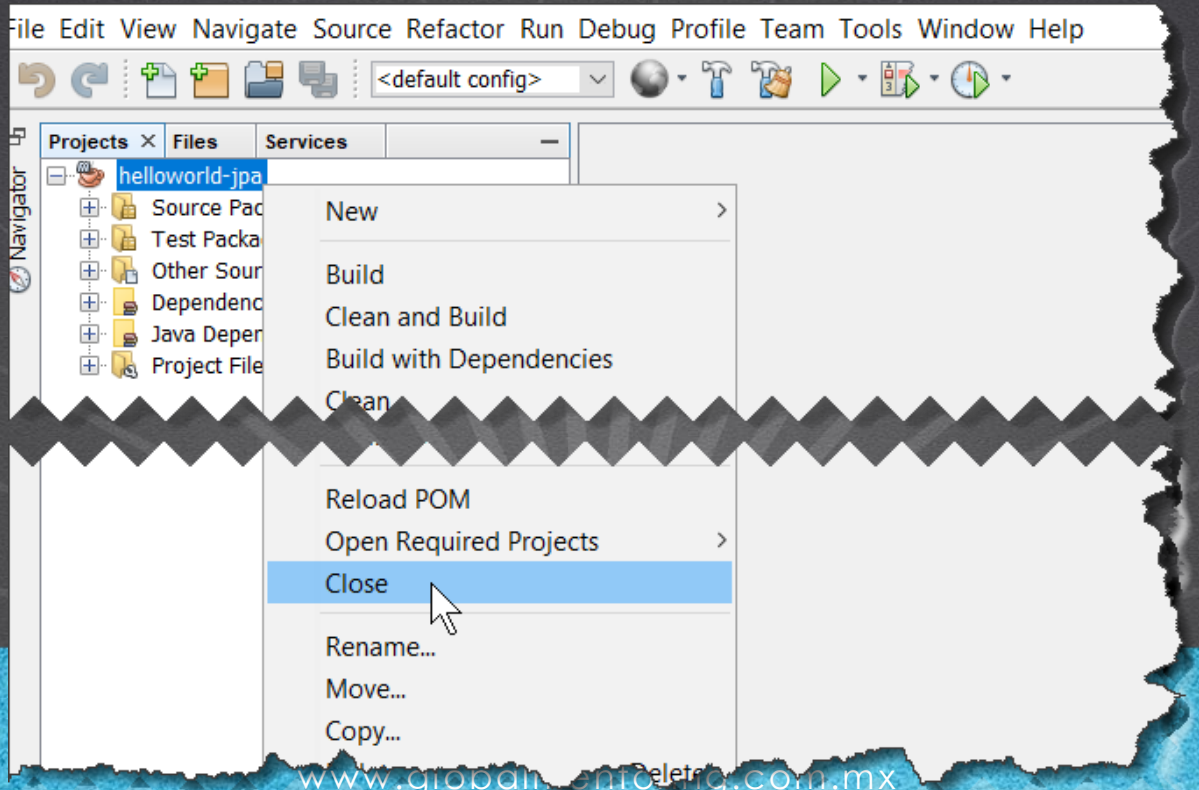
STOP GLASSFISH IF IT IS ACTIVE

Stop the Glassfish server if it was started:



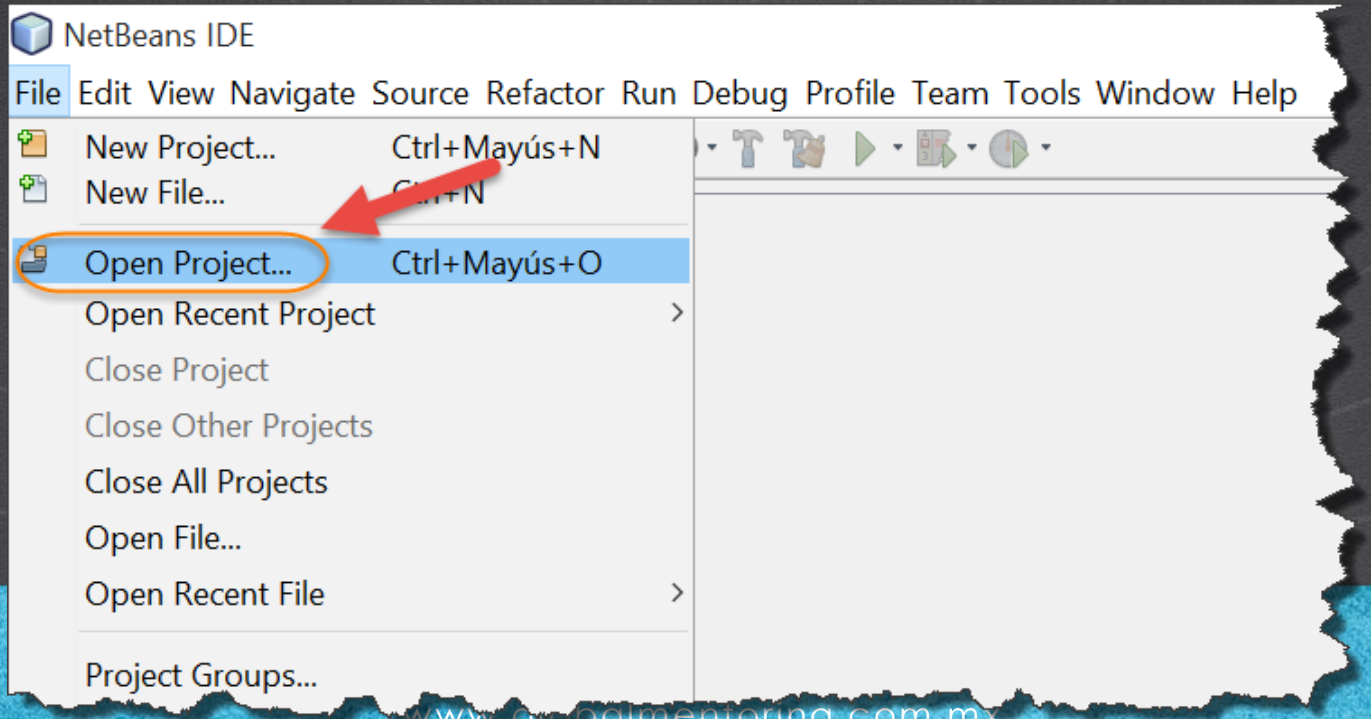
1. WE CLOSE ANY OTHER PROJECT

In case of having an open project, we close it:



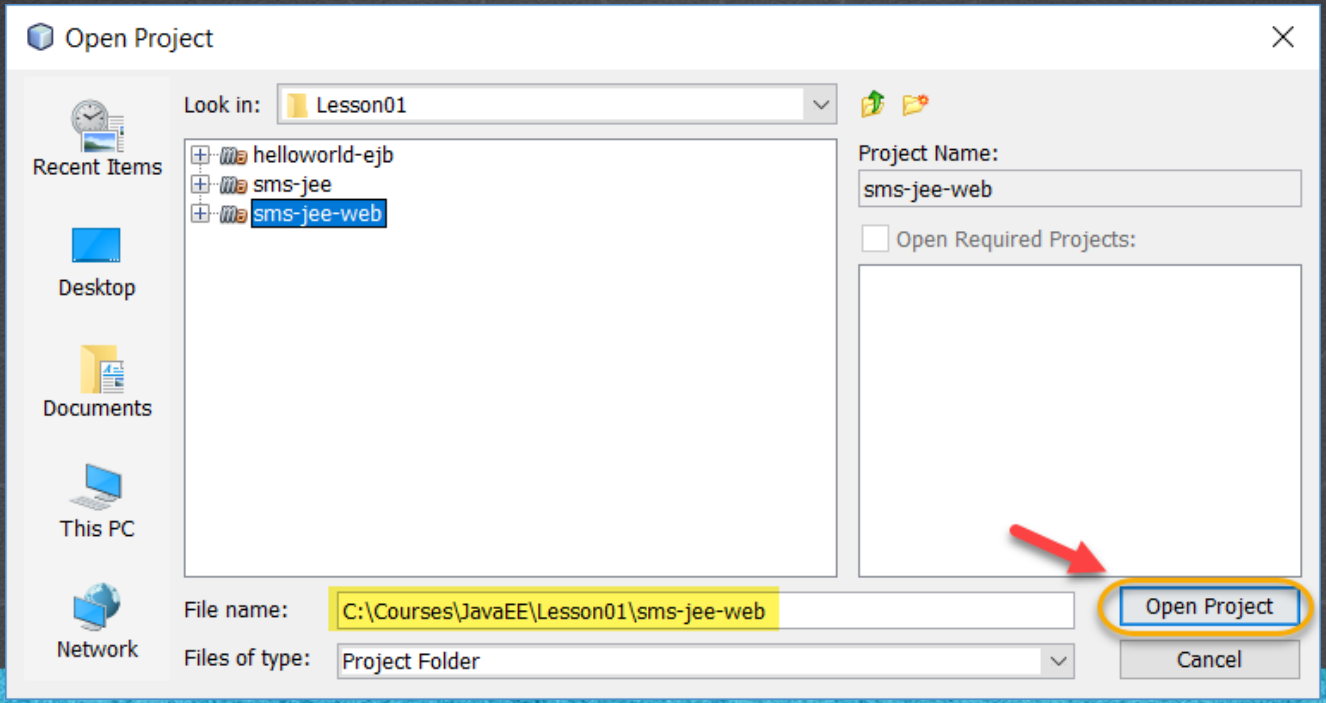
2. OPEN THE PROJECT

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



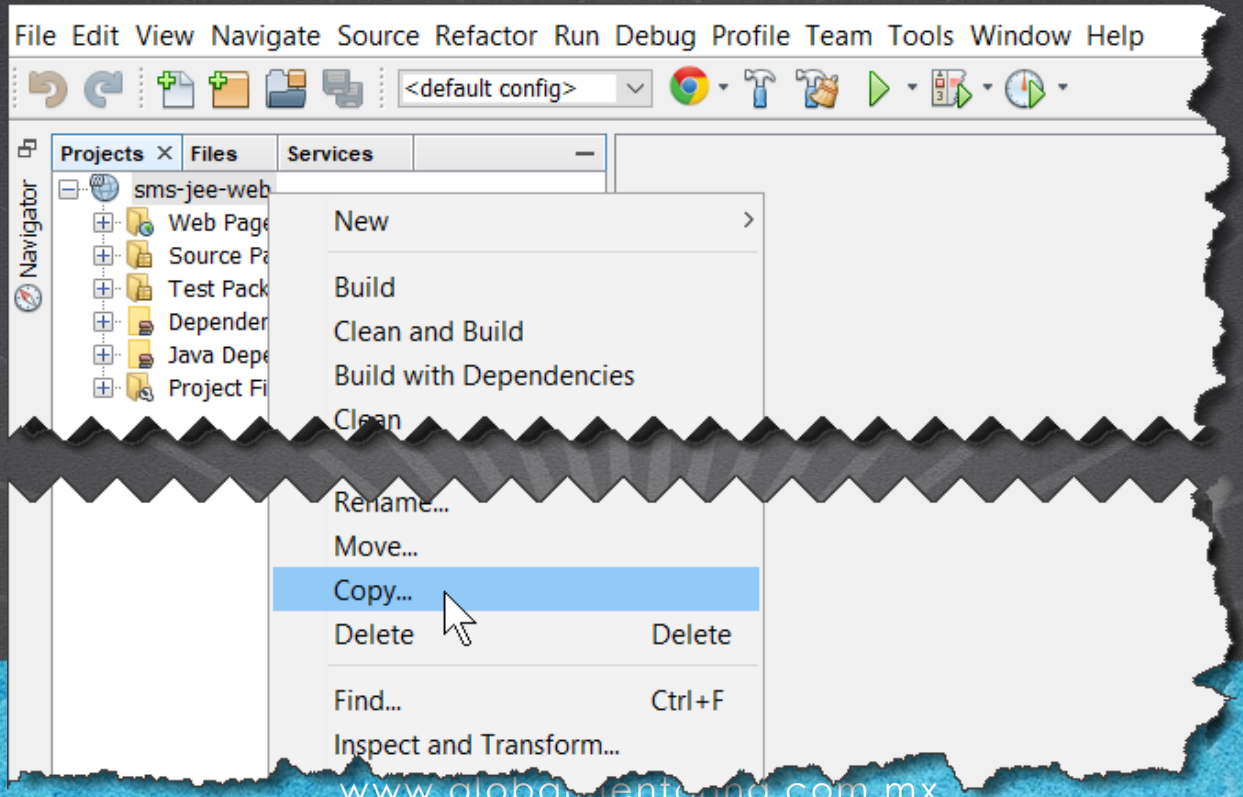
2. OPEN THE PROJECT

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



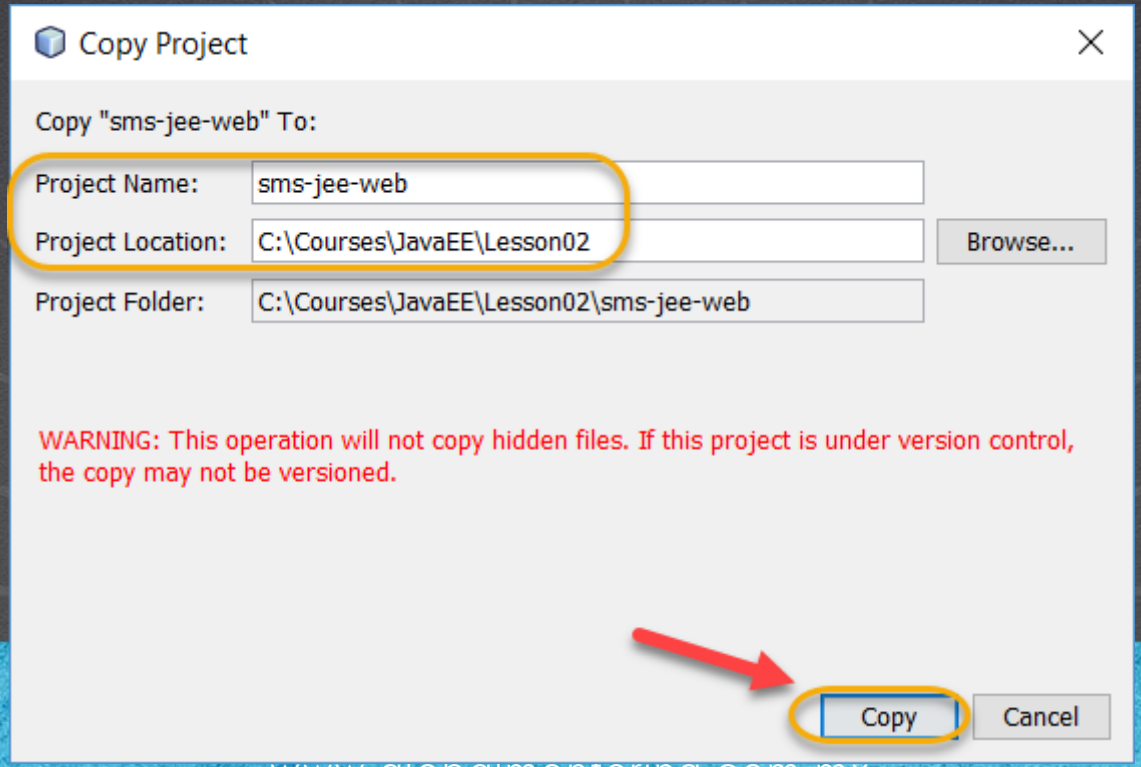
3. COPY THE PROJECT

We copy the project:



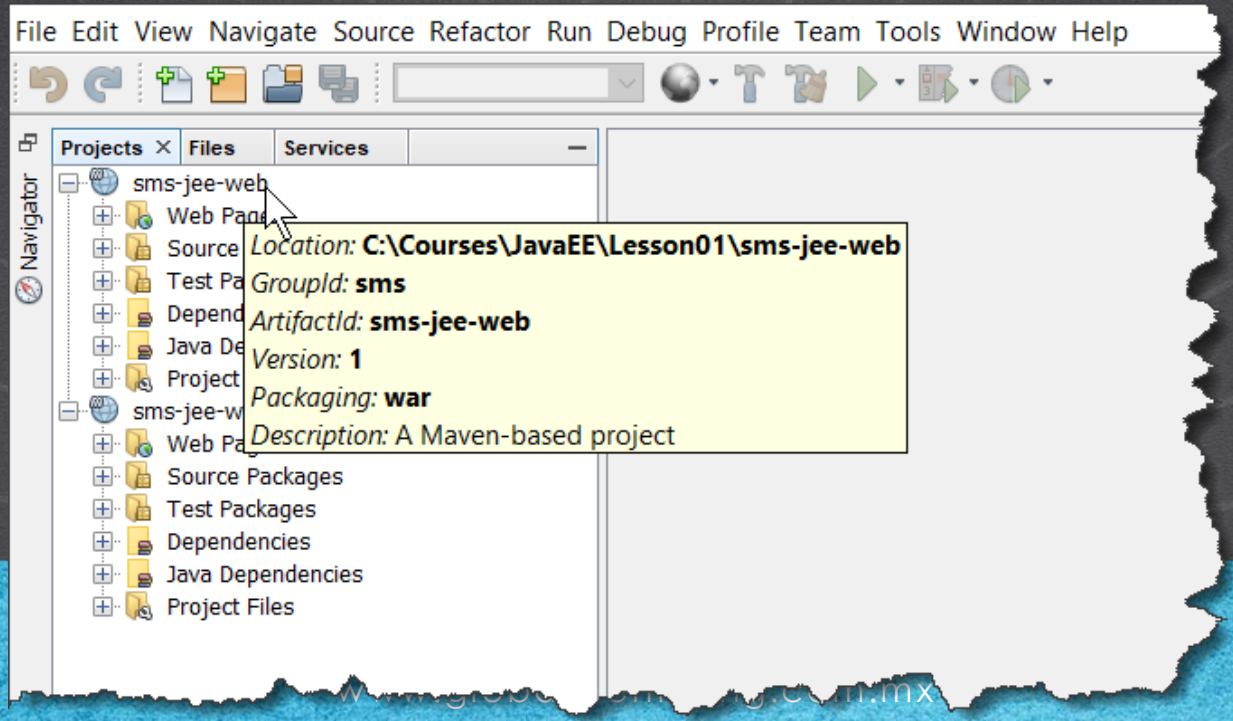
3. COPY THE PROJECT

Rename the Project and change the path:



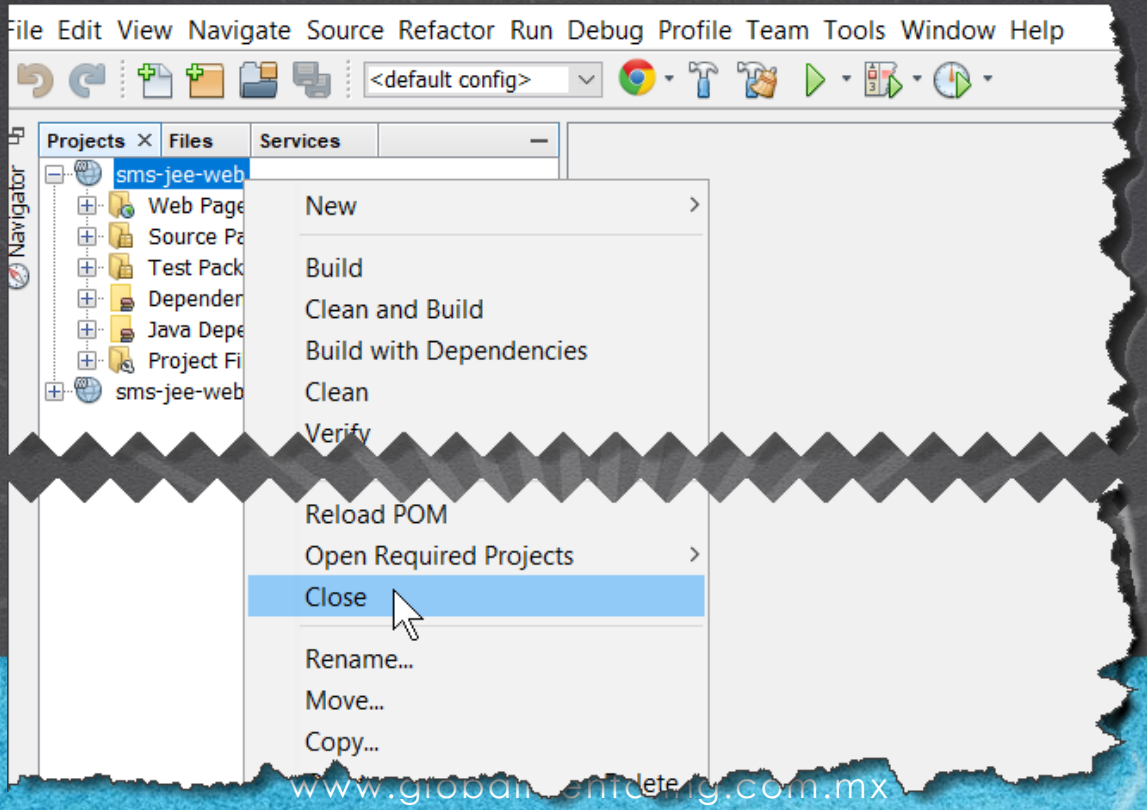
4. CLOSE THE PREVIOUS PROJECT

We closed the previous project and left the new one. If we position the cursor on the project we will see what the route indicates and with that we can decide which project to close (In this case we close the project of Lesson01):



