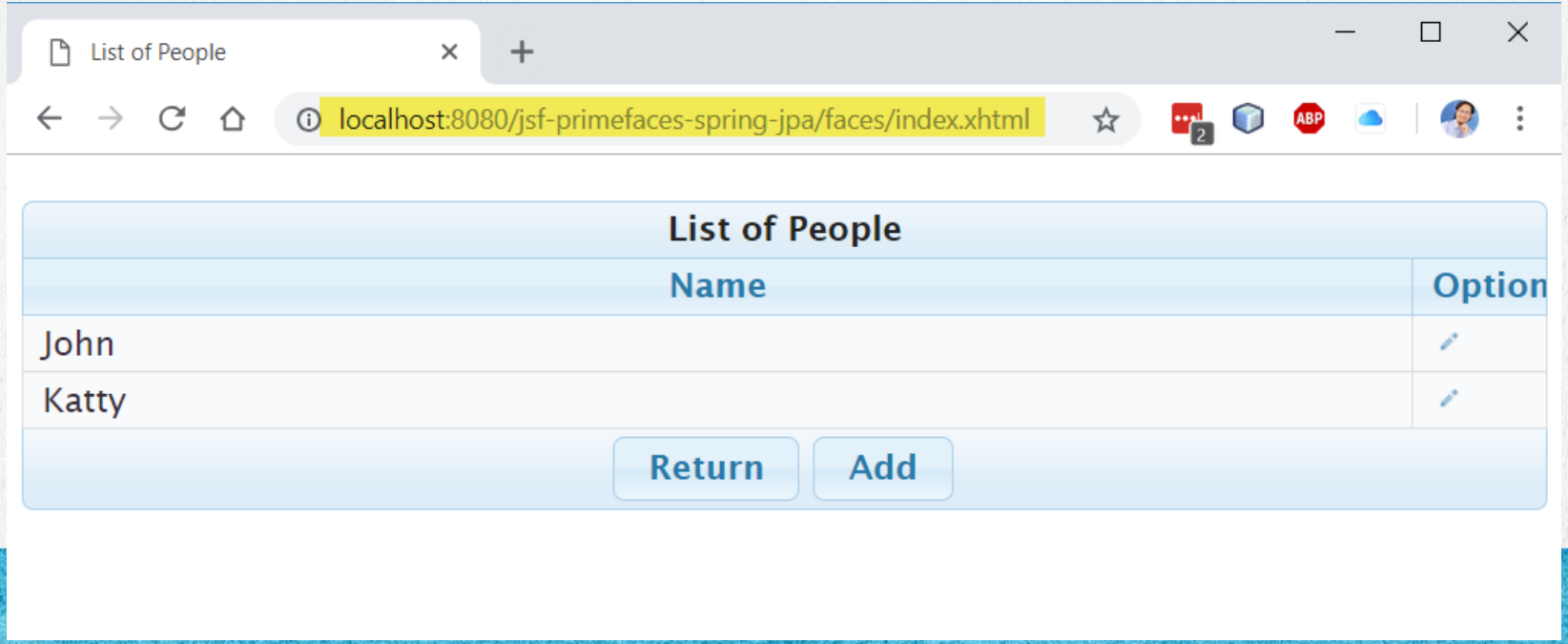
SPRING FRAMEWORK COURSE

EXERCISE

JSF/PRIMEFACES+SPRING+JPA INTEGRATION

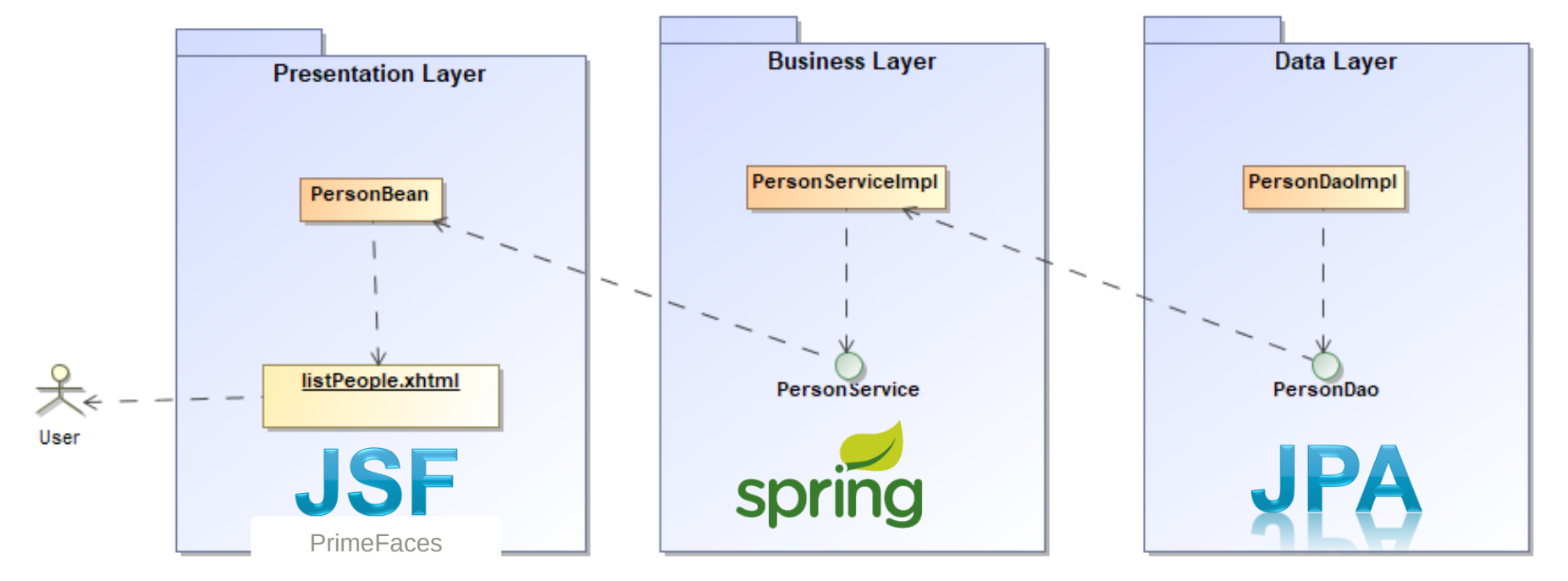
EXERCISE OBJECTIVE

- The objective of the exercise is to configure a project to integrate the Spring framework with JPA. We will rely on Maven for the creation of the project. The result should be similar to the following :



CLASS DIAGRAM

- This is the Exercise Class Diagram, where you can see the presentation layer, business layer and the data layer implemented by JSF, Spring and JPA respectively.



GLASSFISH AS AN APPLICATION SERVER

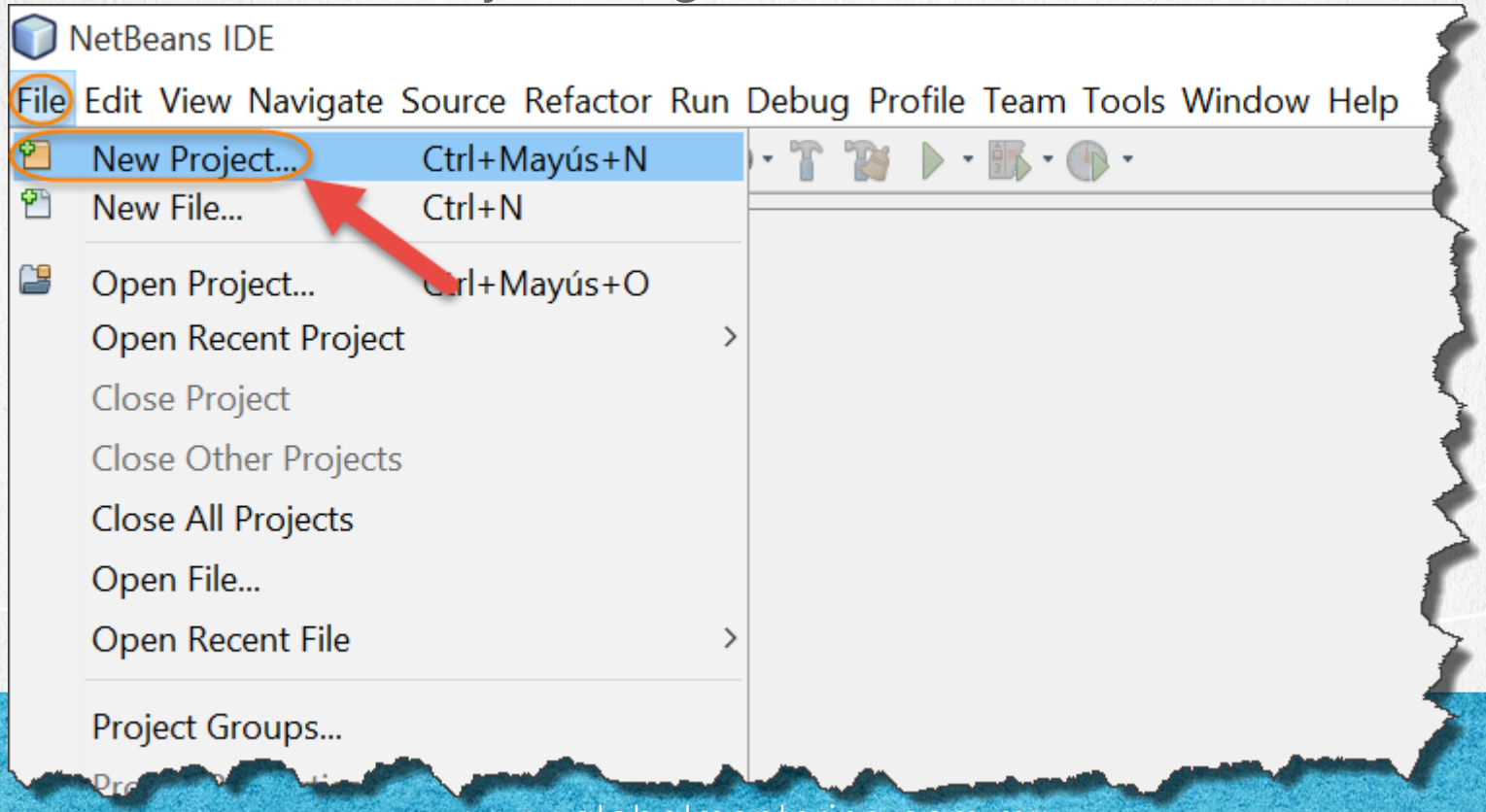
- For this exercise we are going to use the Glassfish server. At the beginning of the course we gave the guides to configure the Glassfish application server, but if it is not yet installed we provide them again with the guides that we will need for this exercise.

Glassfish installation:
http://icursos.net/en/Installations/CJ-B-Exercise-05-InstallingGlassfish.pdf

- **Connection configuration of MySql in Glassfish**, including a pool of connections. This configuration is what we will use to connect to our database and thus separate the connection data to the database and not have them directly in the Java code, but in the Glassfish application server via JTA:
http://icursos.net/en/Installations/CJ-B-Exercise-04-JTAGlassfish.pdf

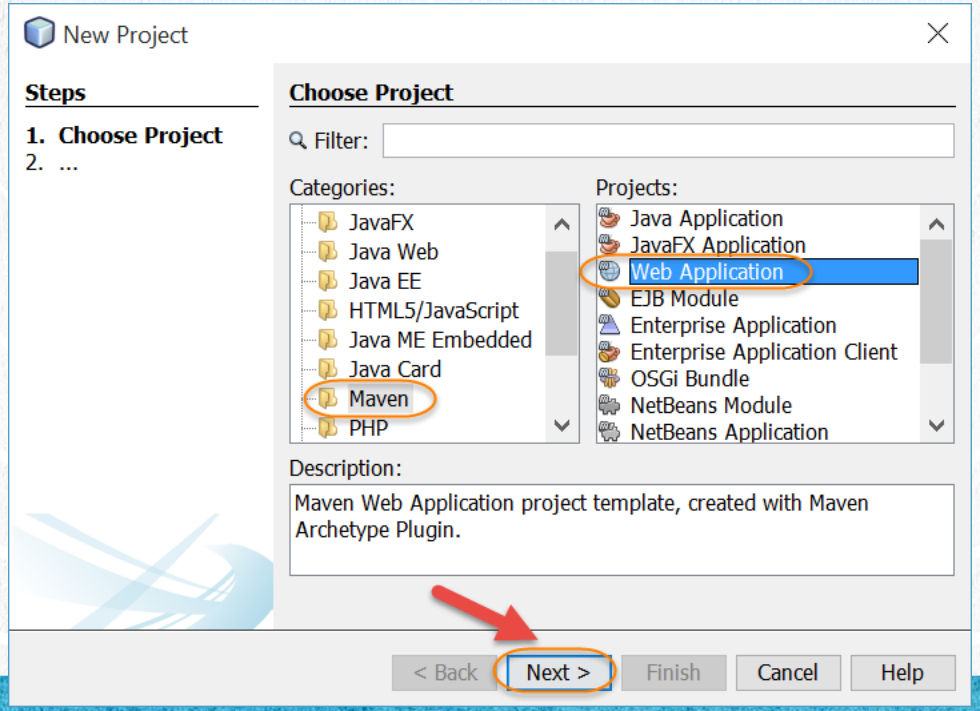
1. CREATE A NEW PROJECT

Create a new Java Web Project using Maven:



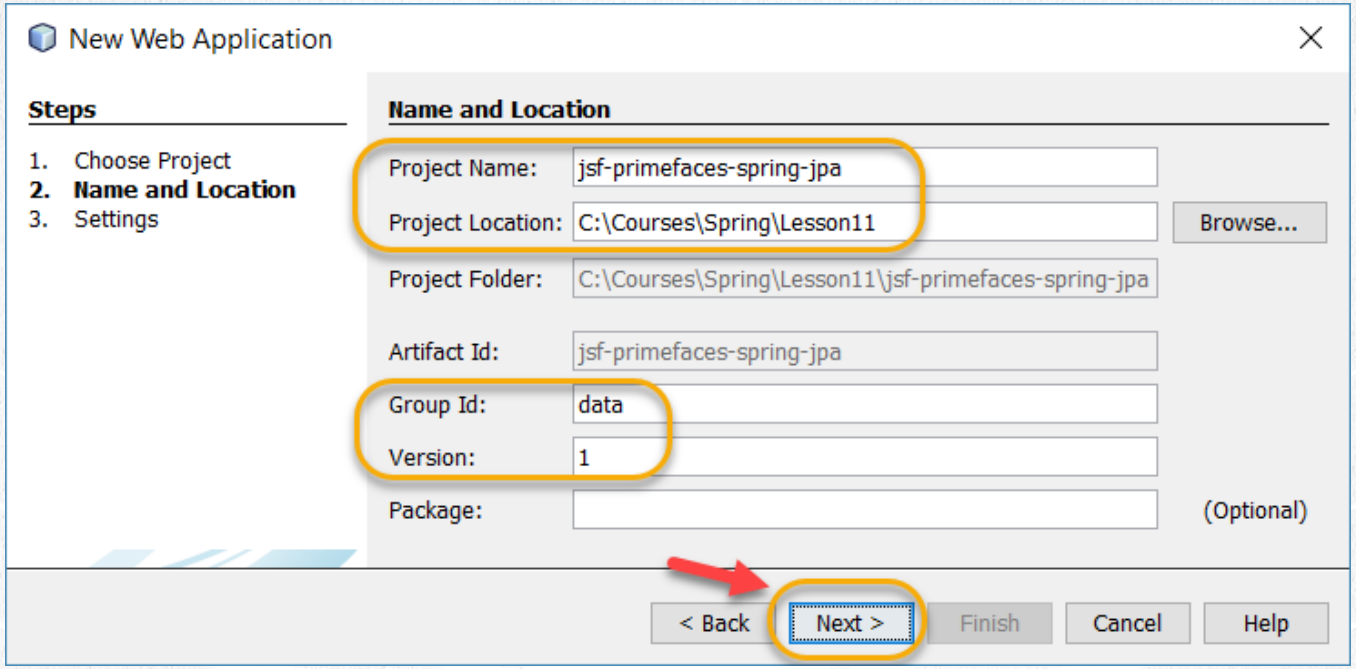
1. CREATE A NEW PROJECT

Create a new Java Web Project using Maven:



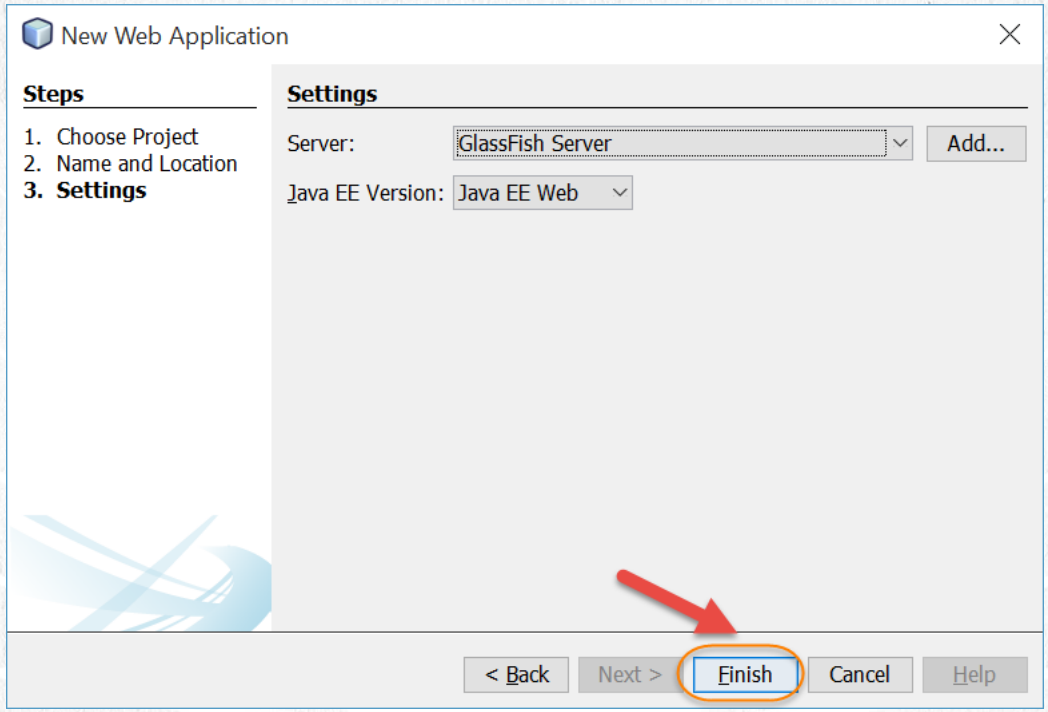
1. CREATE A NEW PROJECT

We write the following values:



1. CREATE A NEW PROJECT

We click on Finish:



2. ADD LIBRARIES TO THE PROJECT

We add the following .jar libraries to the pom.xml file:

- spring-core
- spring-context
- spring-test
- spring-orm
- spring-web
- spring-webmvc
- javaee-web-api
- primefaces
- primefaces-themes
- mysql
- junit
- log4j

2. 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>data</groupId>
    <artifactId>jsf-primefaces-spring-jpa</artifactId>
    <version>1</version>
    <packaging>war</packaging>

    <name>jsf-primefaces-spring-jpa</name>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <spring.version>5.1.0.RELEASE</spring.version>
        <log4j.version>2.11.1</log4j.version>
        <junit.version>5.3.1</junit.version>
    </properties>
    <dependencies>
        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
```

2. MODIFY THE CODE

pom.xml:

Click to download

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${spring.version}</version>
    <scope>test</scope>
    <type>jar</type>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
</dependency>
```

2. MODIFY THE CODE

pom.xml:

Click to download

```
<!-- Log4j -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>${log4j.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>${log4j.version}</version>
</dependency>
<!-- MySql -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.47</version>
</dependency>
<!--JPA-->
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa</artifactId>
    <version>2.7.3</version>
</dependency>
```

2. MODIFY THE CODE

pom.xml:

Click to download

```
        <!--Java EE-->
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-web-api</artifactId>
            <version>8.0</version>
            <scope>provided</scope>
        </dependency>
        <!-- prime faces-->
        <dependency>
            <groupId>org.primefaces</groupId>
            <artifactId>primefaces</artifactId>
            <version>6.2</version>
        </dependency>
        <dependency>
            <groupId>org.primefaces.themes</groupId>
            <artifactId>all-themes</artifactId>
            <version>1.0.10</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>
```

2. MODIFY THE CODE

pom.xml:

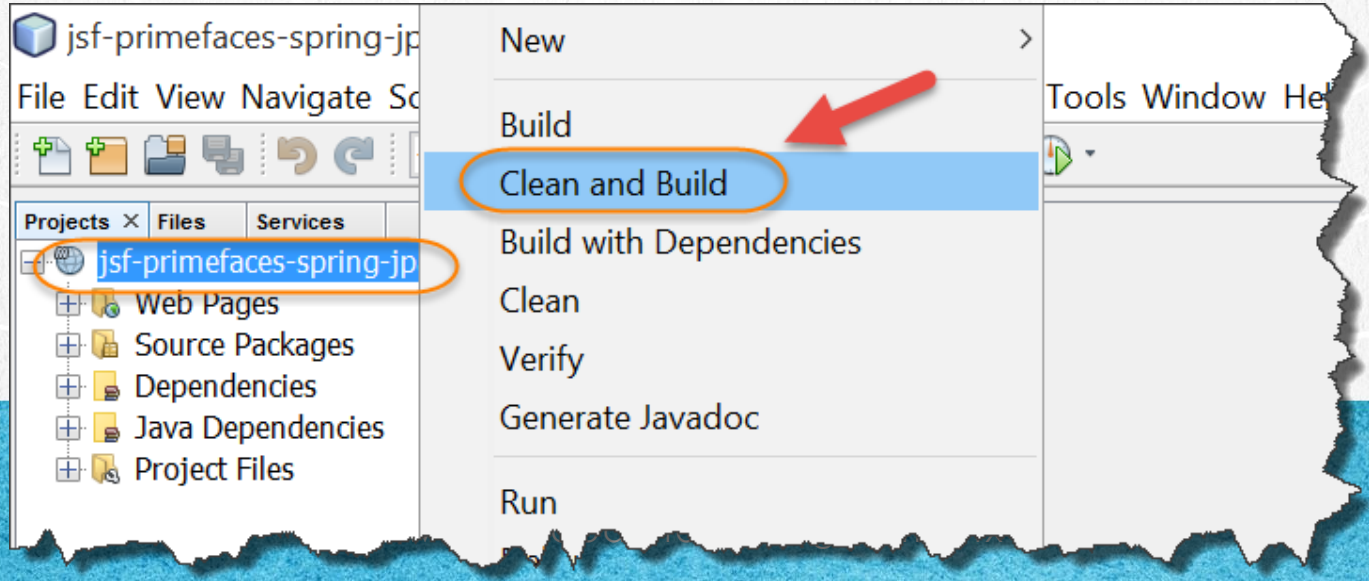
Click to download

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.3</version>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.7.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

3. MAVEN REPOSITORY UPDATE

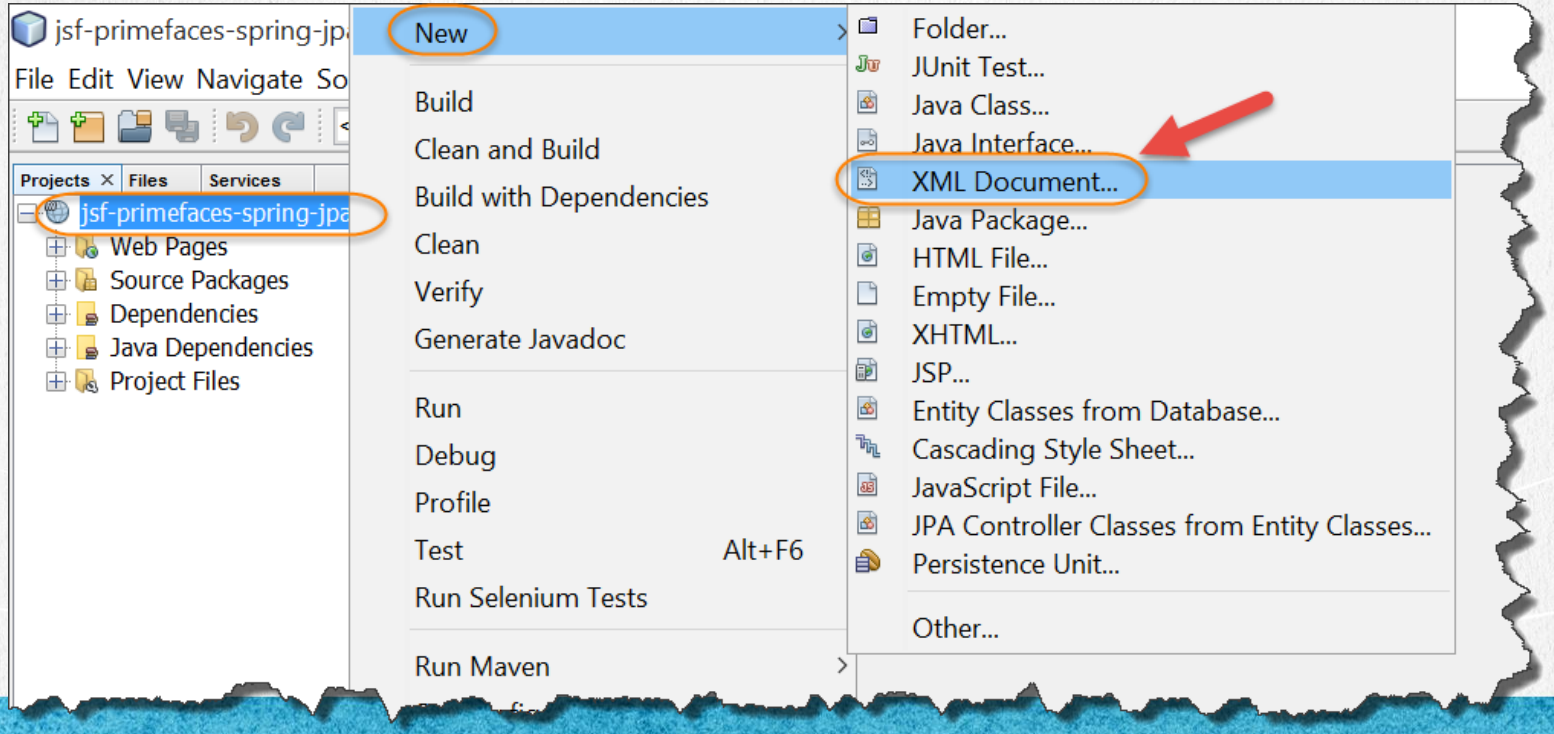
To update the Maven repository, which will be responsible for administering the .jar libraries of our project, it is enough to make clean & build our project

Note: If for some reason the repository is not updated, disable the antivirus or verify if you have a proxy configuration and disable it.



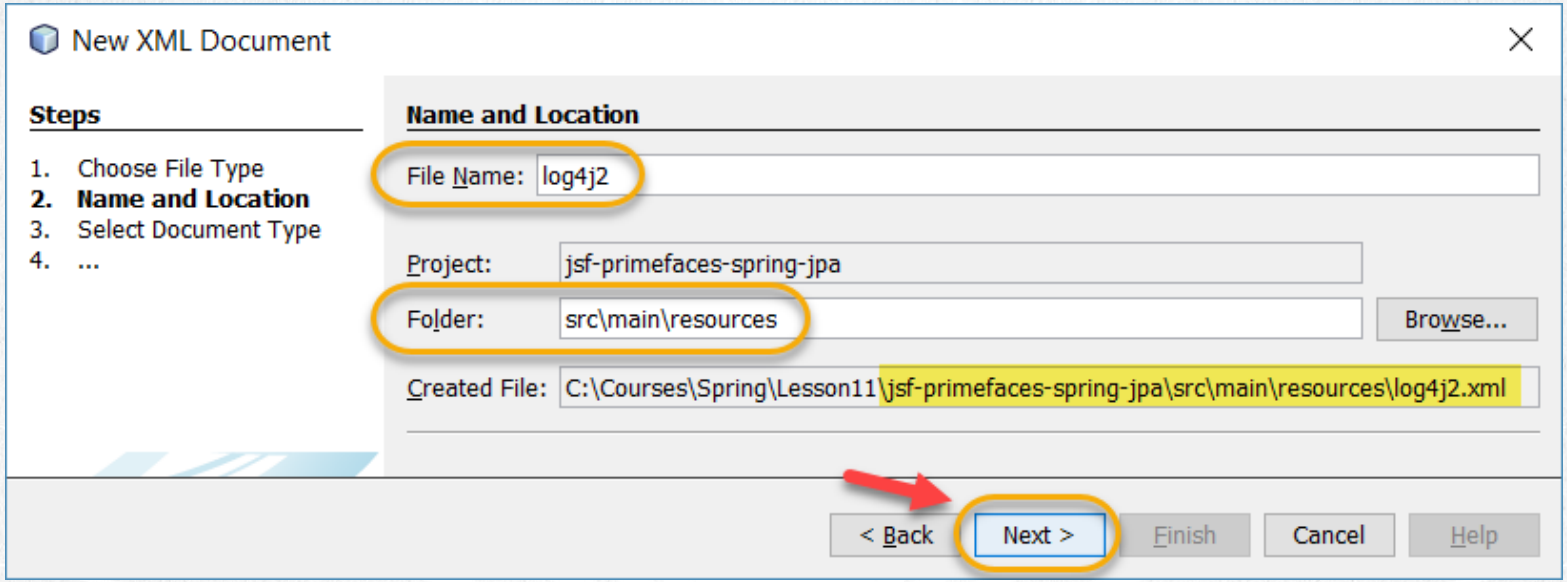
4. CREATE AN XML FILE

- We create the log4j2.xml file:



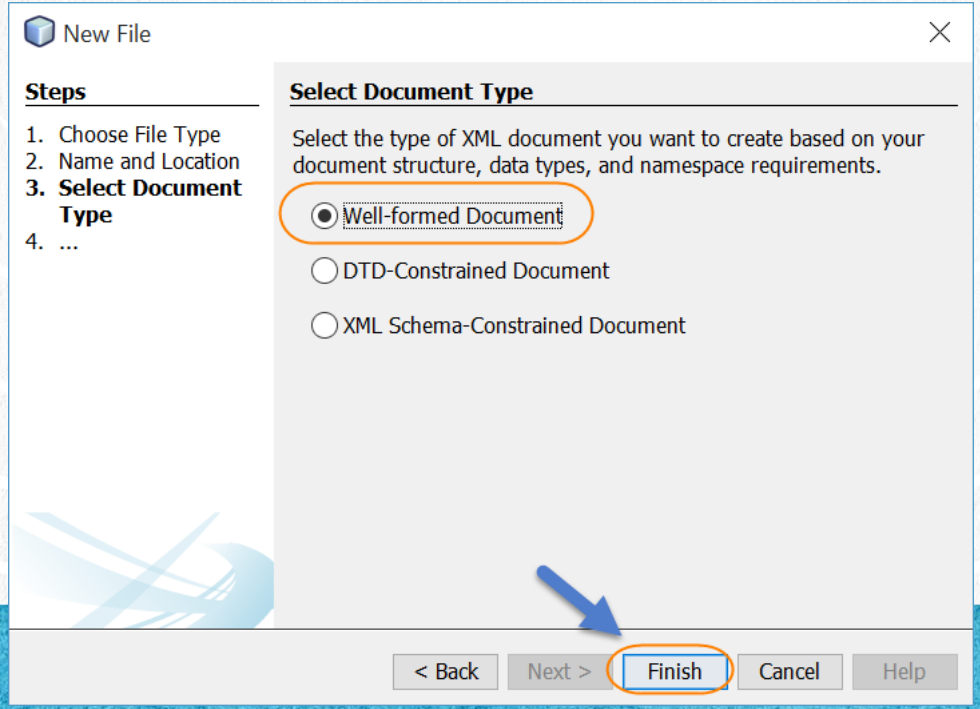
4. CREATE AN XML FILE

- We create the log4j2.xml file:



4. CREATE AN XML FILE

- We create the log4j2.xml file. In this step we select any option, it is not important since we are going to overwrite the file:



5. MODIFY THE FILE

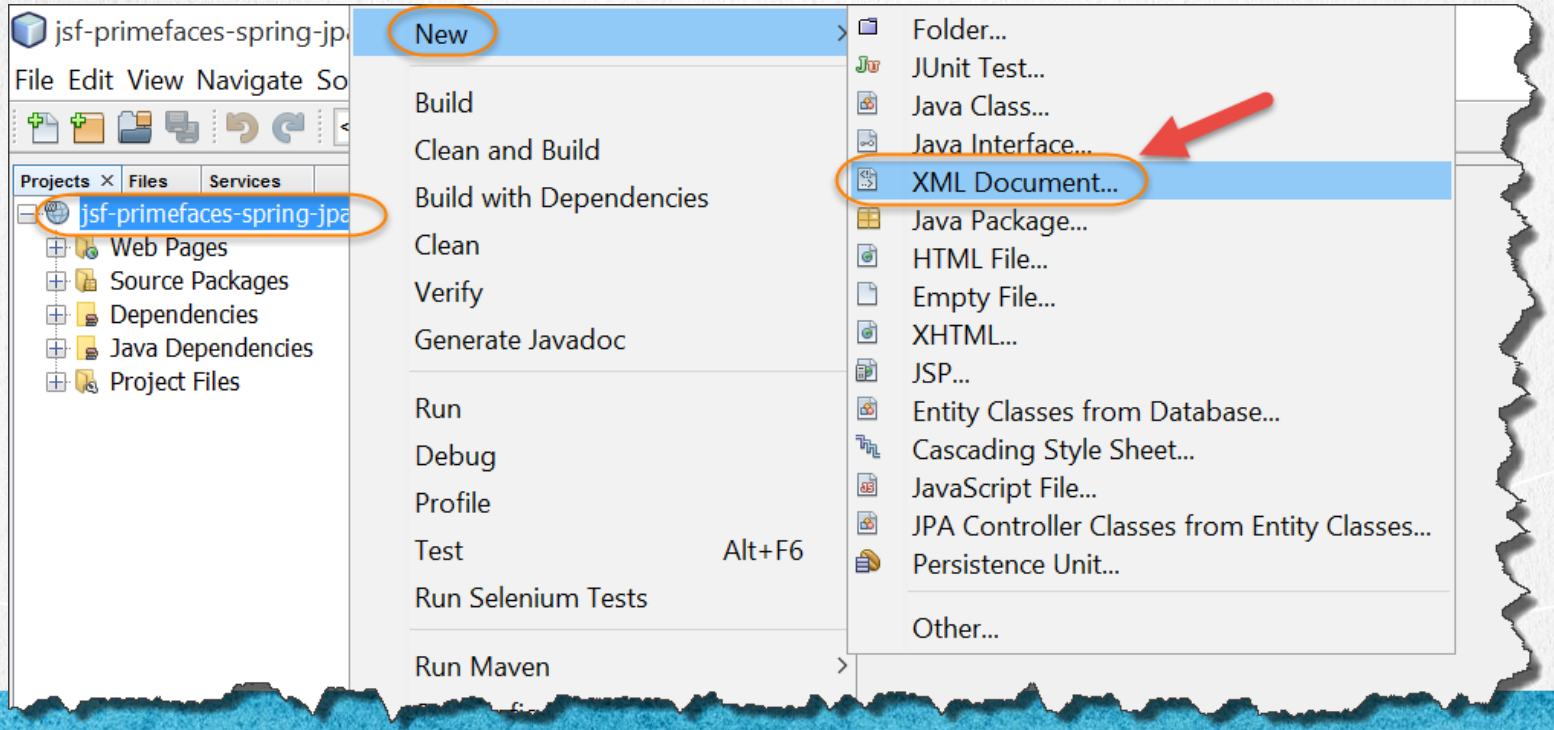
log4j2.xml:

Click to download

```
<?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>
        <logger name="org.springframework.jdbc.core" level="info" additivity="false">
            <appender-ref ref="Console" />
        </logger>
         <logger name="org.springframework.transaction" level="info" additivity="false">
            <appender-ref ref="Console" />
        </logger>
        <Root level="info">
            <AppenderRef ref="Console" />
        </Root>
    </Loggers>
</Configuration>
```

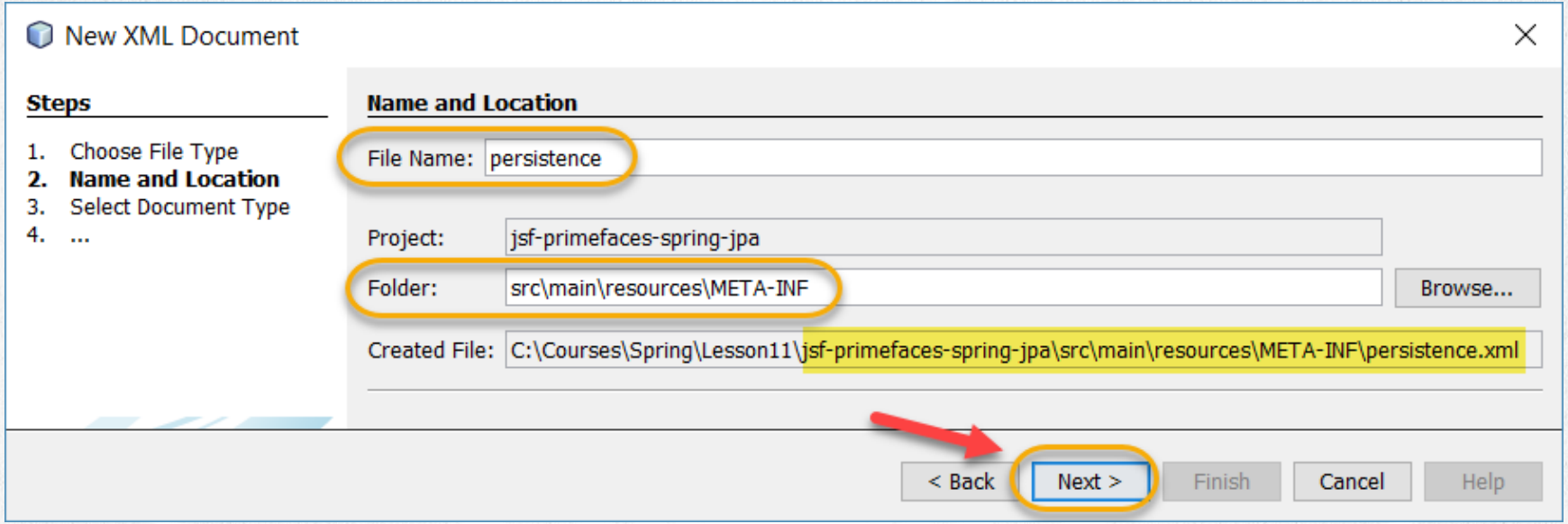
6. CREATE AN XML FILE

- We create the persistence.xml file:



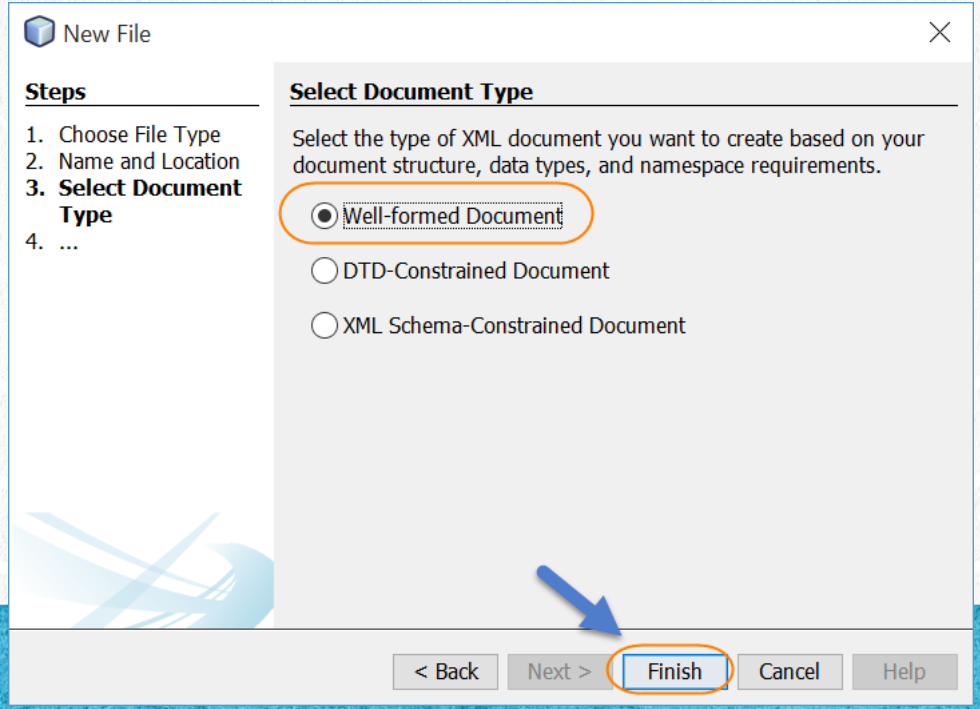
6. CREATE AN XML FILE

- We create the persistence.xml file:



6. CREATE AN XML FILE

- We create the persistence.xml file. In this step we select any option, it is not important since we are going to overwrite the file:



7. MODIFY THE FILE

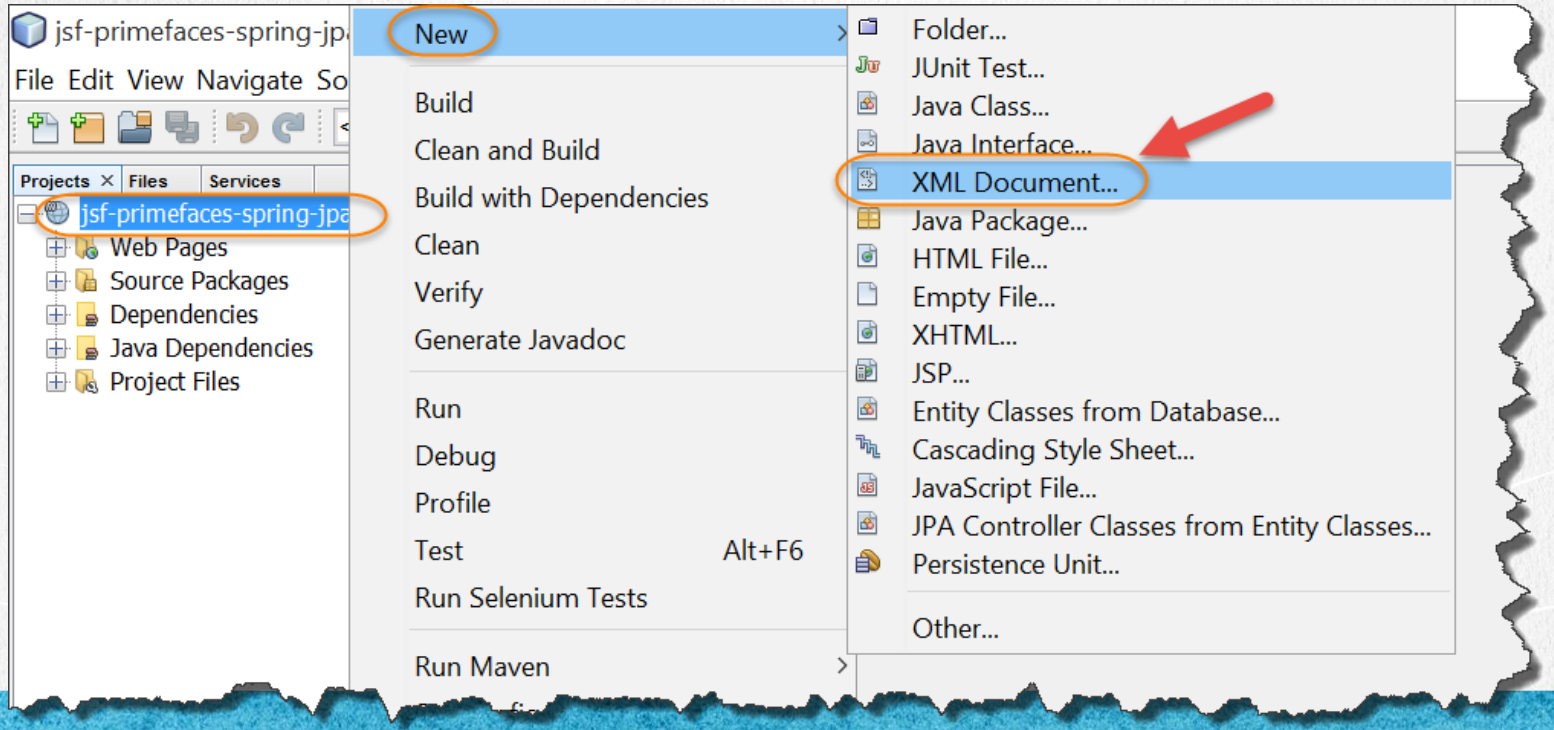
persistence.xml:

Click to download

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence 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"
             version="2.2">
    <persistence-unit name="PersistenceUnit" transaction-type="JTA">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <jta-data-source>jdbc/PersonDb</jta-data-source>
        <properties>
            <property name="eclipselink.logging.level" value="FINE"/>
            <property name="eclipselink.logging.parameters" value="true"/>
            <property name="eclipselink.logging.timestamp" value="false"/>
        </properties>
    </persistence-unit>
</persistence>
```

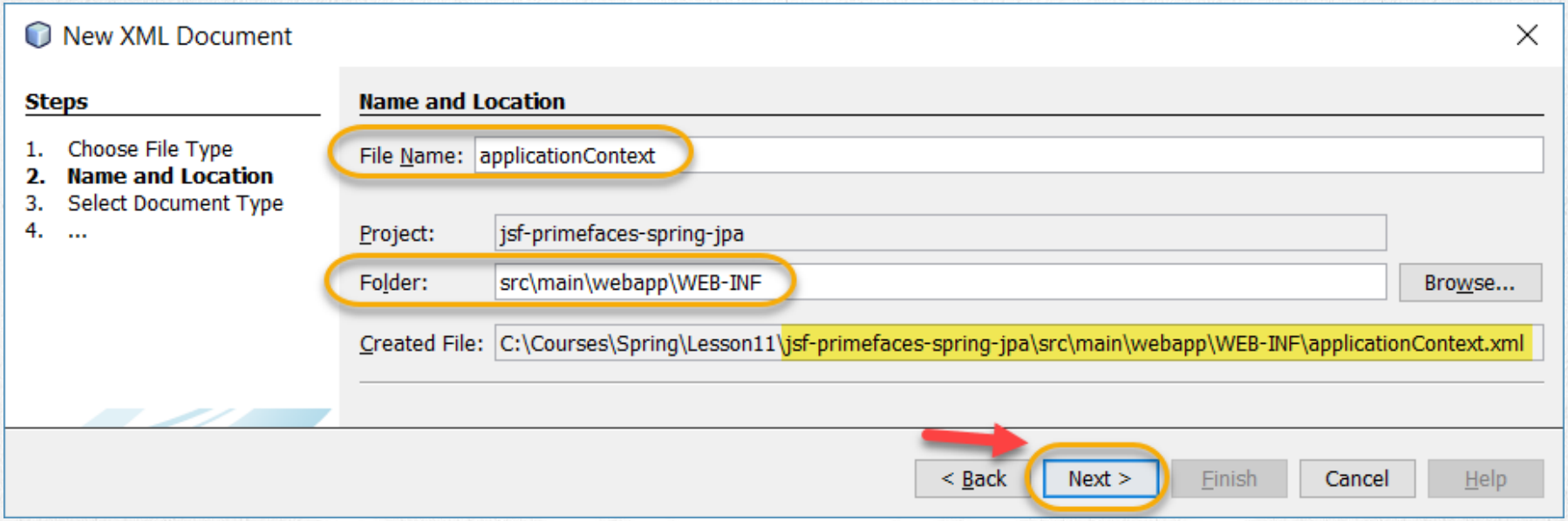
8. CREATE AN XML FILE

- We create the applicationContext.xml file:



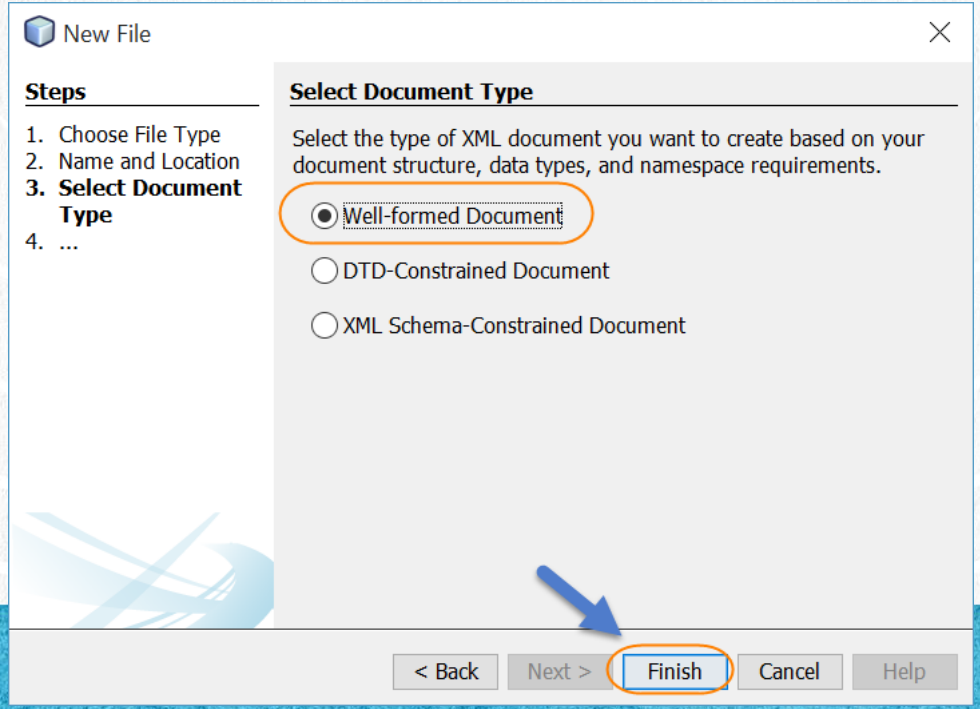
8. CREATE AN XML FILE

- We create the applicationContext.xml file:



8. CREATE AN XML FILE

- We create the applicationContext.xml file. In this step we select any option, it is not important since we are going to overwrite the file:



9. MODIFY THE CODE

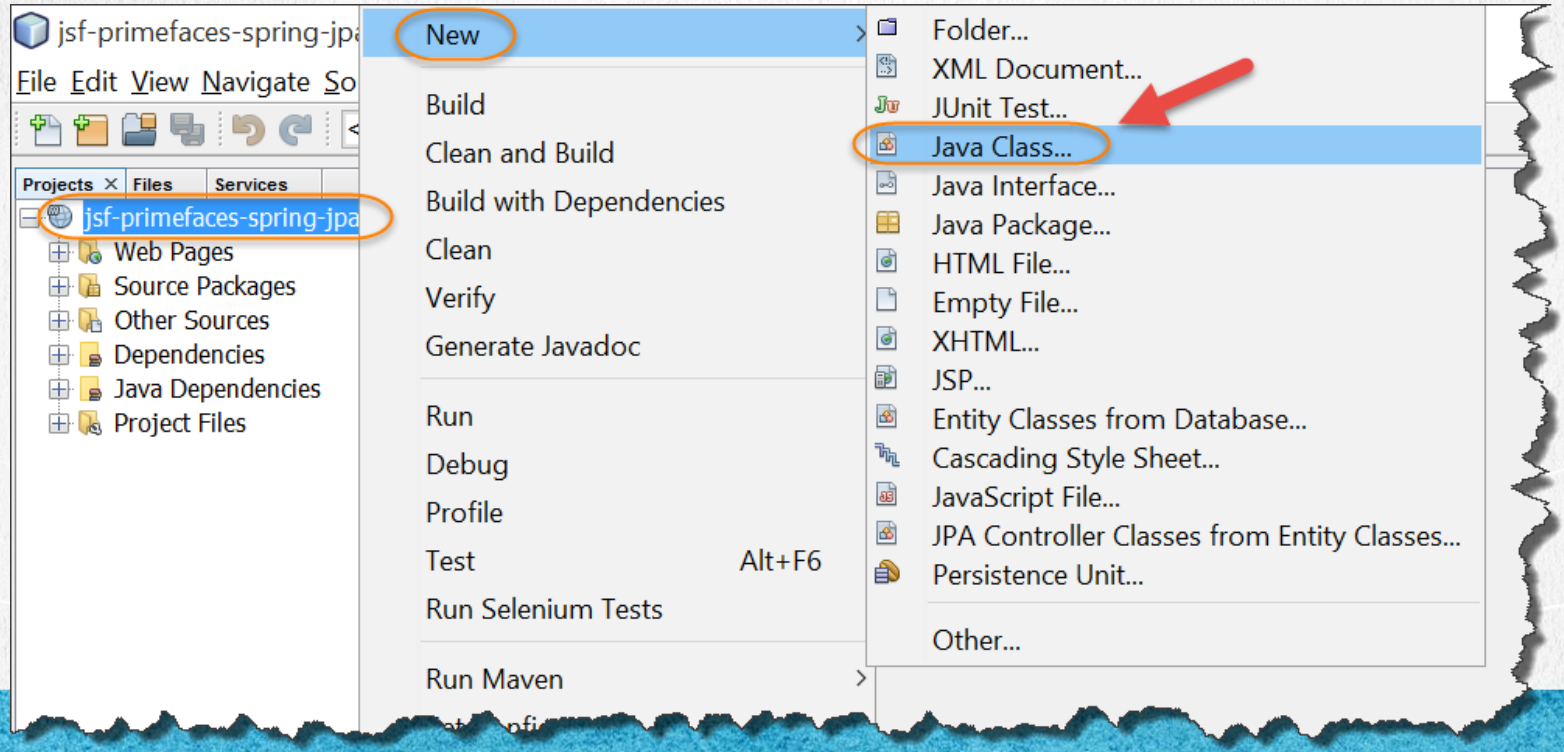
applicationContext.xml:

Click to download

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xsi:schemaLocation="
                http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans.xsd
                        http://www.springframework.org/schema/context
                http://www.springframework.org/schema/context/spring-context.xsd
                http://www.springframework.org/schema/jee
                http://www.springframework.org/schema/jee/spring-jee.xsd
                        http://www.springframework.org/schema/tx
                http://www.springframework.org/schema/tx/spring-tx.xsd">

    <context:component-scan base-package="data" />
    <context:component-scan base-package="service" />

   <jee:jndi-lookup id="entityManagerFactory" jndi-name="persistence/PersistenceUnit" />

    <!-- Detect @Transactional -->
    <tx:annotation-driven transaction-manager="transactionManager" />

</beans>
```

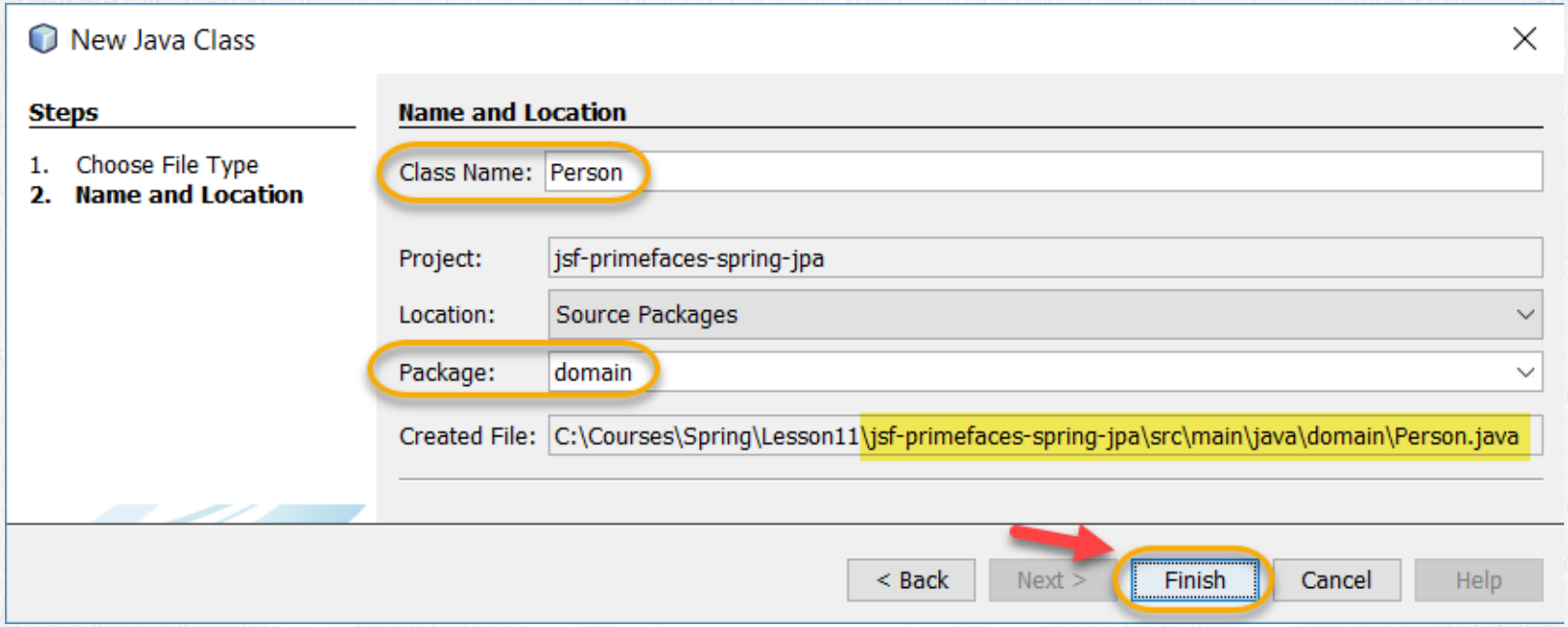
10. CREATE A NEW CLASS

Next we create the Person.java class:



10. CREATE A NEW CLASS

Next we create the Person.java class:



11. MODIFY THE CODE

Person.java:

Click to download

```
package domain;

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

@Entity
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;
    }

    public int getIdPerson() {
        return idPerson;
    }
```

11. MODIFY THE CODE

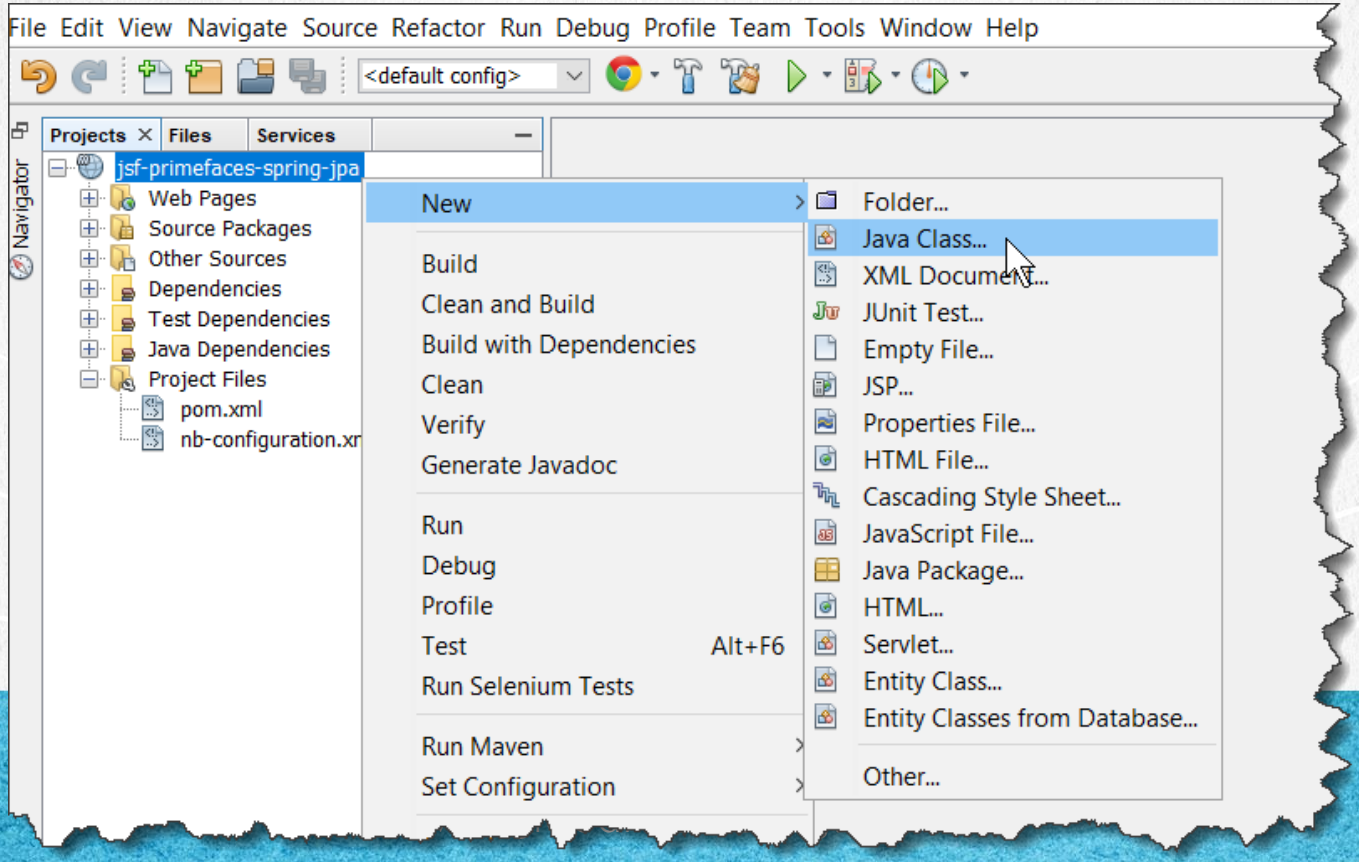
Person.java:

Click to download

```
    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 + '}';
    }
}
```

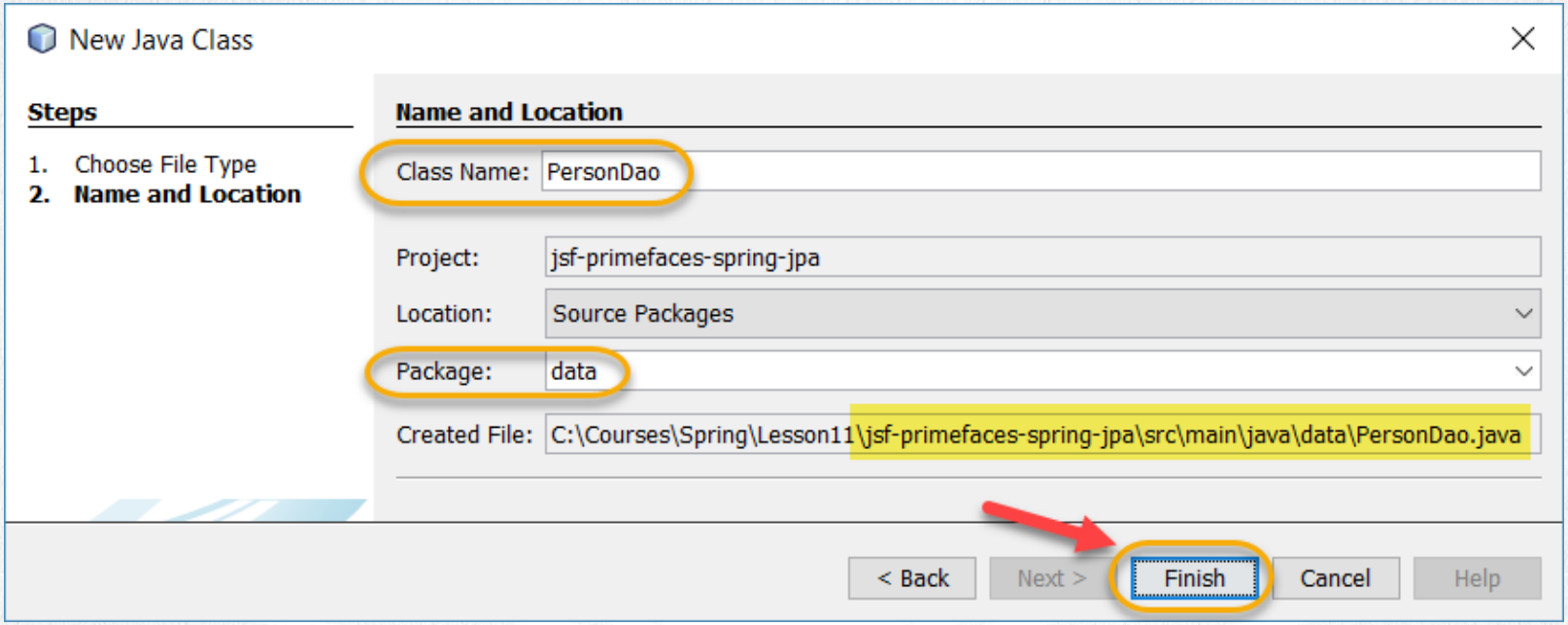
12. CREATE A NEW CLASS

Next we create the PersonDao.java interface:



12. CREATE A NEW CLASS

Next we create the PersonDao.java interface:



13. MODIFY THE FILE

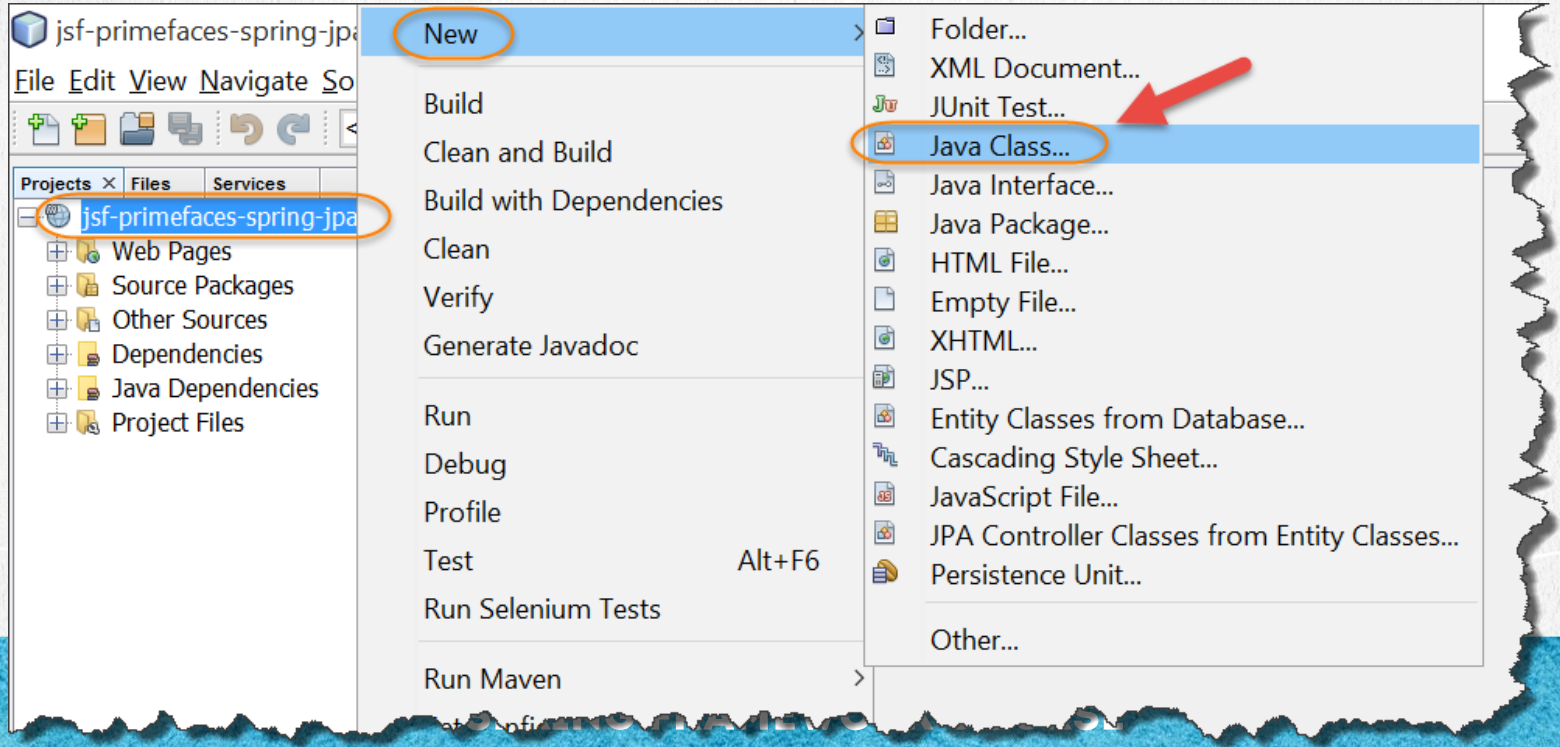
PersonDao.java:

Click to download

```
package data;

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

public interface PersonDao {

    void insertPerson(Person person);

    void updatePerson(Person person);

    void deletePerson(Person person);

    Person findPersonById(int idPerson);

    List<Person> findAllPeople();

    long countPeople();
}
```

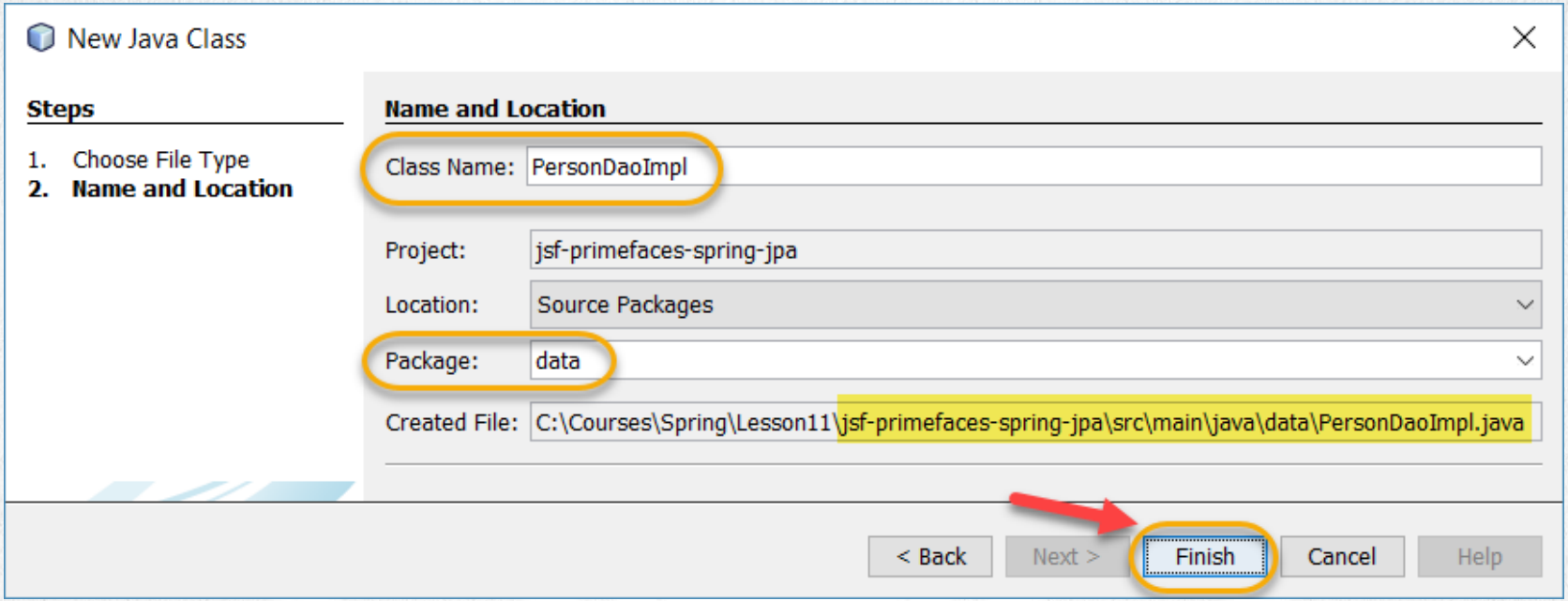
14. CREATE A NEW CLASS

We create the PersonDaoImpl.java class.



14. CREATE A NEW CLASS

We create the PersonDaoImpl.java class:



15. MODIFY THE CODE

PersonDaoImpl.java:

Click to download

```
package data;

import domain.Person;
import java.util.List;
import javax.persistence.*;
import org.springframework.stereotype.Repository;
import org.apache.logging.log4j.*;
import javax.persistence.Query;

@Repository
public class PersonDaoImpl implements PersonDao {

    Logger log = LogManager.getRootLogger();

    @PersistenceContext
    private EntityManager em;

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

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

15. MODIFY THE CODE

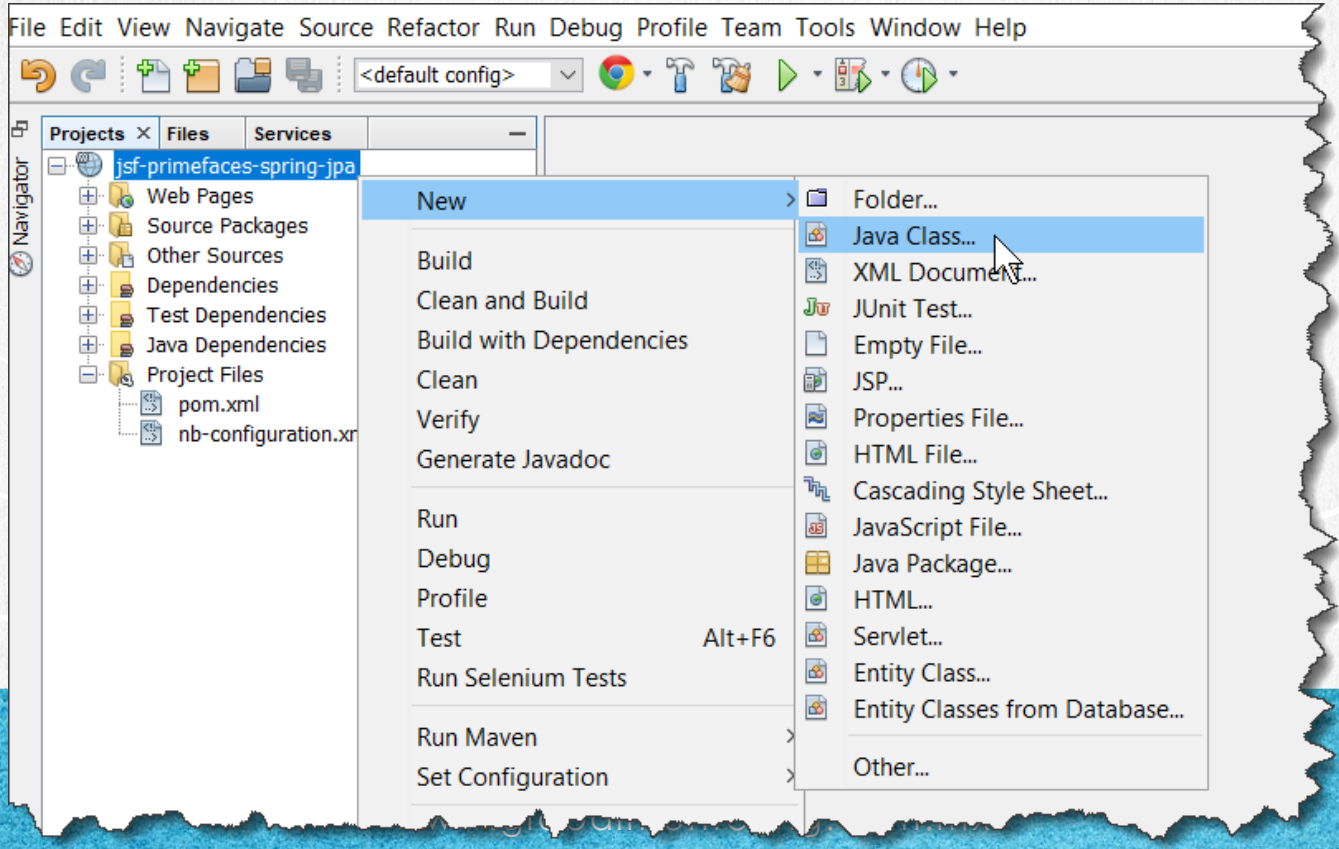
PersonDaoImpl.java:

Click to download

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

    @Override
    public Person findPersonById(int idPerson) {
        return em.find(Person.class, idPerson);
    }

    @Override
    public List<Person> findAllPeople() {
        String jpql = "SELECT p FROM Person p";
        Query query = em.createQuery(jpql);
        return query.getResultList();
    }

    @Override
    public long countPeople() {
        String consulta = "select count(p) from Person p";
        Query q = em.createQuery(consulta);
        return (long) q.getSingleResult();
    }
}
```

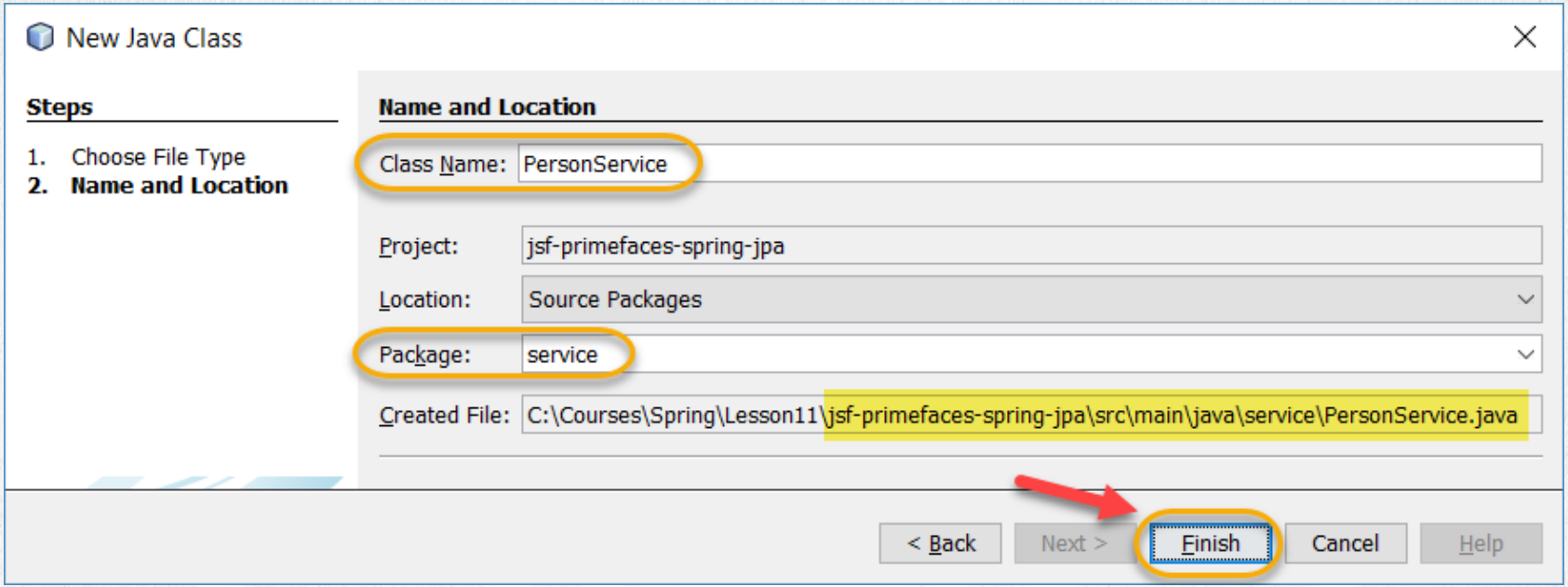
16. CREATE A NEW CLASS

Next we create the PersonService.java interface:



16. CREATE A NEW CLASS

Next we create the PersonService.java interface:



17. MODIFY THE FILE

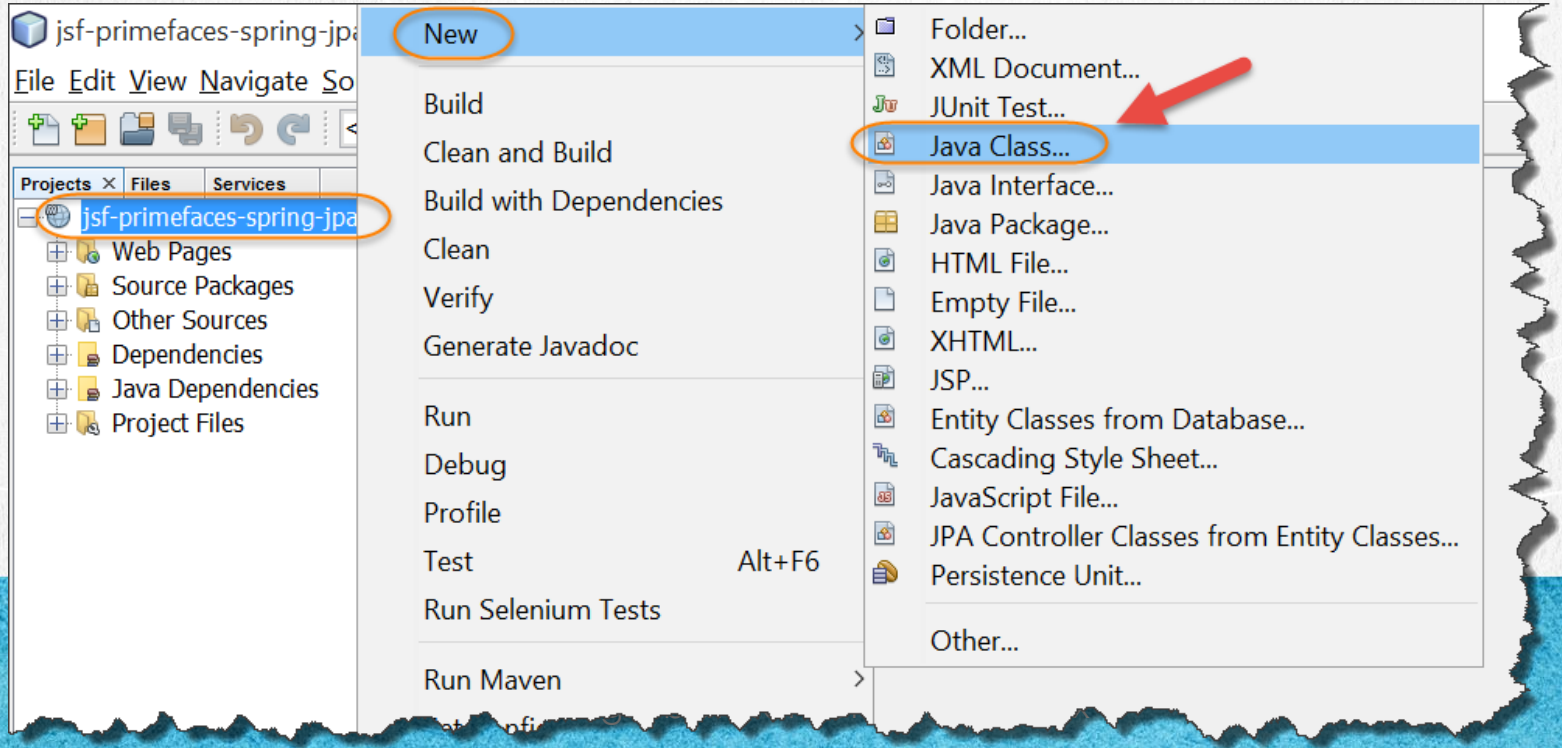
PersonService.java:

Click to download

```
package service;

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

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);
}
```

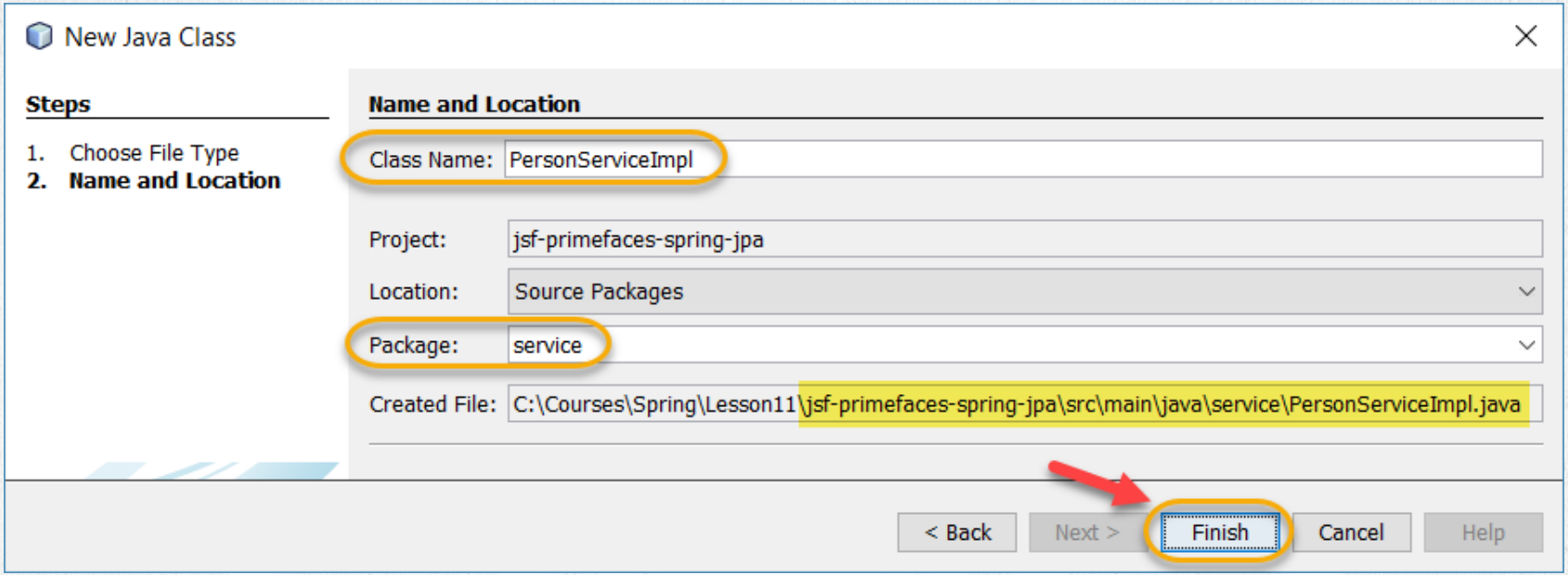
18. CREATE A NEW CLASS

We create the PersonServiceImpl.java class. Due to we will use a Java Enterprise server, we will use the Java EE annotations to do the dependency injection, that is, we will use the annotation of @Inject instead of @Autowire. The same for the @Transactional annotation, we will use the Java EE annotation and not the Spring annotation:



18. CREATE A NEW CLASS

We create the PersonServiceImpl.java class:



19. MODIFY THE FILE

PersonServiceImpl.java:

Click to download

```
package service;

import data.PersonDao;
import domain.Person;
import java.util.List;
import javax.inject.Inject;
import javax.transaction.Transactional;
import org.springframework.stereotype.Service;

@Service("personService")
@Transactional
public class PersonServiceImpl implements PersonService {

    @Inject
    private PersonDao personDao;

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

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

19. MODIFY THE FILE

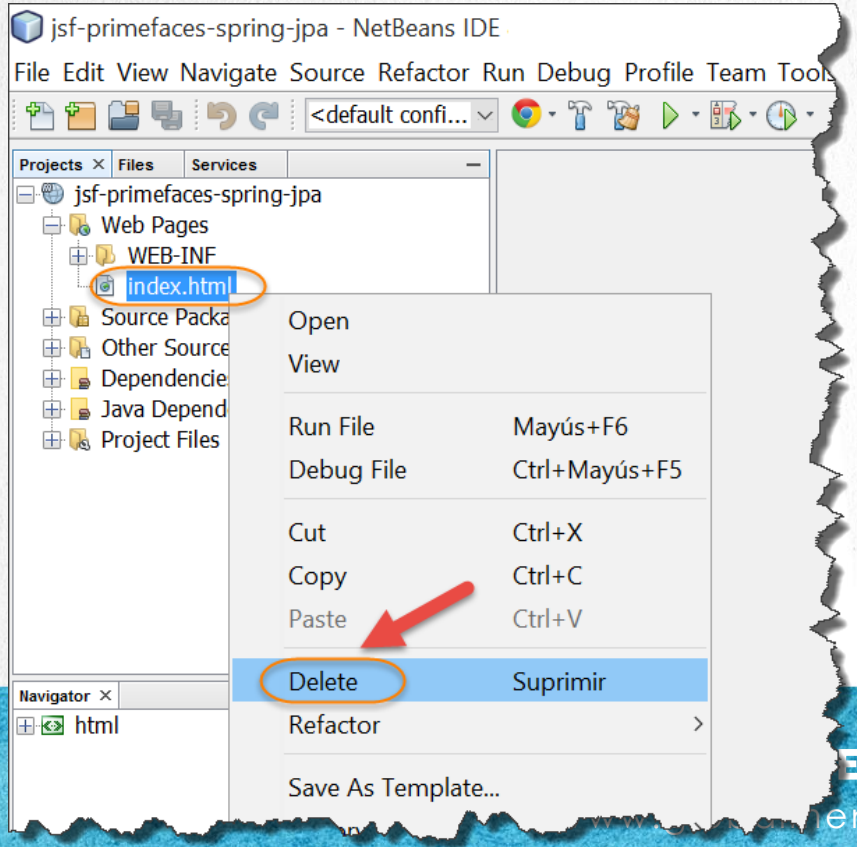
PersonServiceImpl.java:

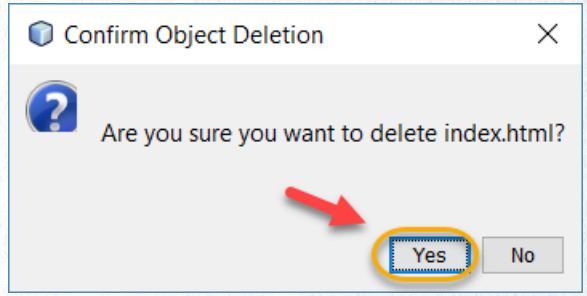
Click to download

```
    @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);
    }
}
```

20. REMOVE THE INDEX.HTML FILE

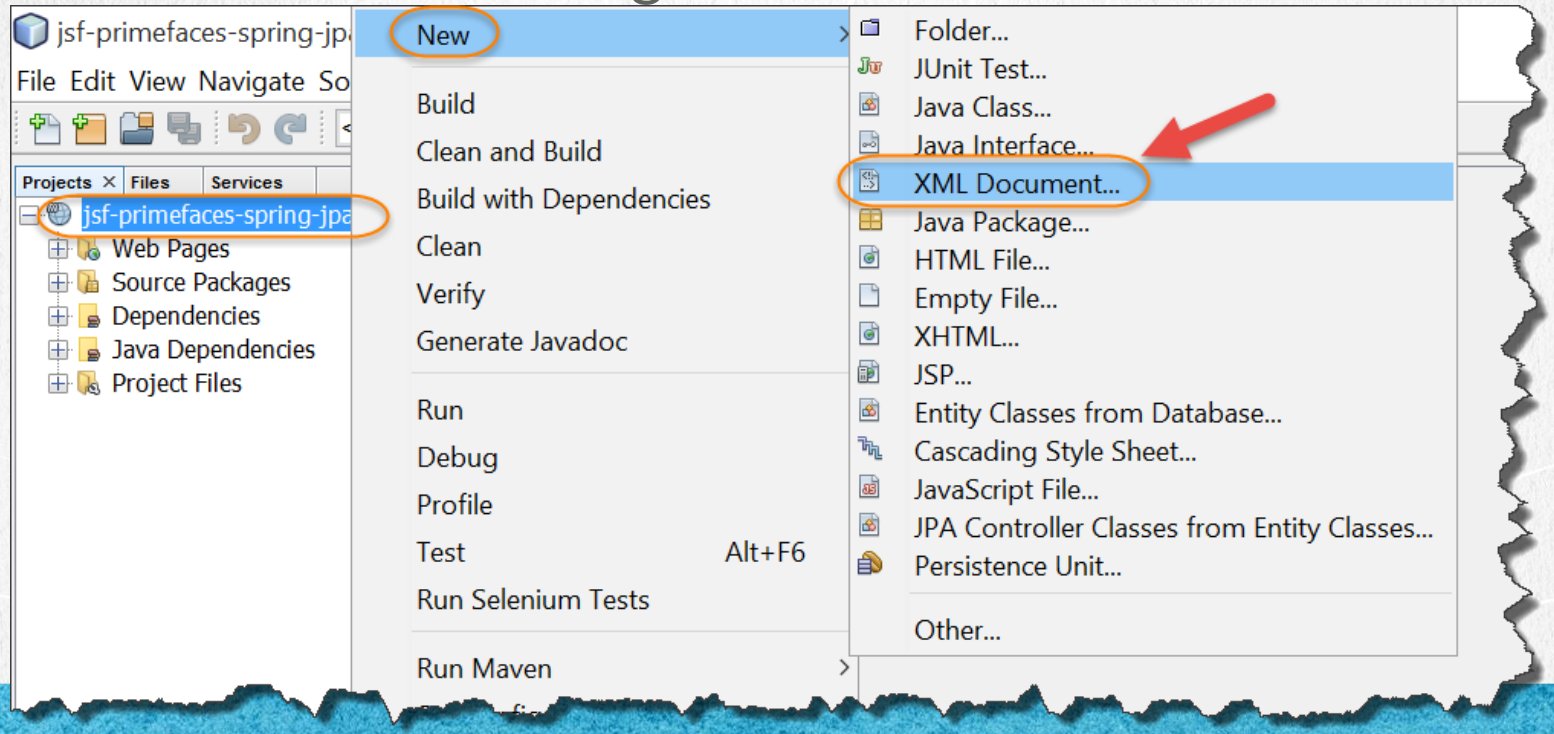
Remove the index.html file if it exists:





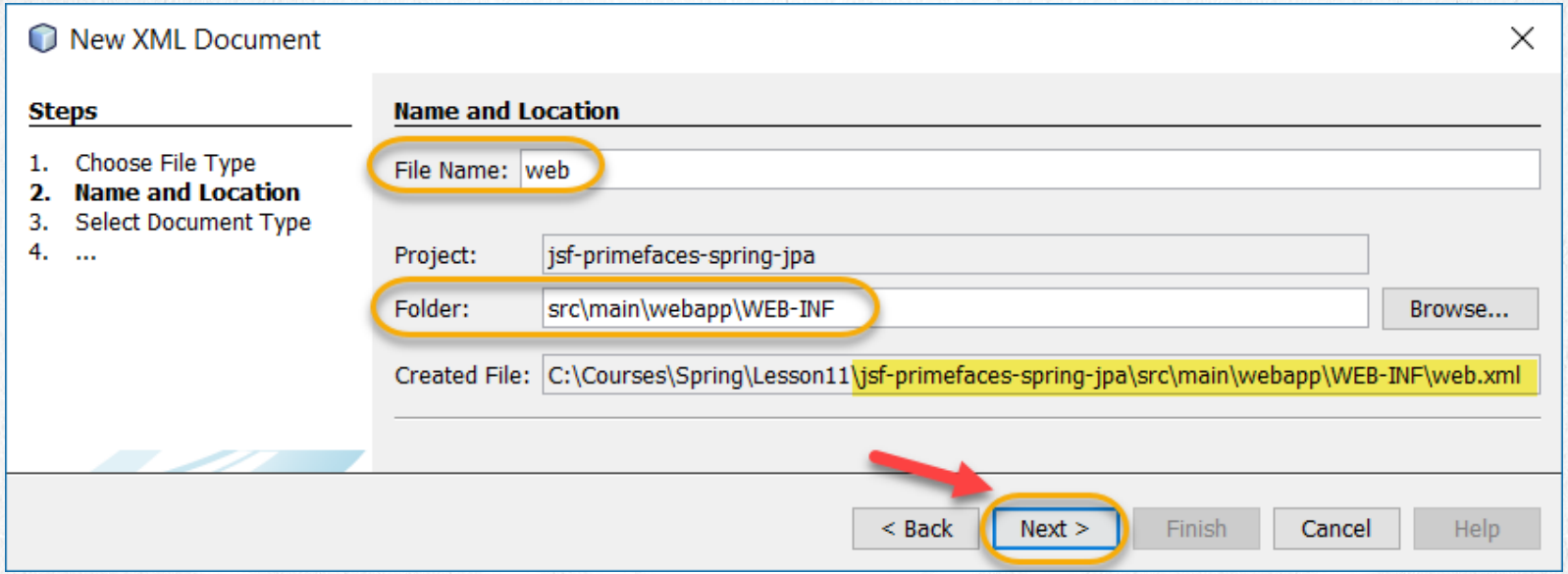
21. ADD THE WEB.XML FILE

We add the web.xml configuration file:



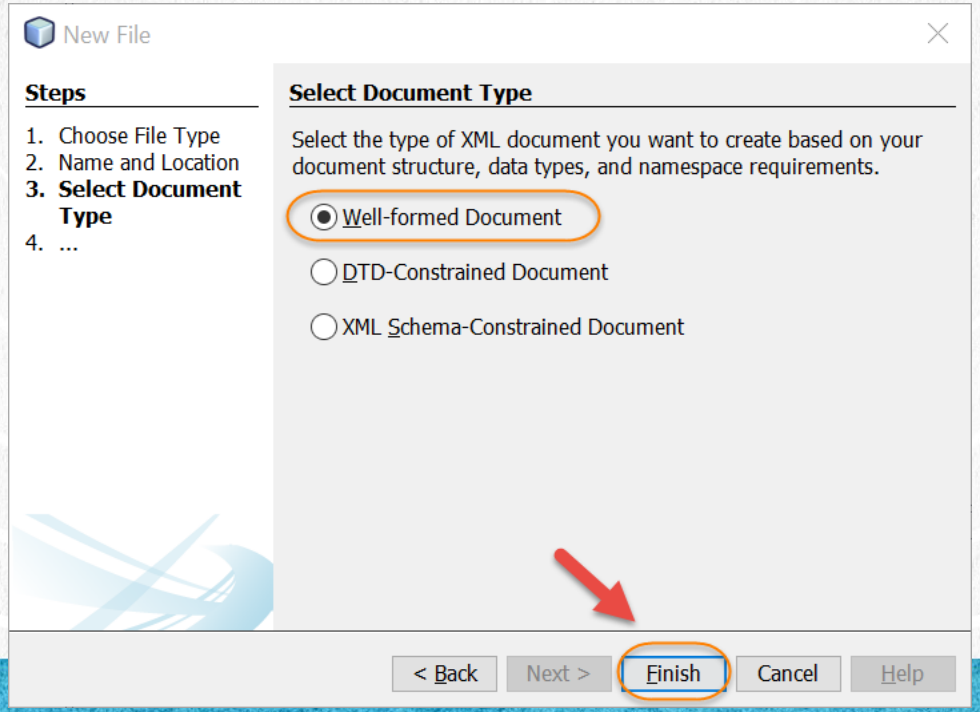
21. ADD THE WEB.XML FILE

We add the web.xml configuration file:



21. ADD THE WEB.XML FILE

We add the web.xml configuration file:



22. MODIFY THE CODE

web.xml:

Click to download

```
<?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 webapp_4_0.xsd"
         version="4.0">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.FACELETS_REFRESH_PERIOD</param-name>
        <param-value>0</param-value>
    </context-param>
    <context-param>
        <param-name>primefaces.THEME</param-name>
        <param-value>cupertino</param-value>
    </context-param>
    <error-page>
        <exception-type>javax.faces.application.ViewExpiredException</exception-type>
        <location>/faces/index.xhtml</location>
    </error-page>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