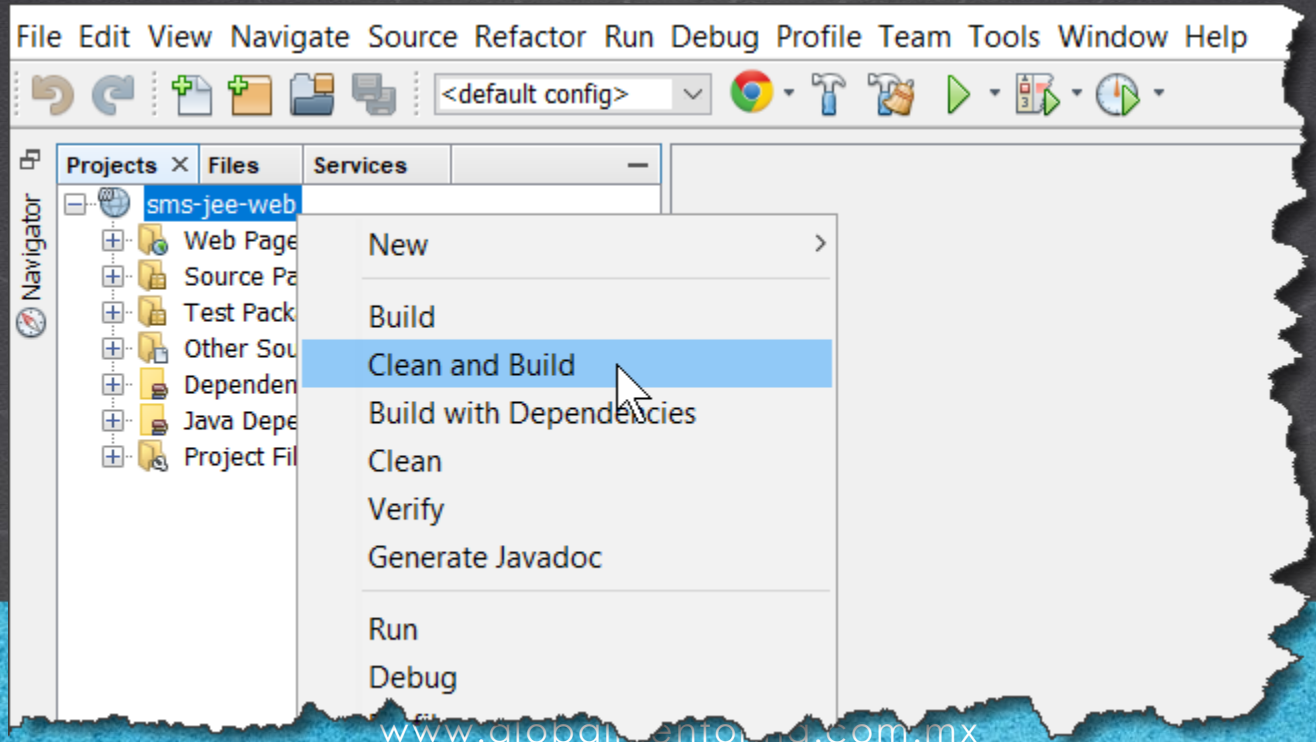
4. CLOSE THE PROJECT

Once identified what project we want to close, we close it :



5. EXECUTE CLEAN & BUILD

We make Clean & Build to download the new libraries if necessary:



5. EXECUTE CLEAN & BUILD

We observe the result of doing Clean & Build:

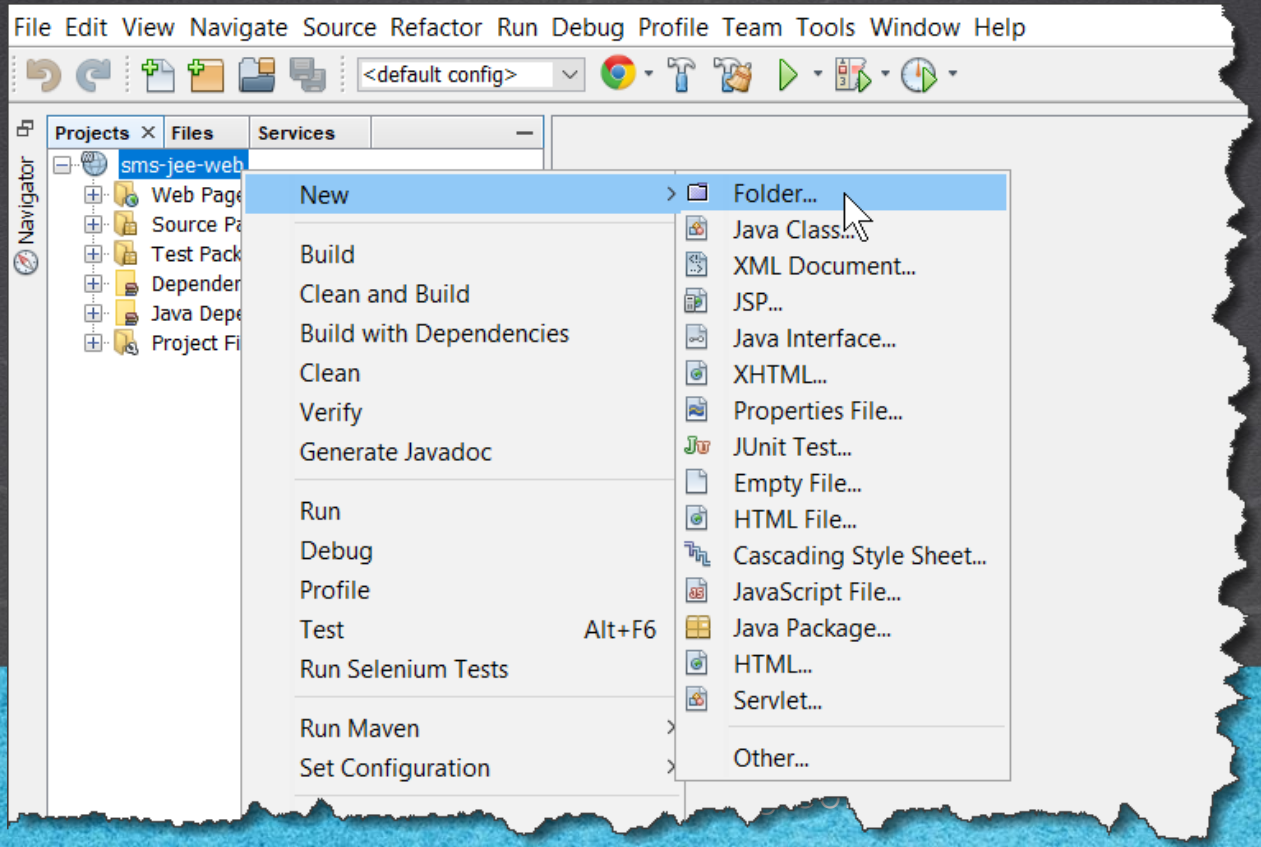
Output

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

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0


--- maven-war-plugin:2.6:war (default-war) @ sms-jee-web ---
Packaging webapp
Assembling webapp [sms-jee-web] in [C:\Courses\JavaEE\Lesson02\sms-jee-web\target\sms-jee-web-1]
Processing war project
Copying webapp resources [C:\Courses\JavaEE\Lesson02\sms-jee-web\src\main\webapp]
Webapp assembled in [71 msecs]
Building war: C:\Courses\JavaEE\Lesson02\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\Lesson02\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\Lesson02\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: 7.270s
Finished at: Wed Oct 17 18:11:46
Final Memory: 18M/293M
------------------------------------------------------------------------
```

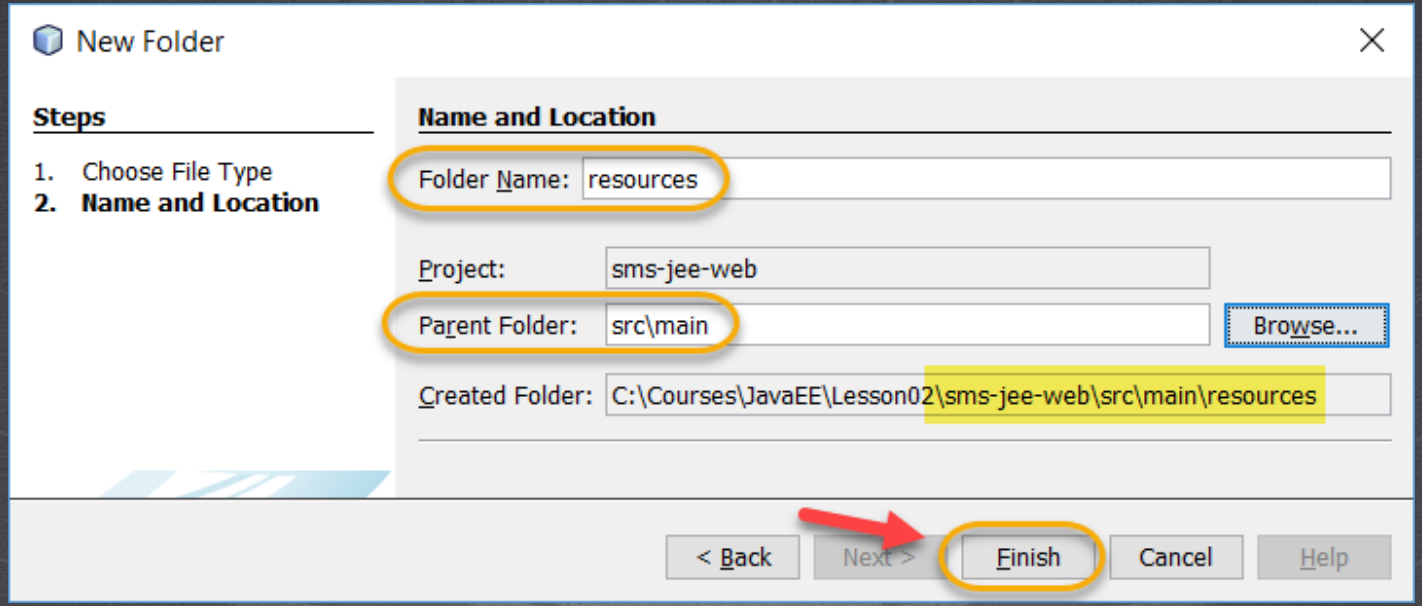
6. CREATE A FOLDER

We create a folder called: resources



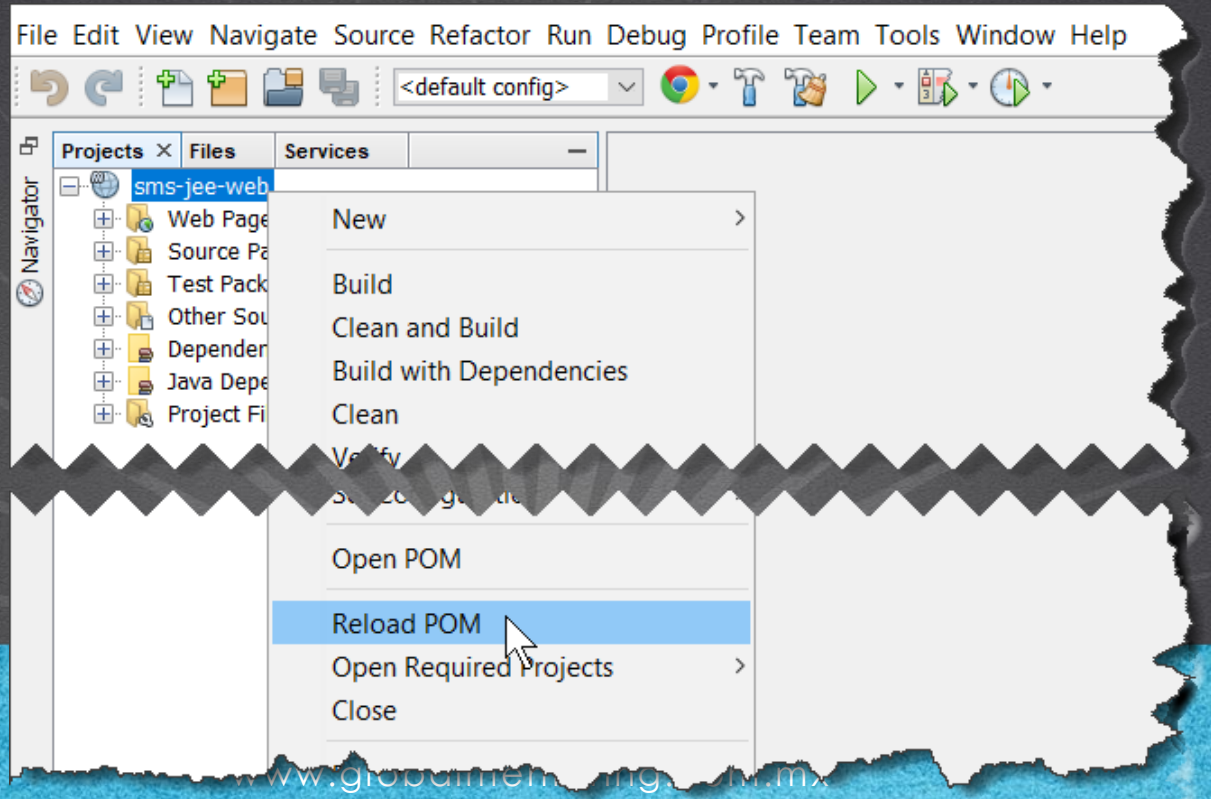
6. CREATE A FOLDER

We create a folder called: resources



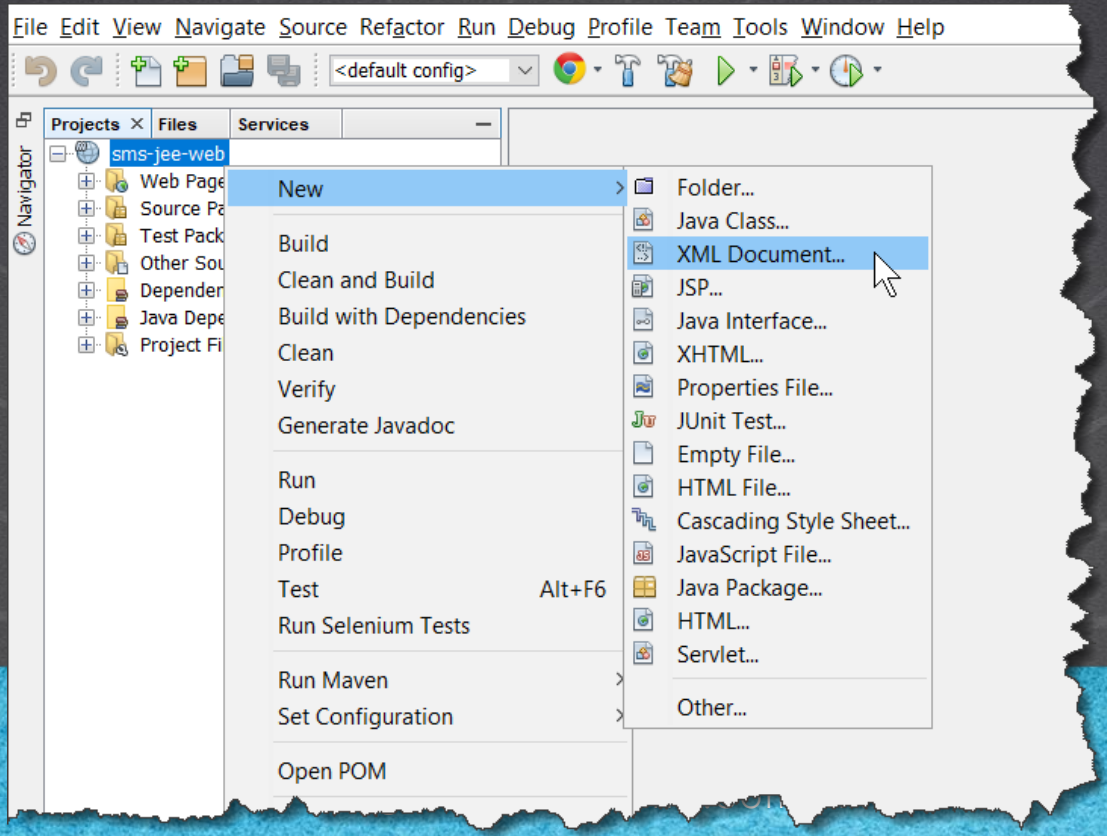
7. RELOAD THE PROJECT

We reload the project to see the new folder in case it is not displayed:



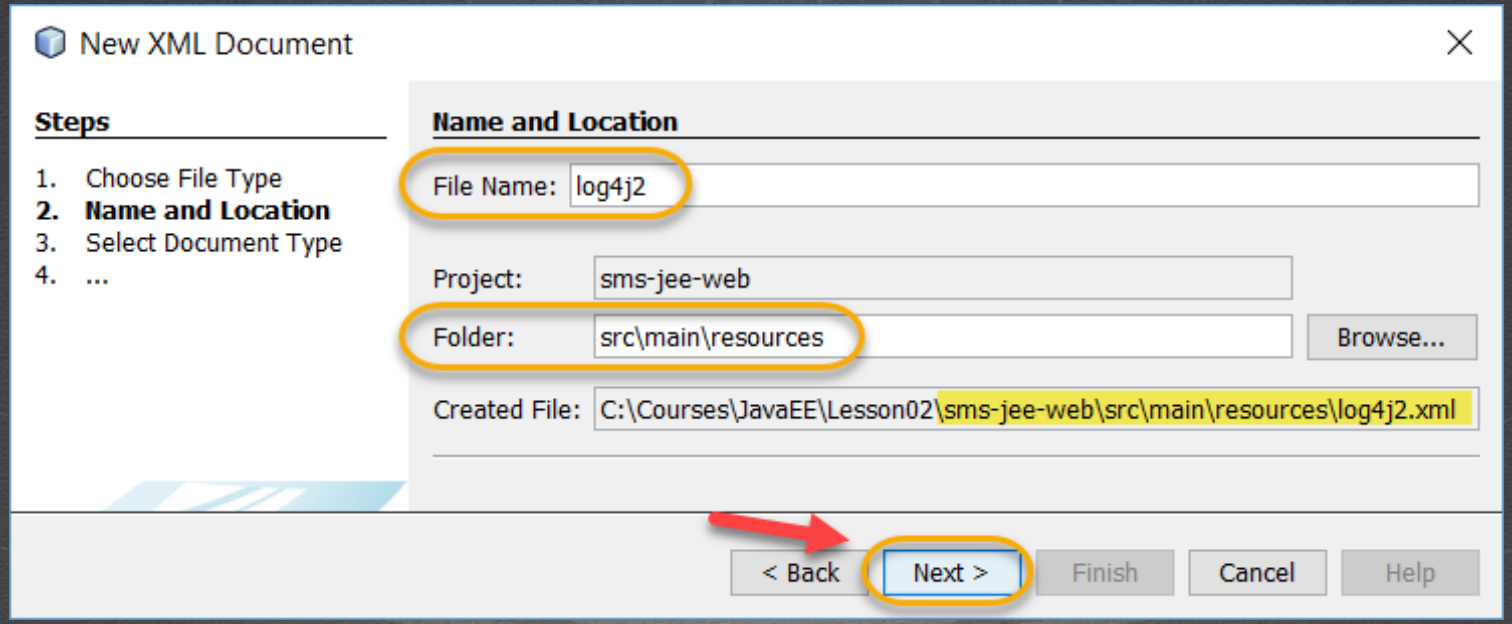
8. CREATE AN XML FILE

We create the log4j2.xml file:



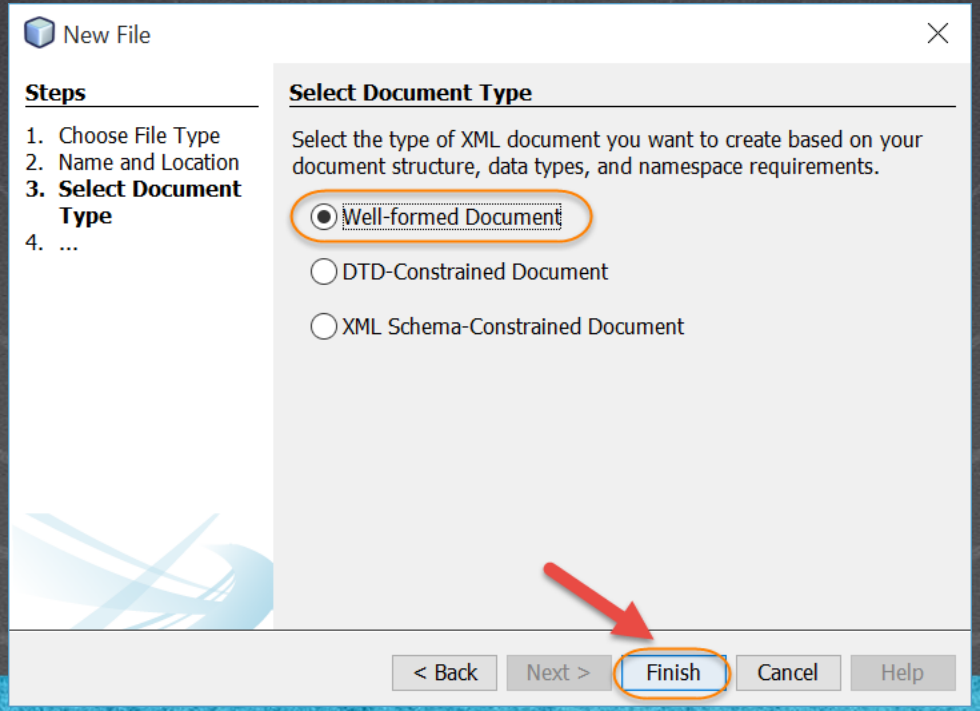
8. CREATE AN XML FILE

We create the log4j2.xml file:



8. CREATE AN XML FILE

We create the log4j2.xml file:



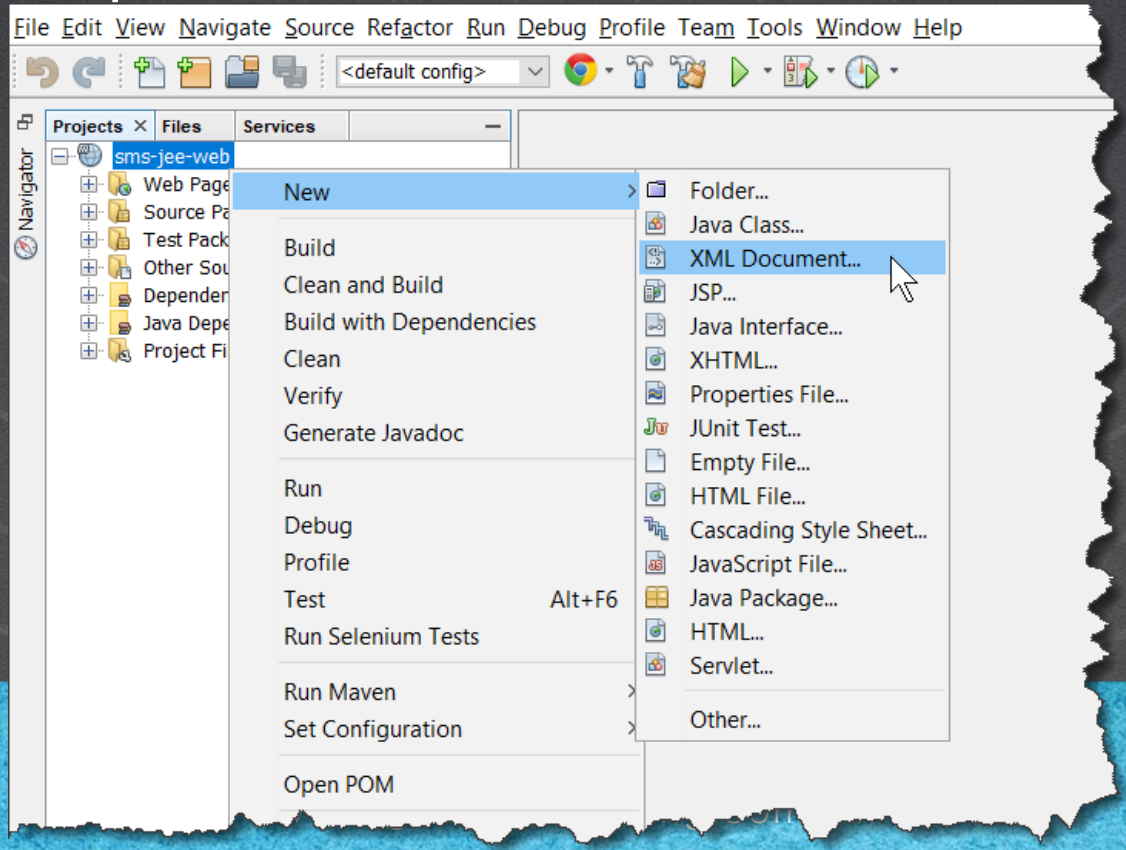
9. MODIFY THE FILE

log4j2.xml:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss} [%t] %-5level %logger{36} - %msg%n" />
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console" />
        </Root>
    </Loggers>
</Configuration>
```