22. MODIFY THE CODE

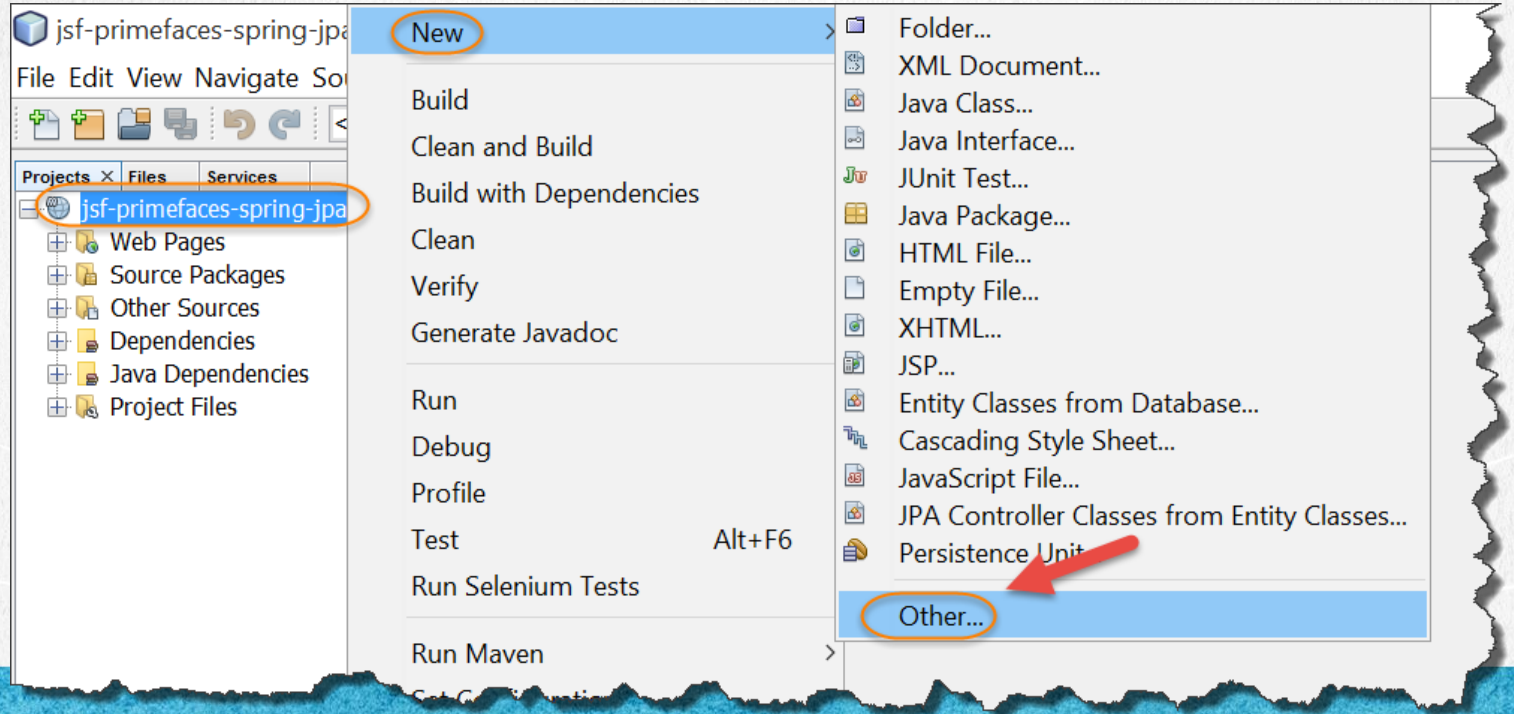
web.xml:

Click to download

```
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <listener>
        <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
    </listener>
    <persistence-unit-ref>
        <persistence-unit-ref-name>persistence/PersistenceUnit</persistence-unit-ref-name>
        <persistence-unit-name>PersistenceUnit</persistence-unit-name>
    </persistence-unit-ref>
</web-app>
```

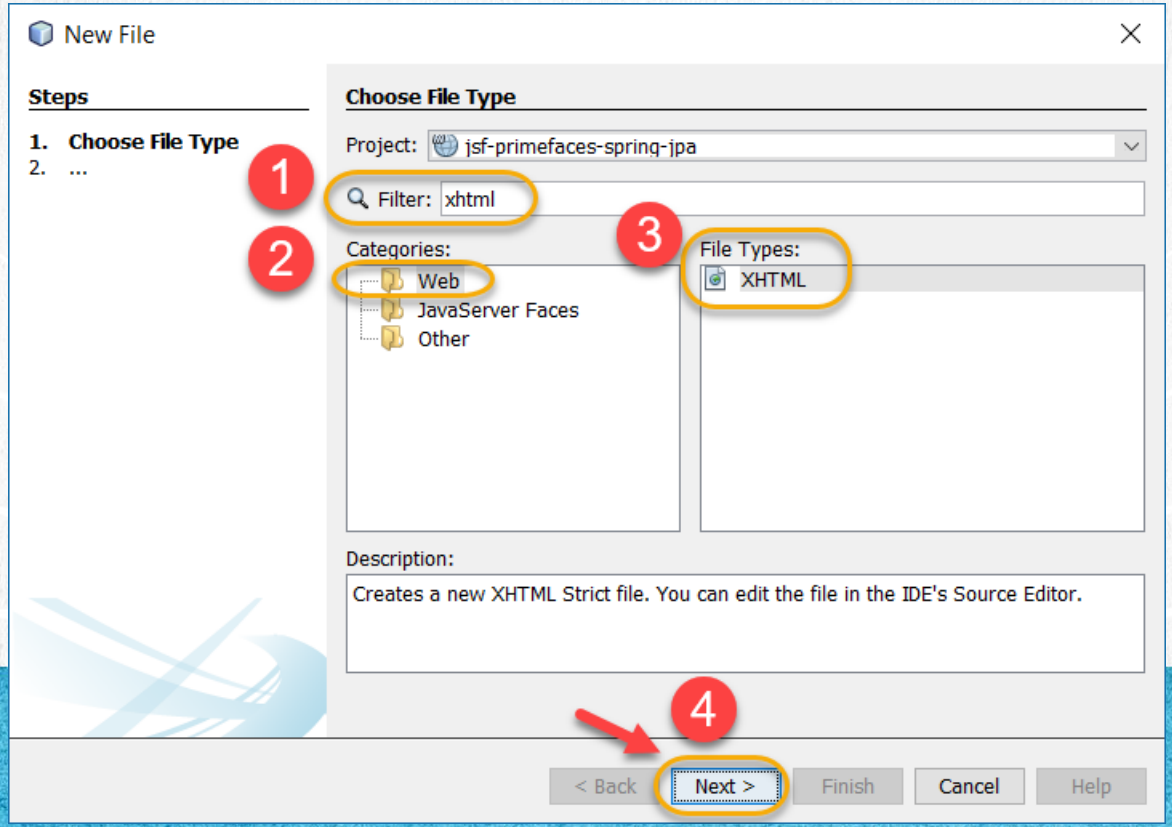
23. CREATE AN XHTML FILE

We create the index.xhtml file:



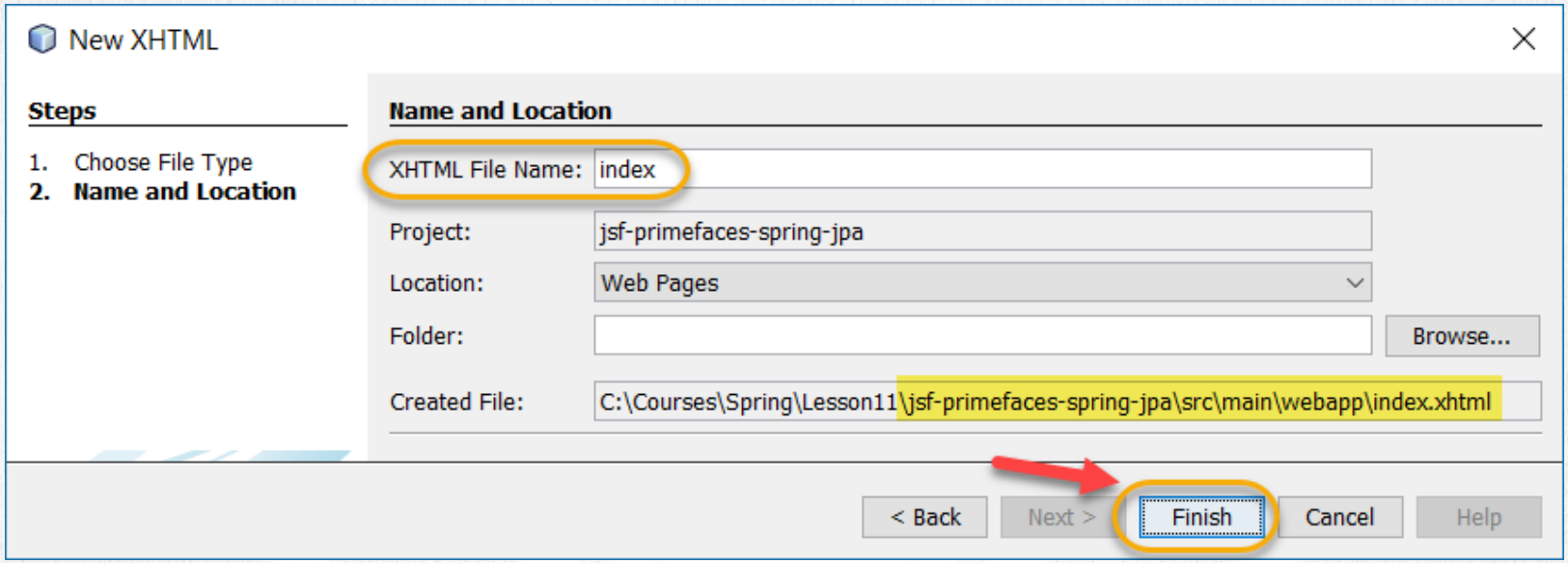
23. CREATE AN XHTML FILE

We create the index.xhtml file:



23. CREATE AN XHTML FILE

We create the index.xhtml file:



24. MODIFY THE FILE

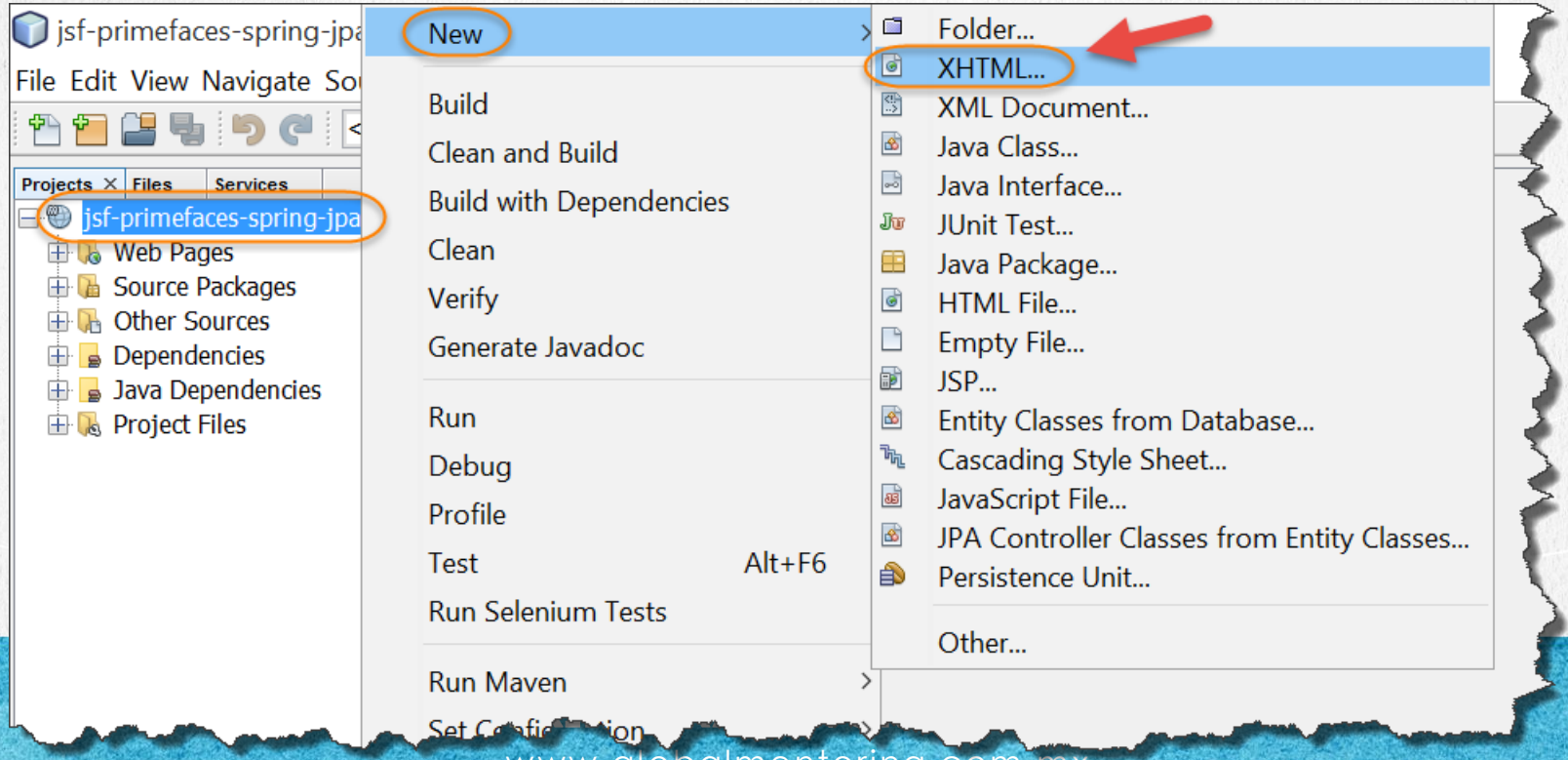
index.xhtml:

Click to download

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
        <title>Admin People</title>
    </h:head>
    <h:body>
        <h2>Admin People</h2>
        <h:form>
            <p:messages />
            <h:panelGrid columns="1">
                <h:commandButton value="List People" action="listPeople" />
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

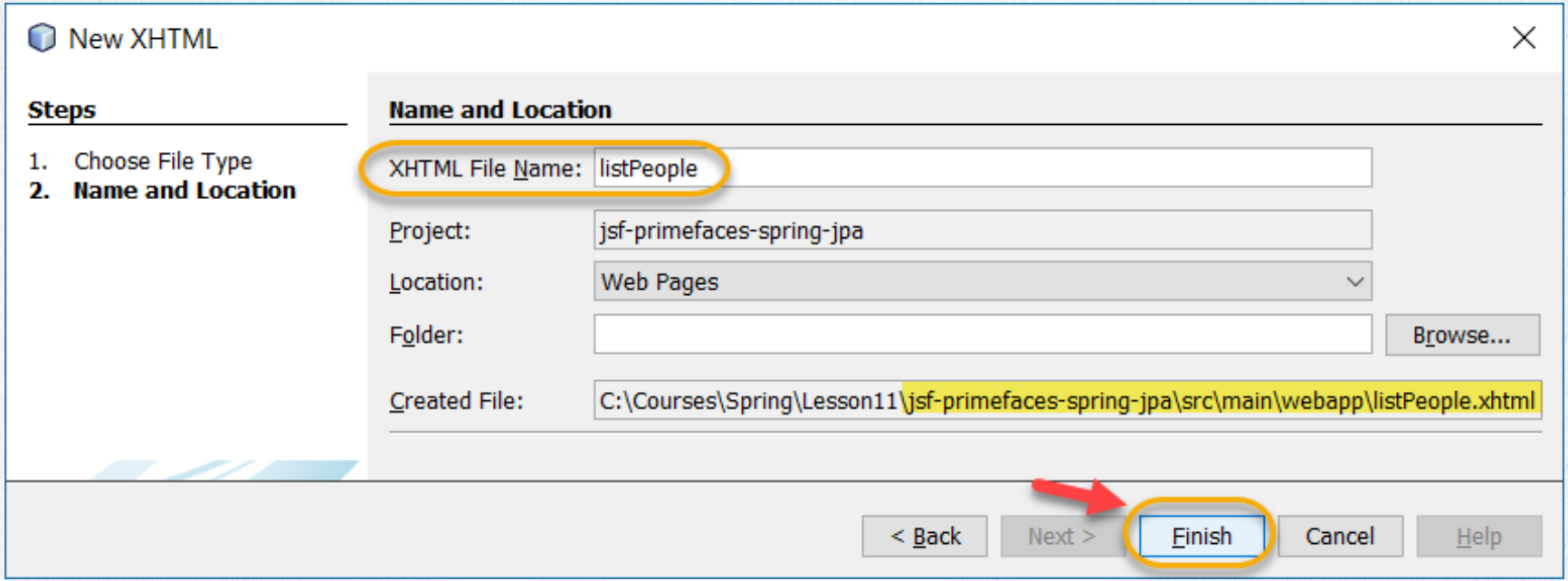
25. CREATE AN XHTML FILE

We create the file listPeople.xhtml:



25. CREATE AN XHTML FILE

We create the file listPeople.xhtml:



26. MODIFY THE CODE

listPeople.xhtml:

Click to download

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">

    <h:head>
        <title>List of People</title>
    </h:head>
    <h:body>

        <p:ajaxStatus style="width:16px;height:16px;" id="ajaxStatusPanel">
            <f:facet name="start">
                <h:graphicImage library="images" name="ajaxloading.gif" />
            </f:facet>
            <f:facet name="complete">
                <h:outputText value="" />
            </f:facet>
        </p:ajaxStatus>

        <h:form id="PersonListForm">

            <p:growl id="messages" showDetail="true" />
            <p:messages />
```

26. MODIFY THE CODE

listPeople.xhtml:

Click to download

```
            <p:dataTable
                id="peopleDataTable"
                var="person"
                value="#{personBean.people}"
                editable="true"
                rowKey="#{person.idPerson}"
                selection="#{personBean.selectedPerson}"
                selectionMode="single">

                <f:facet name="header">
                    List of People
                </f:facet>

                <p:column headerText="Name">
                    <p:cellEditor>
                        <f:facet name="output">
                            <h:outputText value="#{person.name}" />
                        </f:facet>
                        <f:facet name="input">
                            <p:inputText value="#{person.name}" />
                        </f:facet>
                    </p:cellEditor>
                </p:column>
                <f:facet name="footer">
                    <p:commandButton value="Return" action="index" />
                    <p:spacer height="5px;"/>
                    <p:commandButton value="Add" actionListener="#{personBean.restartSelectedPerson}" update=":dialogForm"
oncomplete="PF('personDialog').show()" />
                </f:facet>
```

26. MODIFY THE CODE

listPeople.xhtml:

Click to download

```
                <p:column headerText="Options" style="width:50px">
                    <p:rowEditor />
                </p:column>

                <p:ajax event="rowEdit" listener="#{personBean.editListener}">
                </p:ajax>
            </p:dataTable>

            <p:contextMenu for="peopleDataTable">
                <p:menuitem value="Delete" update="PersonListForm:peopleDataTable" icon="ui-icon-close"
actionListener="#{personBean.deletePerson}"/>
            </p:contextMenu>
        </h:form>


        <p:dialog id="personDlg" widgetVar="personDialog" header="Add Person" modal="true" showEffect="fade"
hideEffect="fade" resizable="false">
            <h:form id="dialogForm">
                <p:outputPanel id="personDetail" style="text-align:center;">
                    <p:panelGrid id="display" columns="2" columnClasses="label,value" >
                        <h:outputText value="Name" />
                        <h:inputText value="#{personBean.selectedPerson.name}" />