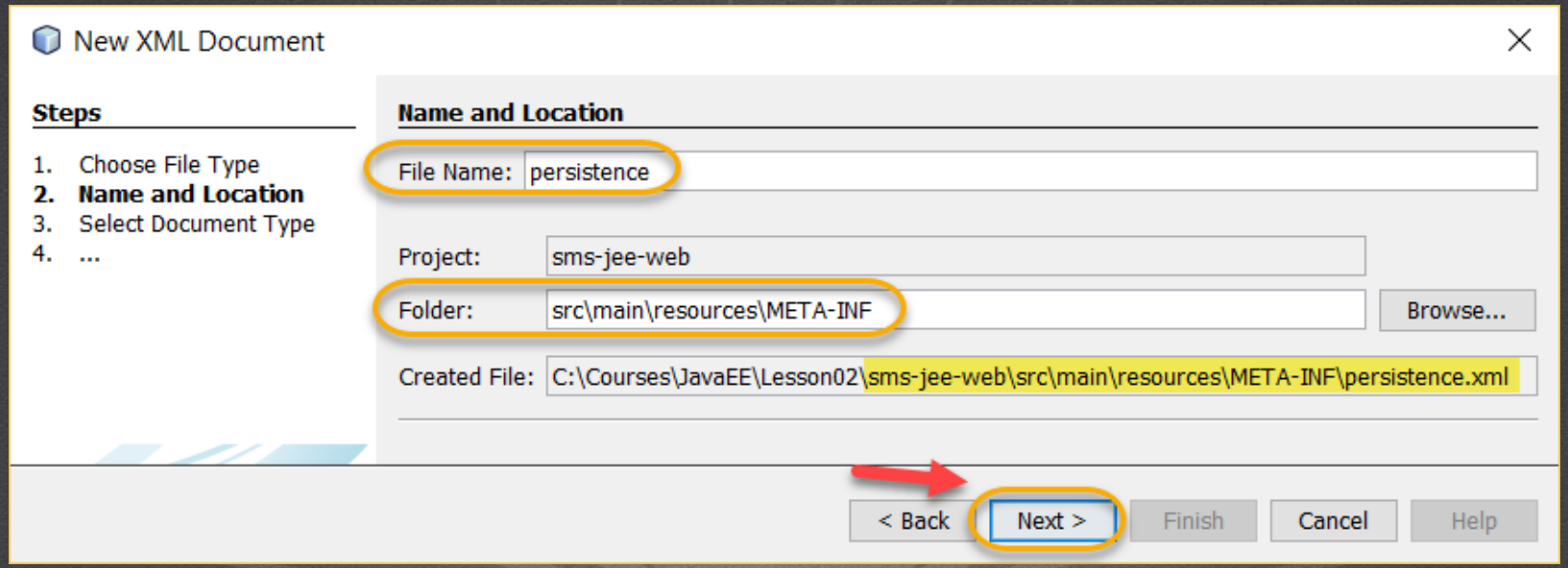
10. CREATE AN XML FILE

We create the persistence.xml file:



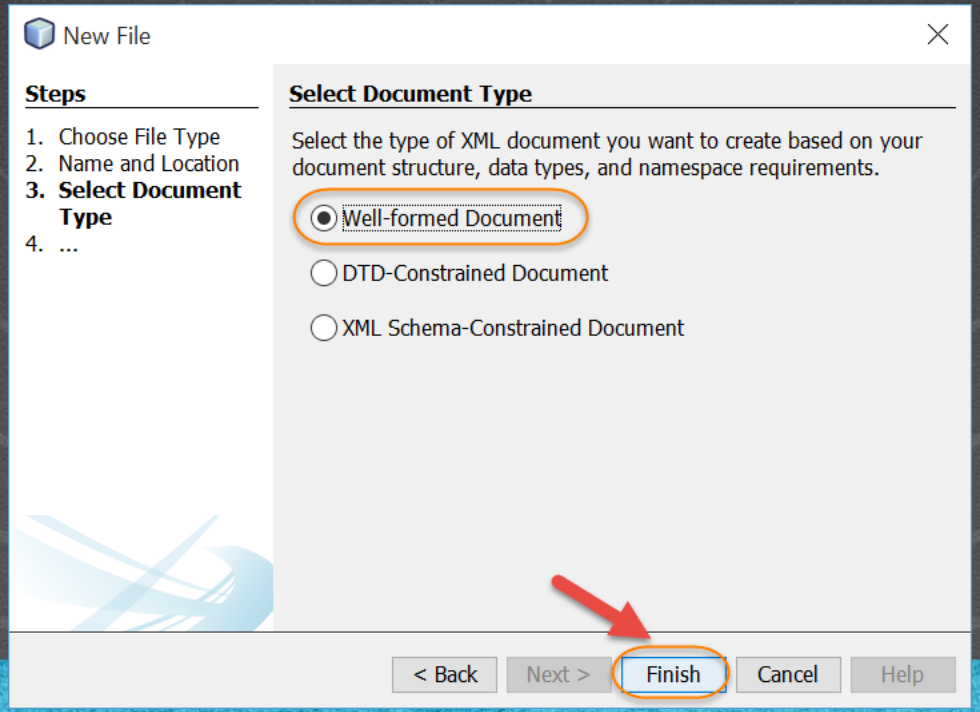
10. CREATE AN XML FILE

We create the persistence.xml file:



10. CREATE AN XML FILE

We create the persistence.xml file:



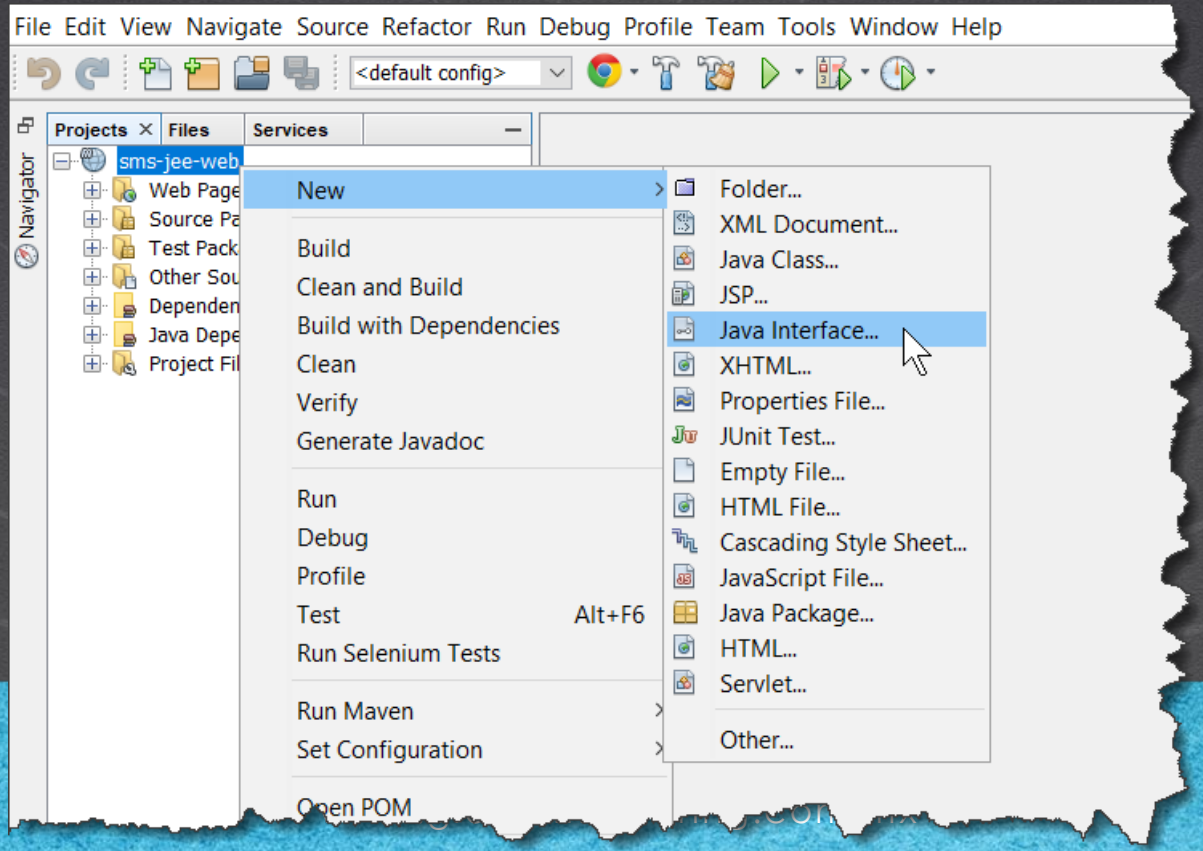
11. MODIFY THE FILE

persistence.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
    <persistence-unit name="PersonPU" transaction-type="JTA">
        <jta-data-source>jdbc/PersonDb</jta-data-source>
    </persistence-unit>
</persistence>
```

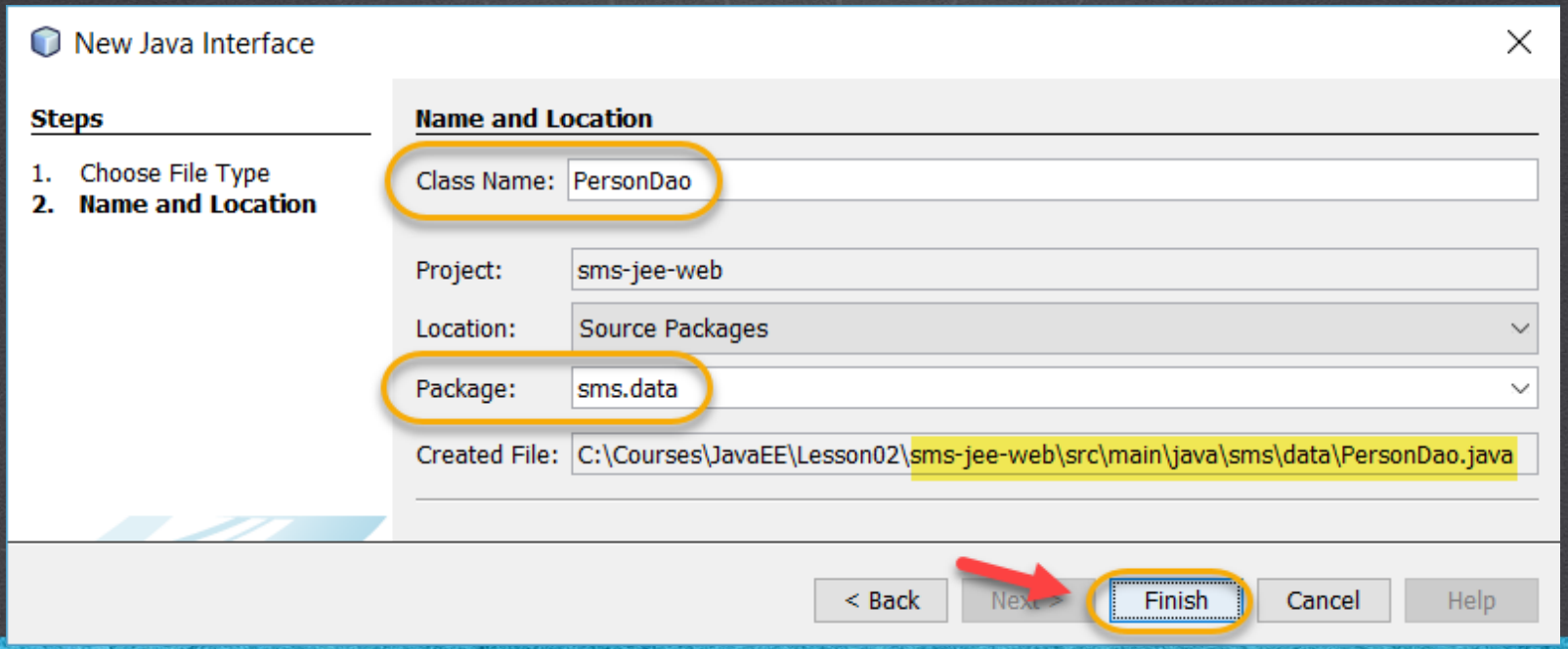
12. CREATE AN INTERFACE

We create a PersonDao interface:



12. CREATE AN INTERFACE

We create a PersonDao interface:



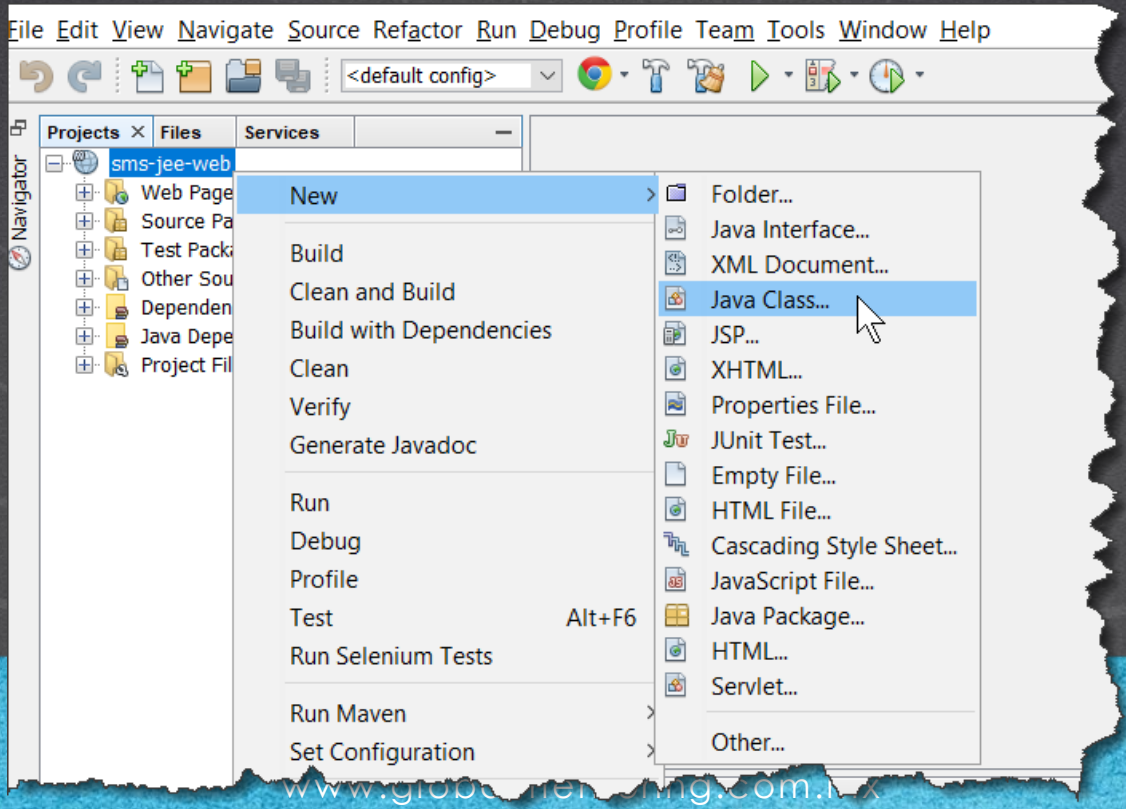
13. MODIFY THE FILE

PersonDao.java:

```
package sms.data;

import java.util.List;
import sms.domain.Person;

public interface PersonDao {

  public List<Person> findAllPeople();

  public Person findPerson(Person person);

  public void insertPerson(Person person);

  public void updatePerson(Person person);

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

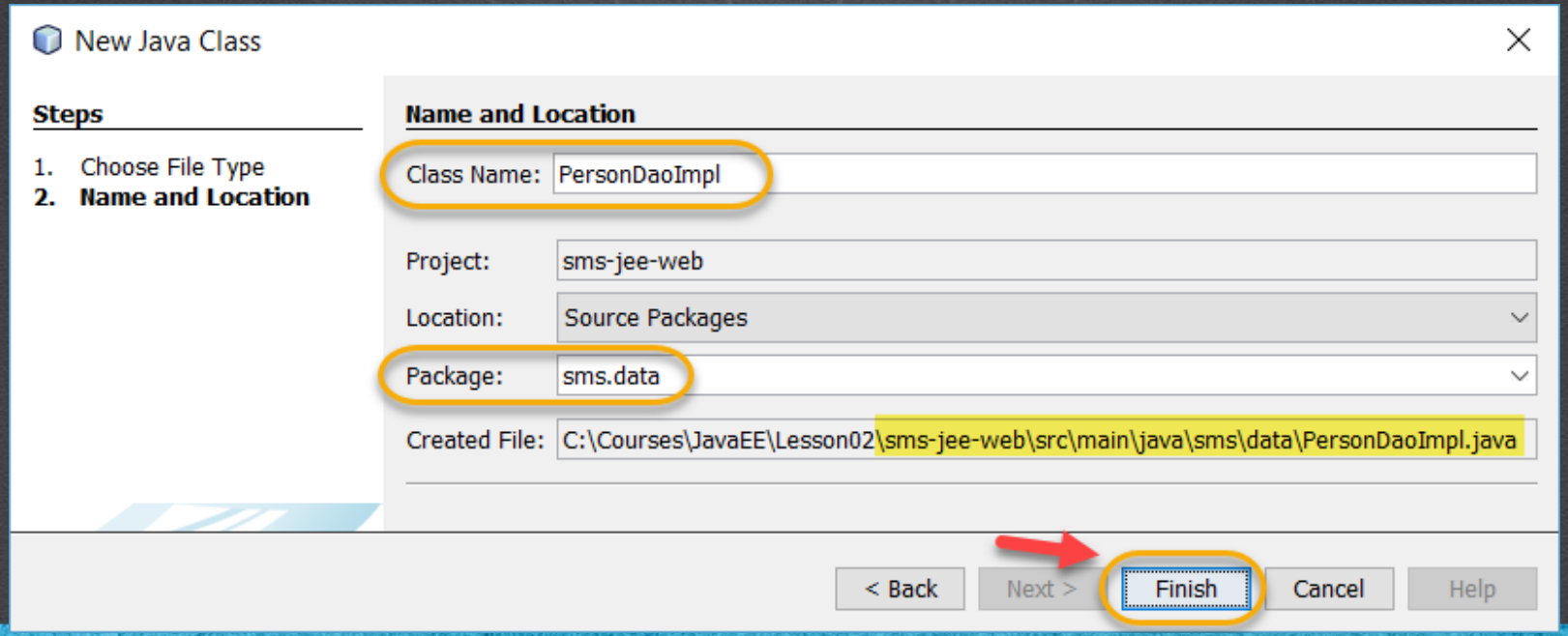
14. CREATE A JAVA CLASS

We create the PersonDaoImpl.java class:



14. CREATE A JAVA CLASS

We create the PersonDaoImpl.java class:



15. MODIFY THE FILE

PersonDaoImpl.java:

Click to download

```
package sms.data;

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

@Stateless
public class PersonDaoImpl implements PersonDao {

    @PersistenceContext(unitName = "PersonPU")
    EntityManager em;

    @Override
    public List<Person> findAllPeople() {
        return em.createNamedQuery("Person.findAll").getResultList();
    }

    @Override
    public Person findPerson(Person person) {
        return em.find(Person.class, person.getIdPerson());
    }
```

15. MODIFY THE FILE

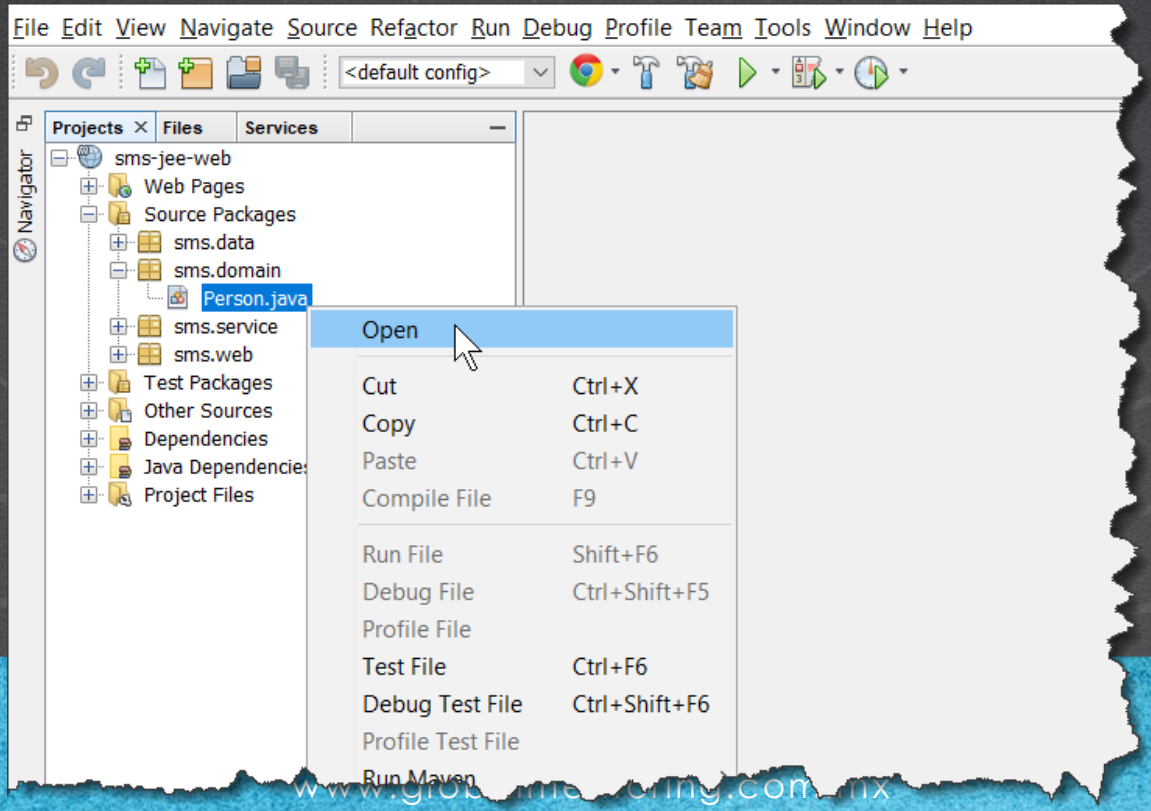
PersonDaoImpl.java:

Click to download

```
    @Override
    public void insertPerson(Person person) {
        em.persist(person);
    }

    @Override
    public void updatePerson(Person person) {
        em.merge(person);
    }

    @Override
    public void deletePerson(Person person) {
        em.remove(em.merge(person));
    }
}
```

16. MODIFY THE JAVA CLASS

Modify the Person.java to conver it to an Entity class:



16. MODIFY THE FILE

Person.java:

Click to download

```
package sms.domain;

import java.io.Serializable;
import javax.persistence.*;

@Entity
@NamedQueries({
    @NamedQuery(name = "Person.findAll", query = "SELECT p FROM Person p ORDER BY p.idPerson")})
@Table(name = "person")
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 idPersona, String name) {
        this.idPerson = idPersona;
        this.name = name;
    }
```

16. MODIFY THE FILE

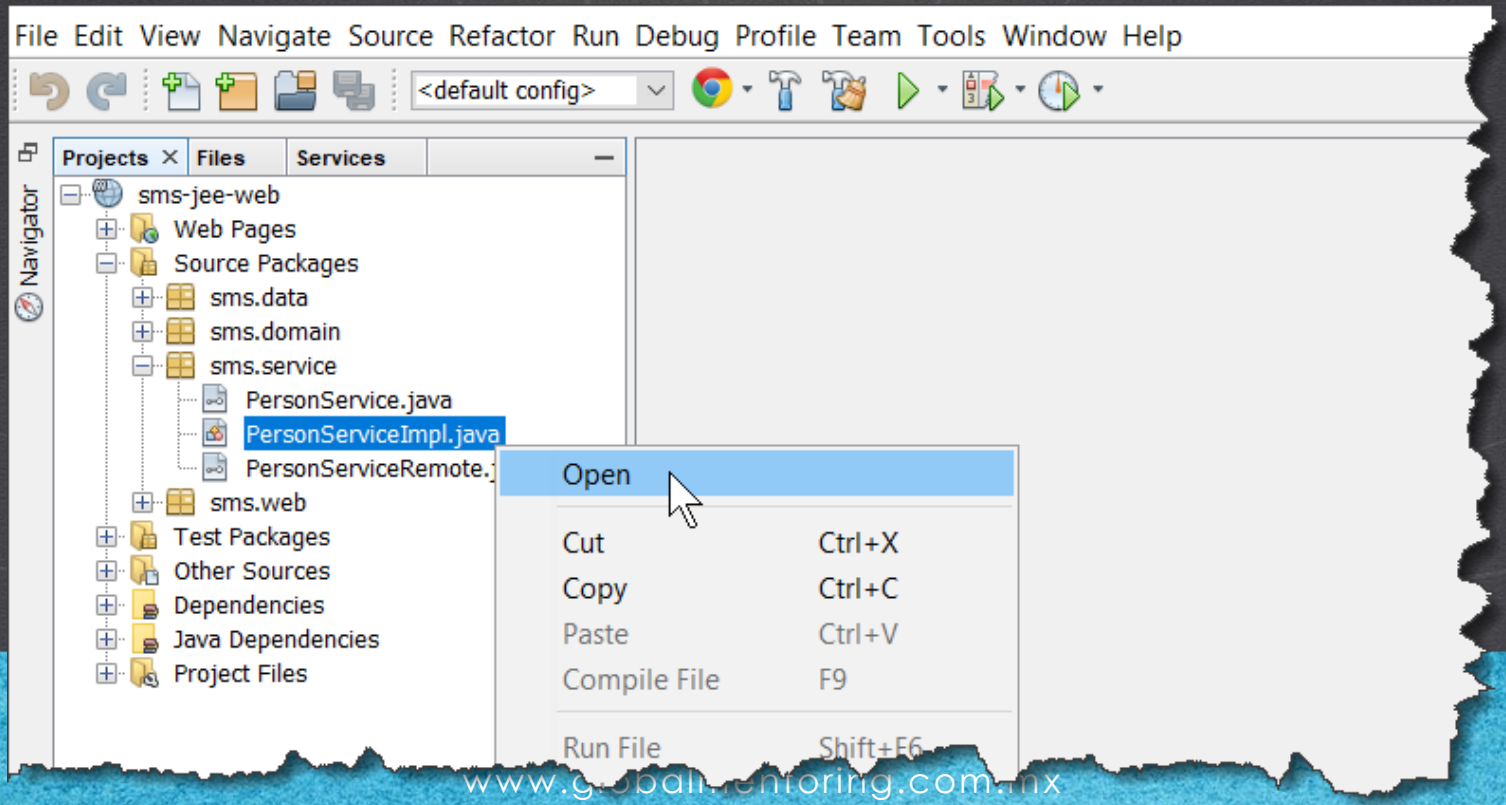
Person.java:

Click to download

```
    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 + '}';
    }

}
```

17. MODIFY THE JAVA CLASS

Modify the PersonServiceImpl.java:



17. MODIFY THE FILE

PersonServiceImpl.java:

Click to download

```
package sms.service;

import java.util.List;
import javax.ejb.Stateless;
import javax.inject.Inject;
import sms.data.PersonDao;
import sms.domain.Person;

@Stateless
public class PersonServiceImpl implements PersonServiceRemote, PersonService {

    @Inject
    private PersonDao personDao;

    @Override
    public List<Person> listPeople() {
       return personDao.findAllPeople();
    }

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

17. MODIFY THE FILE

PersonServiceImpl.java:

Click to download

```
    @Override
    public void addPerson(Person person) {
        personDao.insertPerson(person);
    }

    @Override
    public void modifyPerson(Person person) {
        personDao.updatePerson(person);
    }

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

18. JTA CONNECTION CONFIGURATION

We can see that the same name configured in Glassfish is the name used in the persistence.xml file:

JDBC Resources

JDBC resources provide applications with a means to connect to a database.

Resources (4)

New... Delete Enable Disable

Select	JNDI Name	Logical JNDI Name
	jdbc/PersonDb	
	jdbc/__TimerPool	
	jdbc/__default	java:comp/DefaultDataSource
	jdbc/sample	

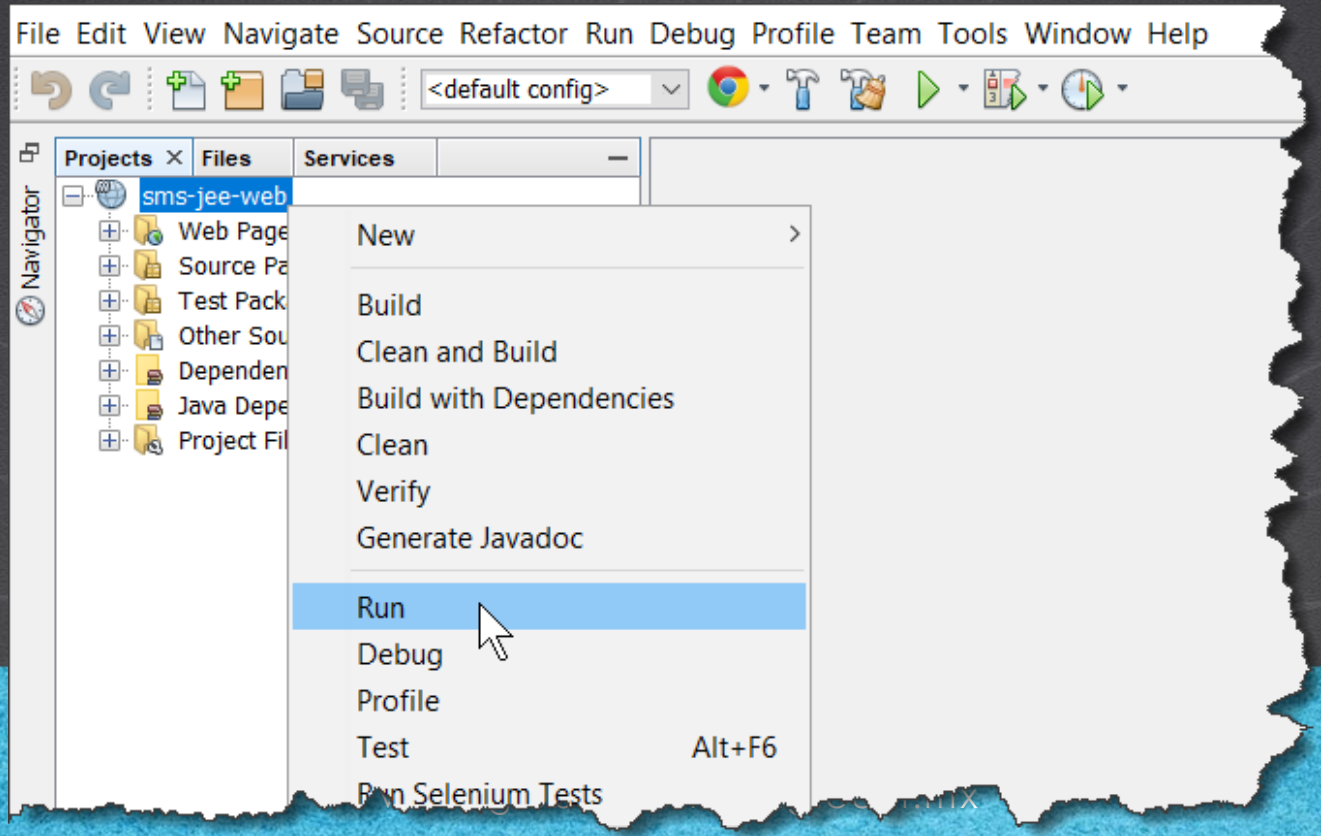
persistence.xml

Source History

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xs
3      <persistence-unit name="PersonaPU" transaction-type="JTA">
4          <jta-data-source>jdbc/PersonDb</jta-data-source>
5      </persistence-unit>
6  </persistence>
```

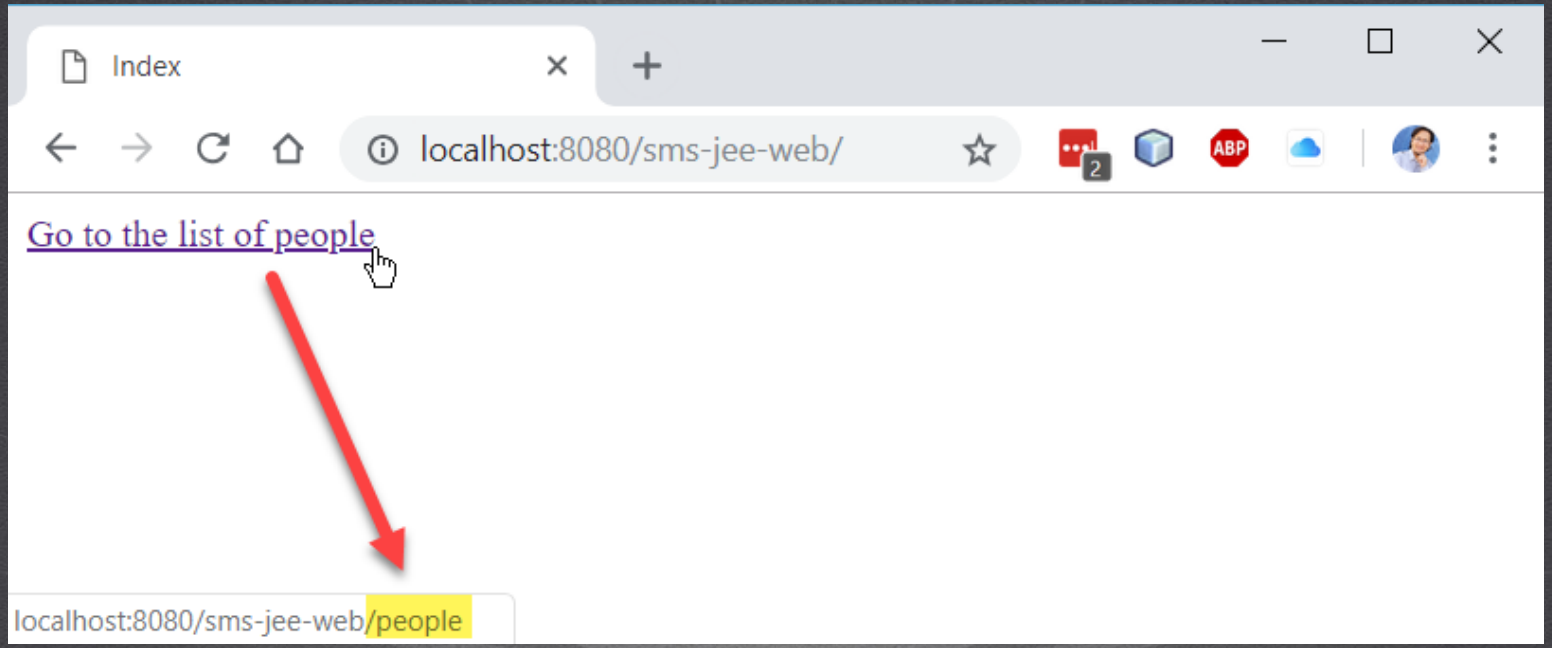
19. EXECUTE THE APPLICATION

We execute our Web application. This will automatically display our application in Glassfish:



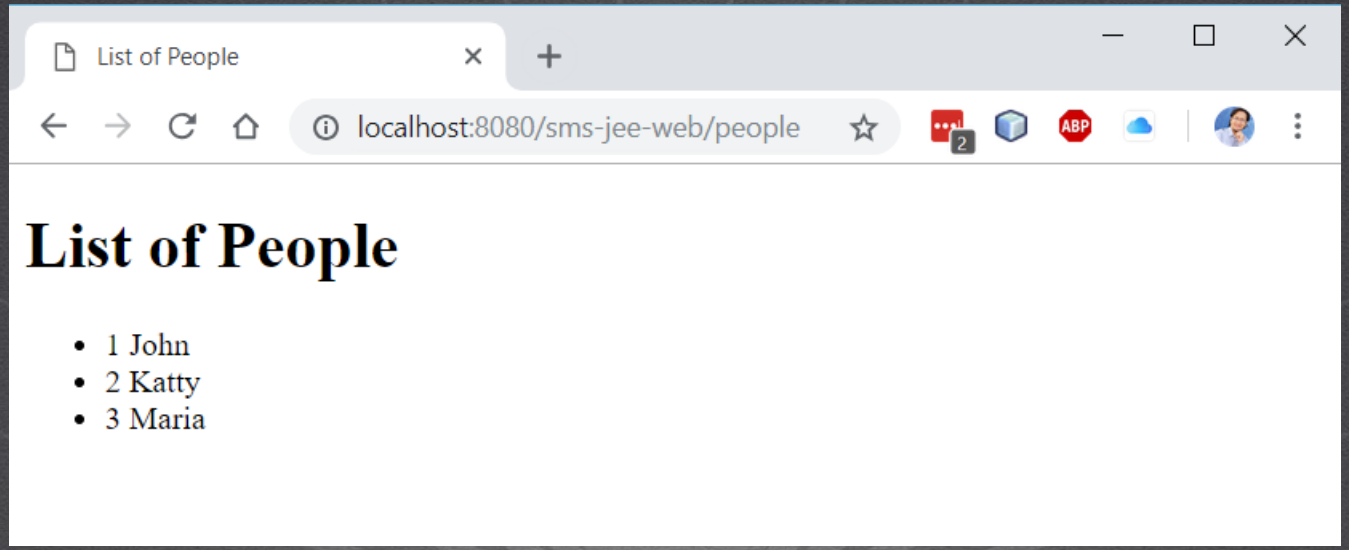
19. EXECUTE THE APPLICATION

Execute the application:



19. EXECUTE THE APPLICATION

Execute the application:



EXERCISE CONCLUSION

- With this exercise we have added JPA to our sms-jee-web exercise.
- In this way we can communicate with the mysql database, in addition to configuring our connection to the database using JTA (Java Transaction API), which allows us to delegate the connection data to Glassfish and thus avoid configuring the connection based on data from our application.
- Afterwards, we integrate the DAO layer with the Service layer using the @Inject annotation.
- There was no need to make any changes to the Web layer, since the service layer interface did NOT change, only the implementation of it. This is one of the advantages of using interfaces, since the implementation of the Service layer went from being data in hard code, to real information of a database, and it was completely transparent for the Web layer and for the final client.
- So in addition to technologies like EJB and JPA, we are also applying the best practices and several design patterns that allow us to create robust and scalable Web applications and Enterprise Applications that are easier to maintain, among several other features.

ONLINE COURSE

JAVA EE
JAKARTA EE

By: Eng. Ubaldo Acosta