                        <f:facet name="footer">
                            <p:commandButton value="Add" actionListener="#{personBean.addPerson}"
update="display,:PersonListForm:peopleDataTable"  oncomplete="PF('personDialog').hide();"/>
                        </f:facet>
```

26. MODIFY THE CODE

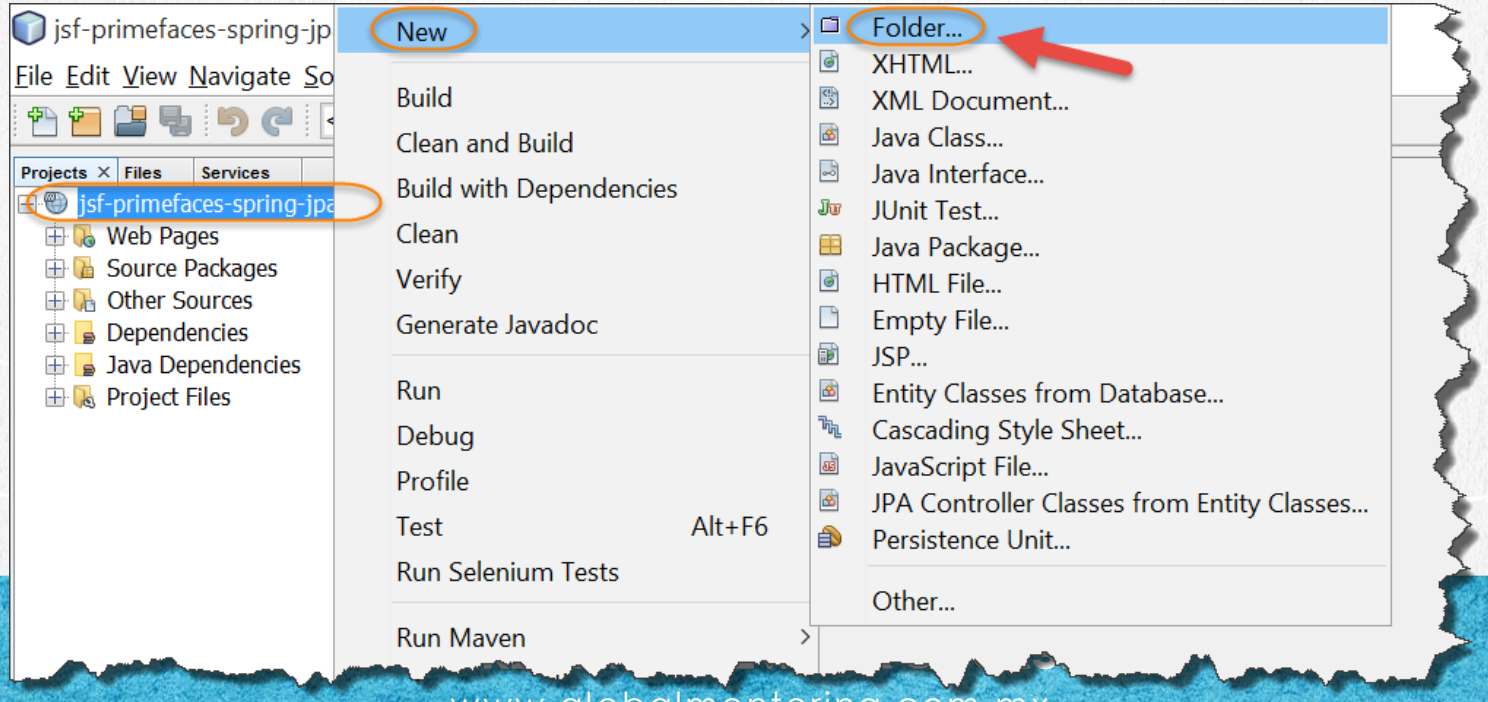
listPeople.xhtml:

Click to download

```
                    </p:panelGrid>
                </p:outputPanel>
            </h:form>
        </p:dialog>

    </h:body>

</html>
```

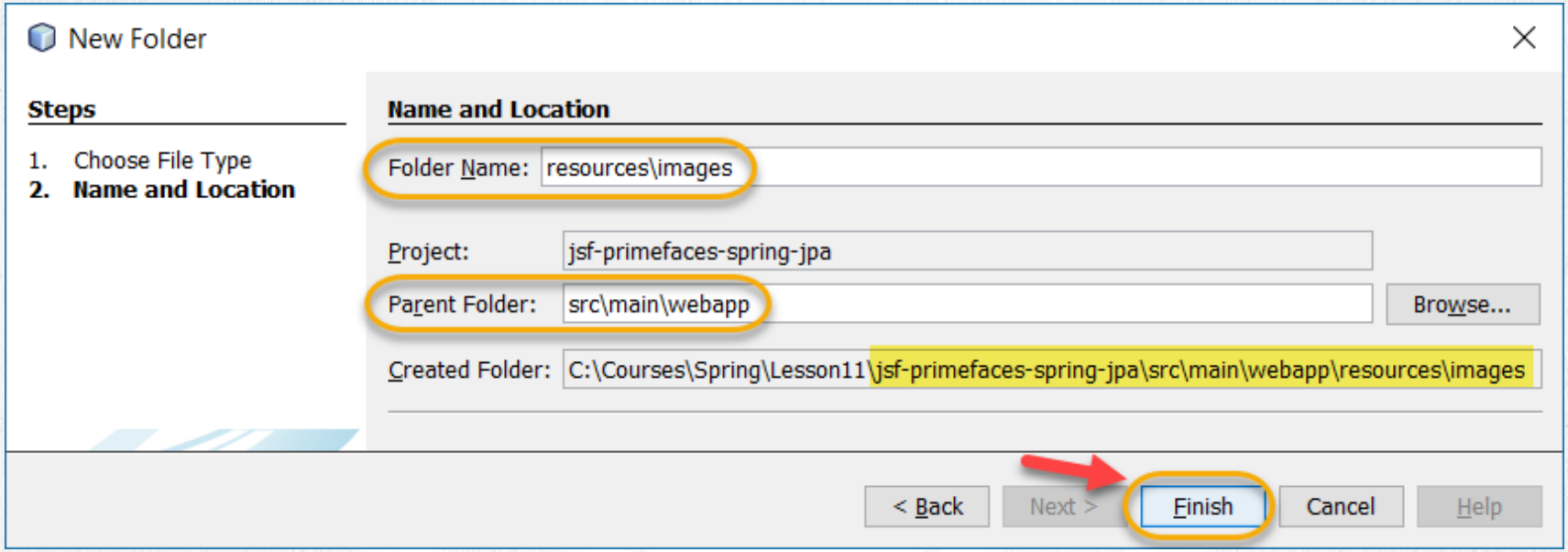
27. DOWNLOAD GIF IMAGE

We create a folder in the web part as follows: resources/images



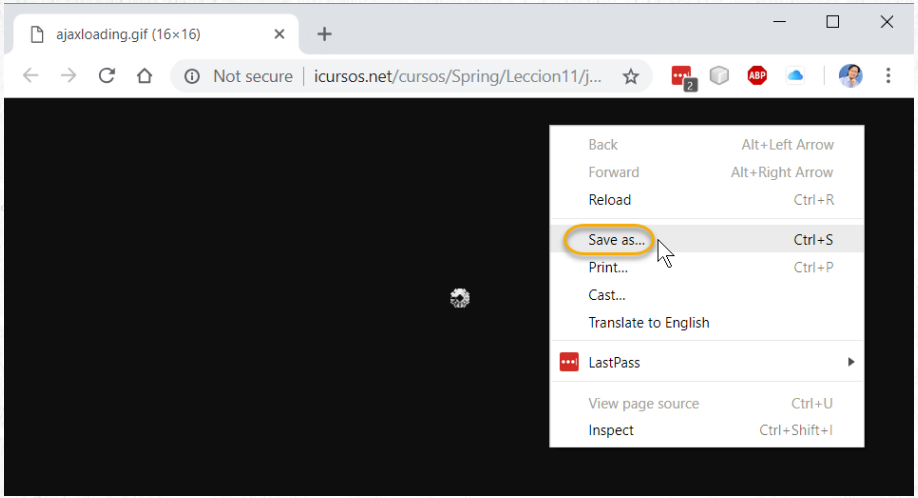
27. DOWNLOAD GIF IMAGE

We create a folder in the web part as follows: resources/images



27. DOWNLOAD GIF IMAGE

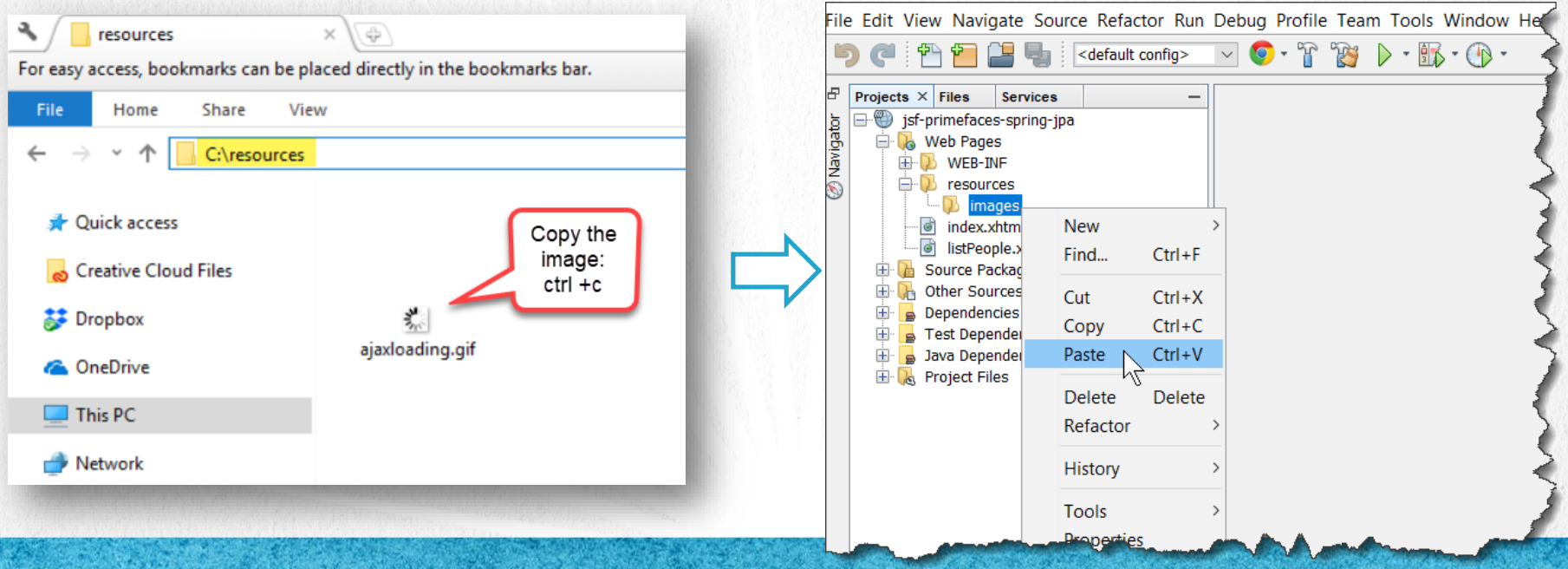
Download the ajaxloading.gif image:
http://icursos.net/cursos/Spring/Leccion11/jsf-primefaces-spring-jpa/ajaxloading.gif



We save the image in some path like: C:\resources

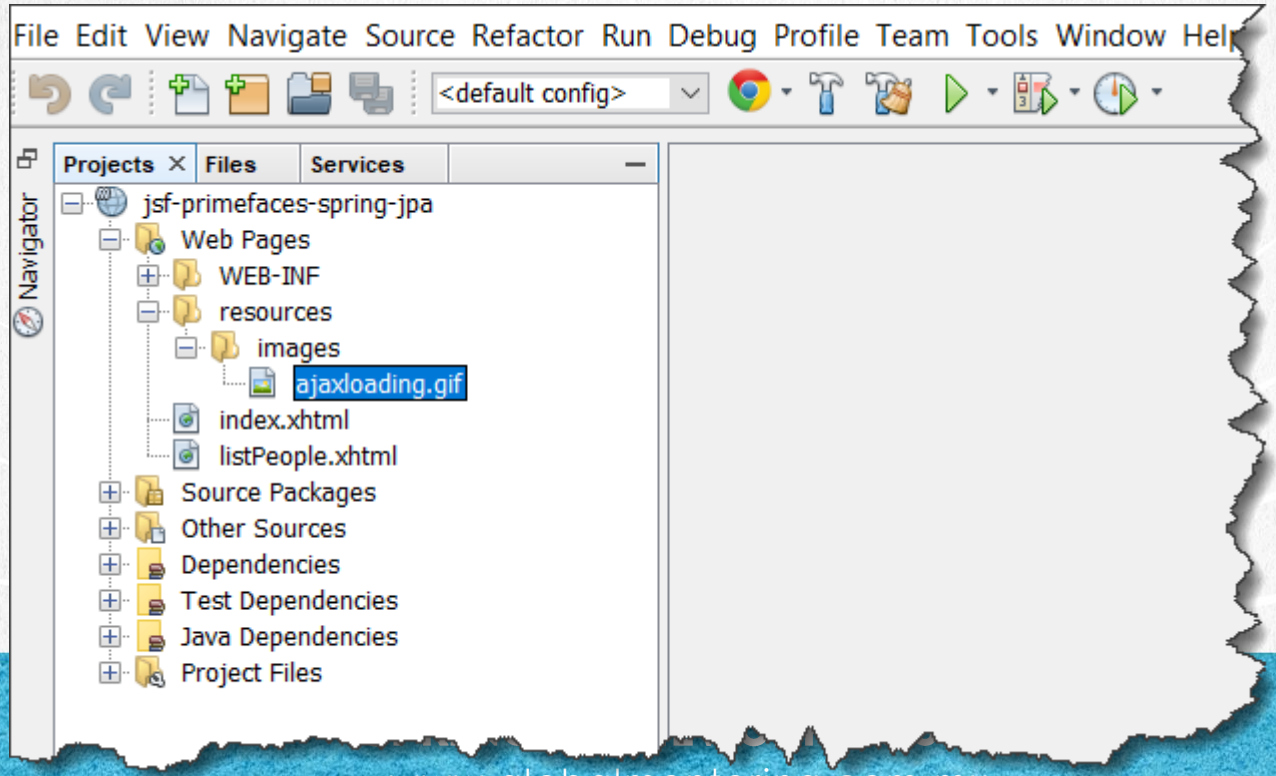
27. COPY AND PASTE THE GIF IMAGE

We copy the image and paste it into the project:



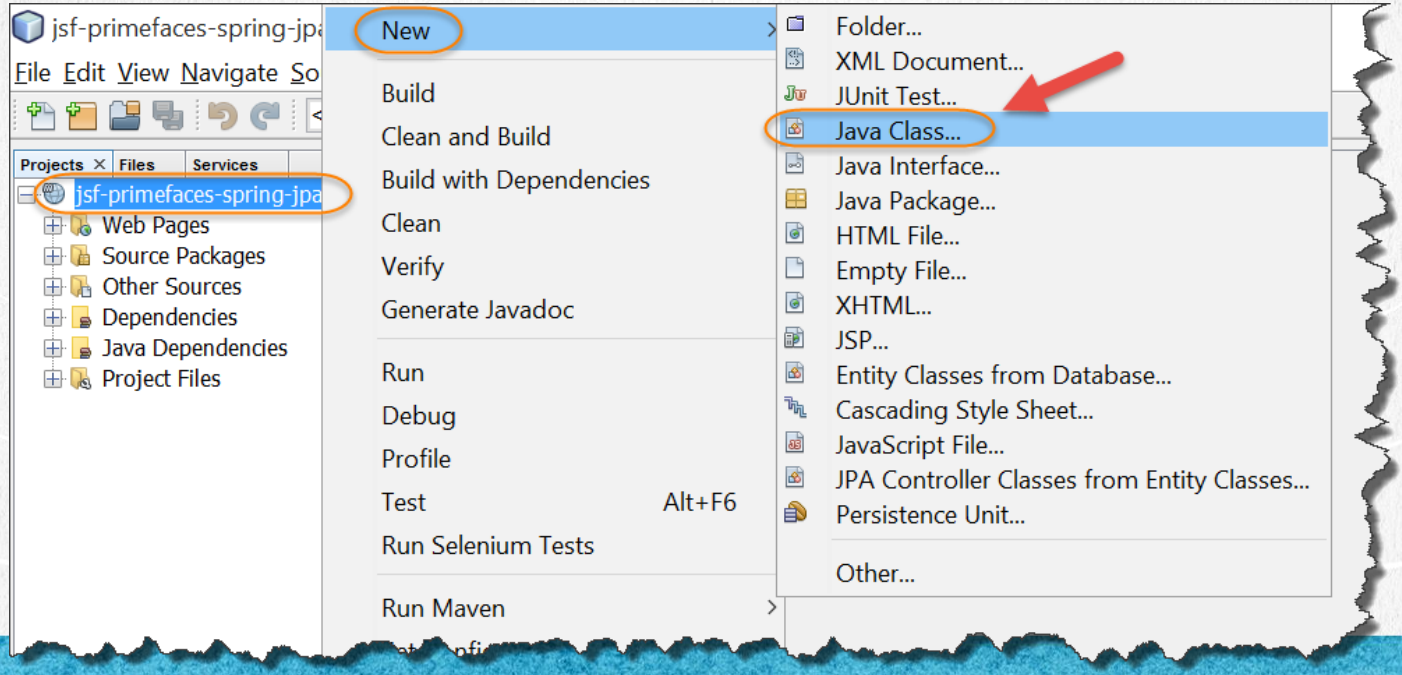
27. COPY AND PASTE THE GIF IMAGE

The result is as follows:



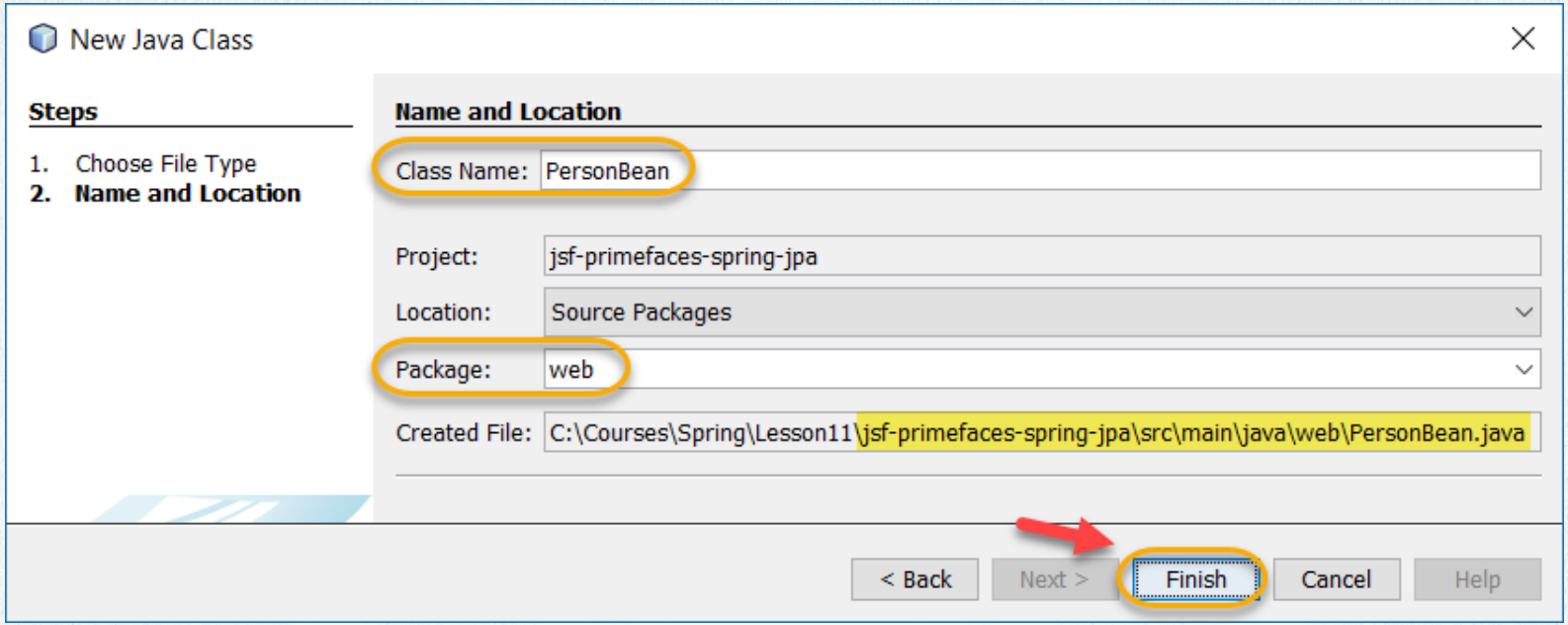
28. CREATE A NEW CLASS

We create the PersonBean.java class, which is a ManagedBean of JSF:



28. CREATE A NEW CLASS

We create the PersonBean.java class, which is a ManagedBean of JSF:



29. MODIFY THE CODE

PersonBean.java:

Click to download

```
package web;

import domain.Person;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import javax.inject.Named;
import org.primefaces.event.RowEditEvent;
import service.PersonService;

@Named
@RequestScoped
public class PersonBean {

    @Inject
    private PersonService personService;

    private Person selectedPerson;

    List<Person> people;

    public PersonBean() {
    }
```

29. MODIFY THE CODE

PersonBean.java:

Click to download

```
@PostConstruct
public void init() {
    //We start the variables
    people = personService.listPeople();
    selectedPerson = new Person();
}

public void editListener(RowEditEvent event) {
    Person person = (Person) event.getObject();
    personService.modifyPerson(person);
}

public List<Person> getPeople() {
    return people;
}

public void setPeople(List<Person> people) {
    this.people = people;
}

public Person getSelectedPerson() {
    return selectedPerson;
}

public void setSelectedPerson(Person selectedPerson) {
    this.selectedPerson = selectedPerson;
}
```

29. MODIFY THE CODE

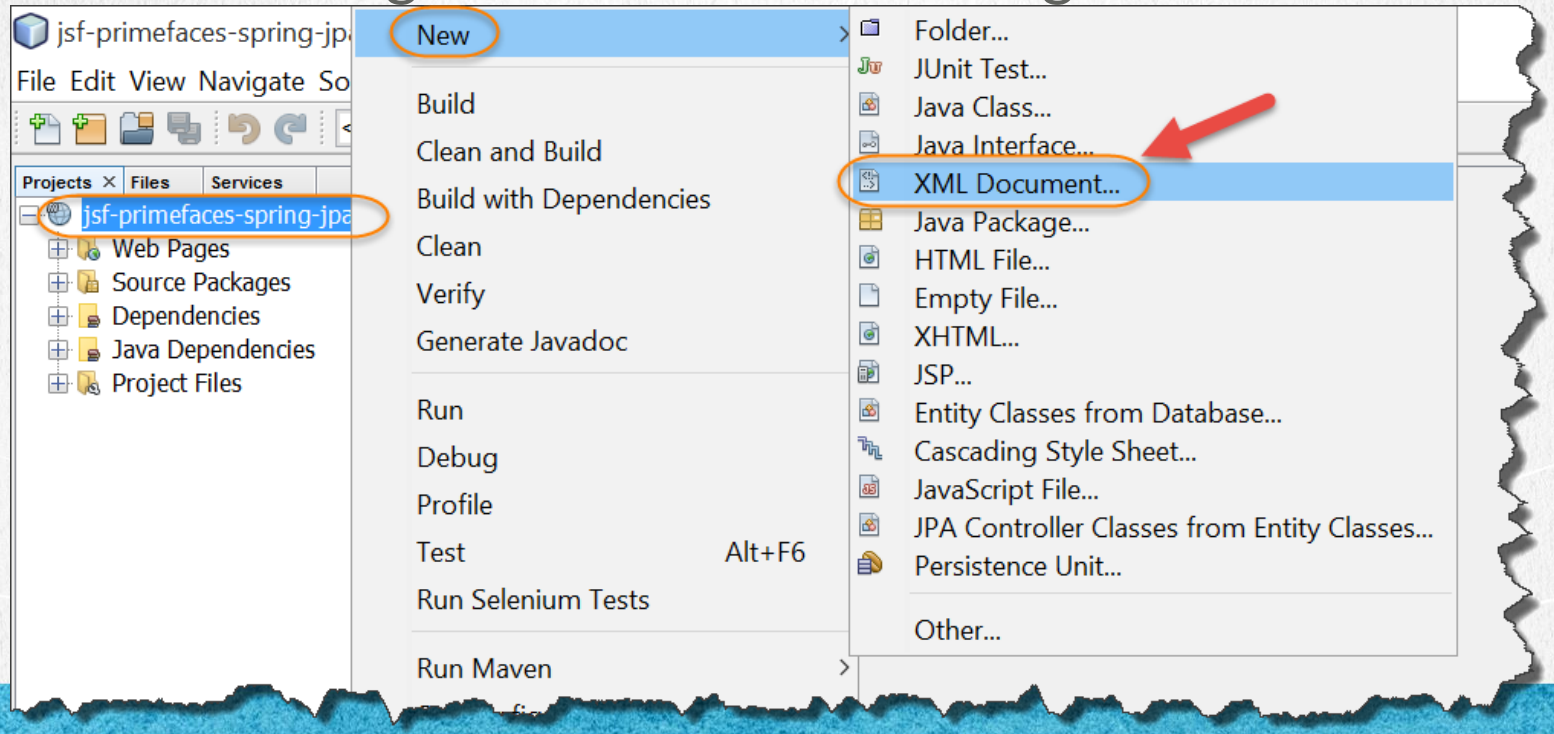
PersonBean.java:

Click to download

```
    public void restartSelectedPerson() {
        this.selectedPerson = new Person();
    }

    public void addPerson() {
        personService.addPerson(this.selectedPerson);
        //We also add it to the view (refresh the Model)
        this.people.add(this.selectedPerson);
        this.selectedPerson = null;
    }

    public void deletePerson() {
        personService.deletePerson(this.selectedPerson);
        //We also eliminate it from the view (refresh the Model)
        people.remove(this.selectedPerson);
        this.selectedPerson = null;
    }
}
```

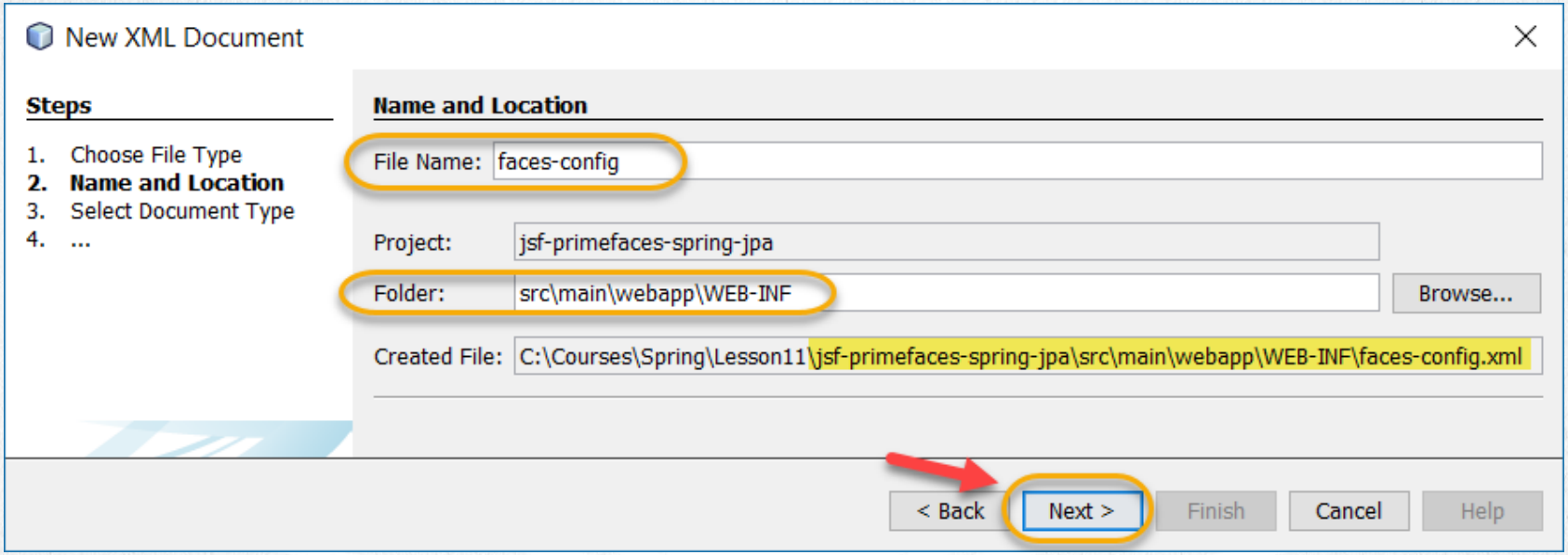
30. CREATE AN XML FILE

We add the configuration file faces-config.xml:



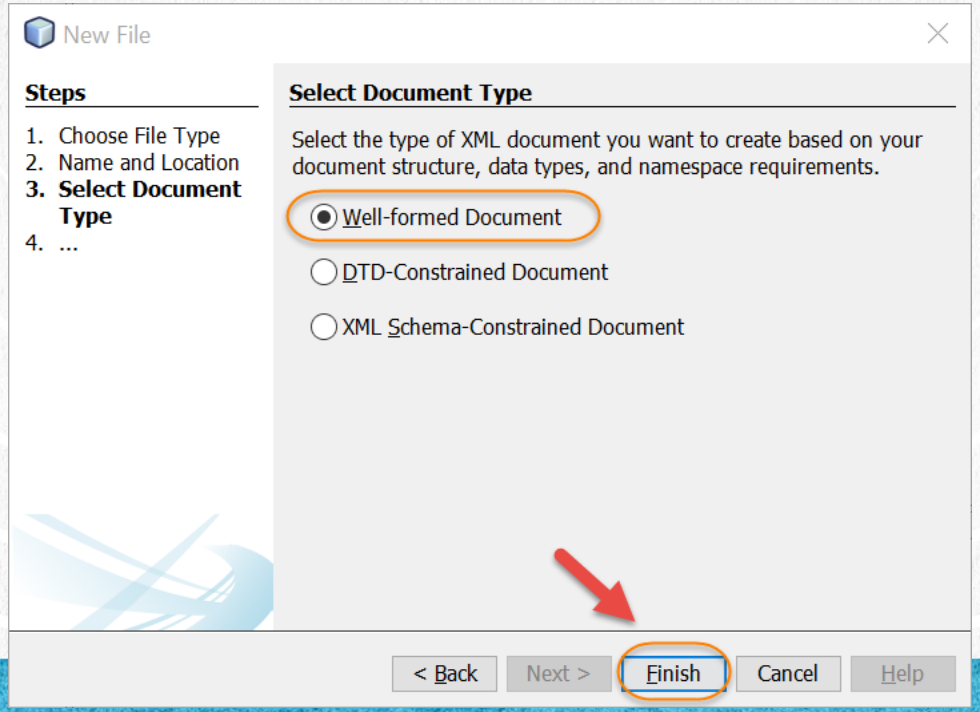
30. CREATE AN XML FILE

We add the configuration file faces-config.xml:



30. CREATE AN XML FILE

We add the configuration file faces-config.xml:



31. MODIFY THE FILE

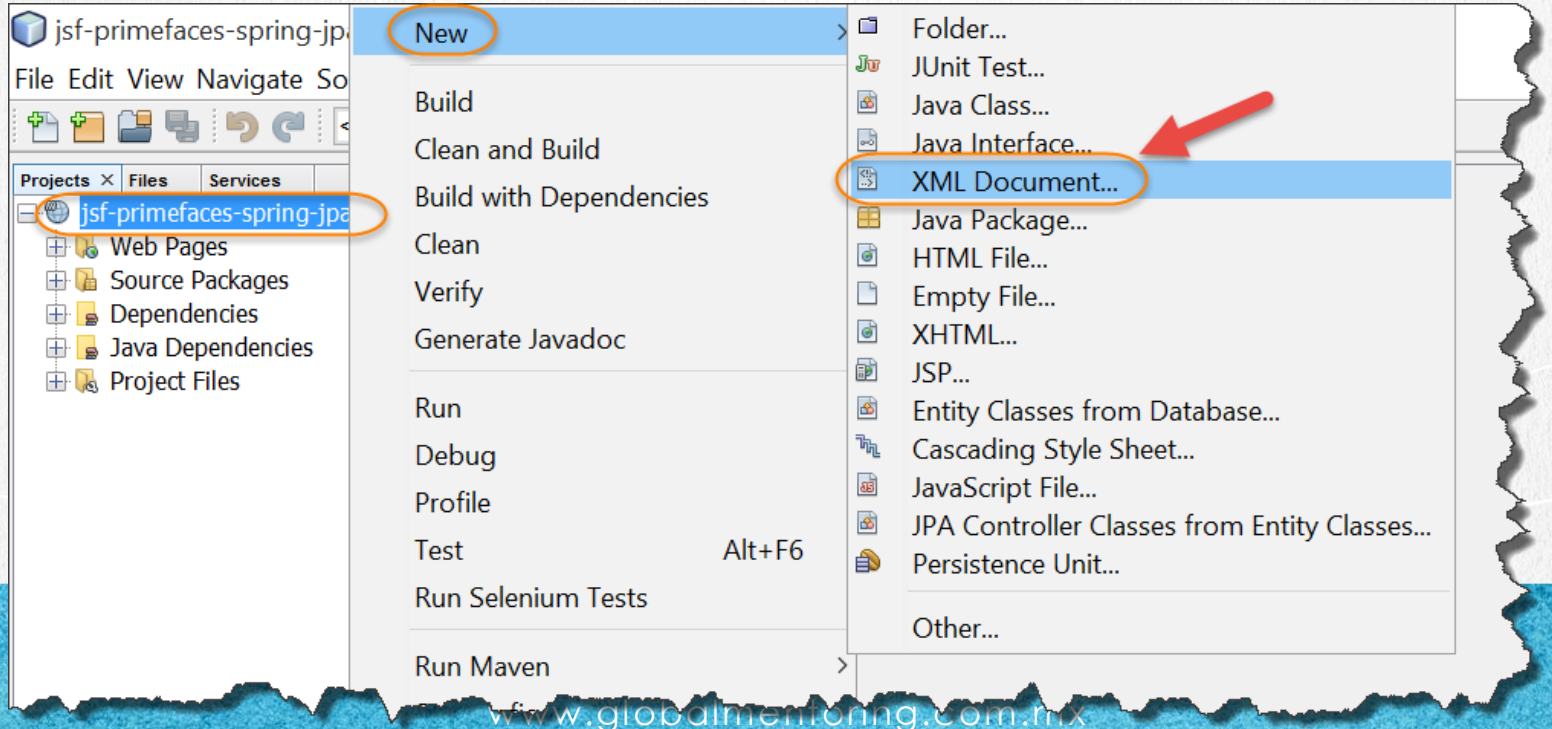
faces-config.xml:

Click to download

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
    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-facesconfig_2_3.xsd"
    version="2.3">

    <application>
        <el-resolver>org.springframework.web.jsf.el.SpringBeanFacesELResolver</el-resolver>
    </application>
</faces-config>
```

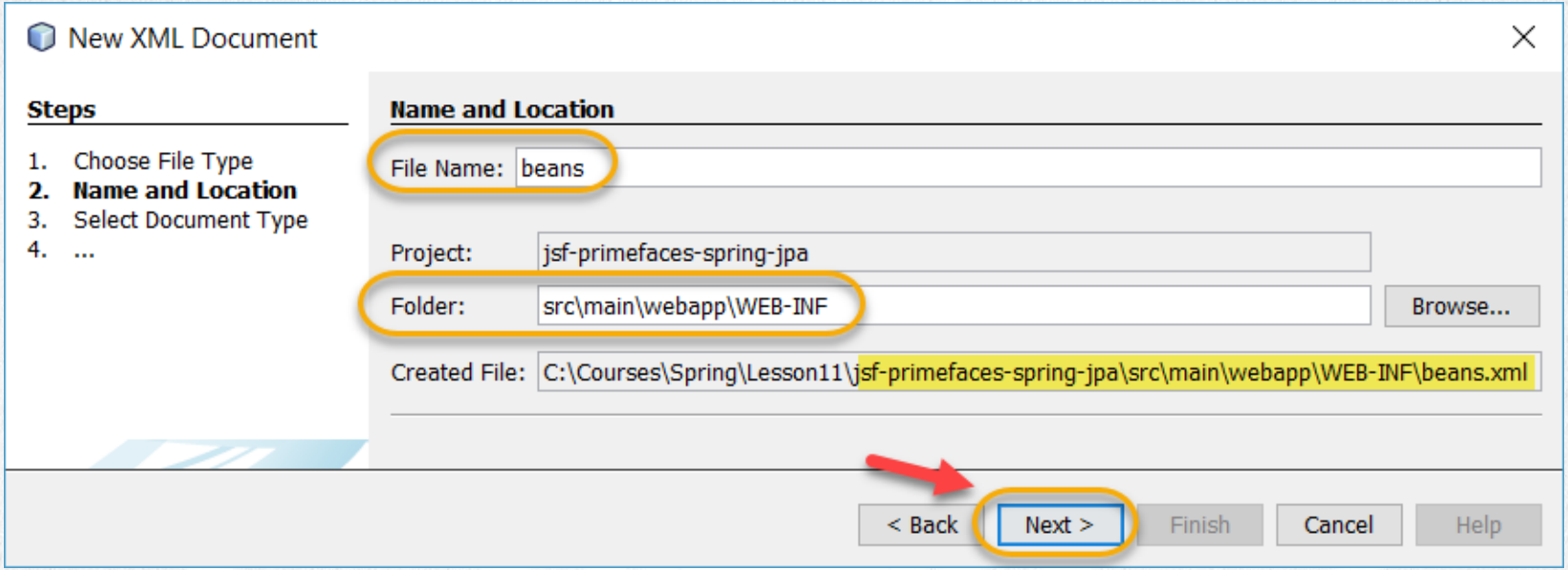
32. CREATE AN XML FILE

We add the configuration file beans.xml. This files enables the CDI injection for Spring beans and Java EE beans:



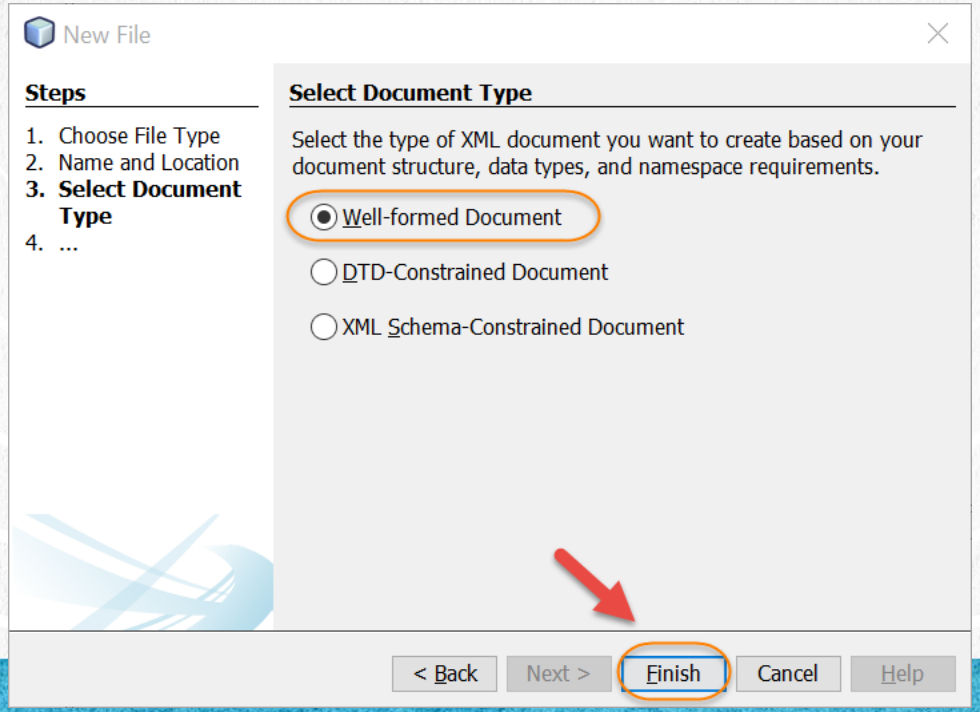
32. CREATE AN XML FILE

We add the configuration file beans.xml:



32. CREATE AN XML FILE

We add the configuration file beans.xml:



33. MODIFY THE FILE

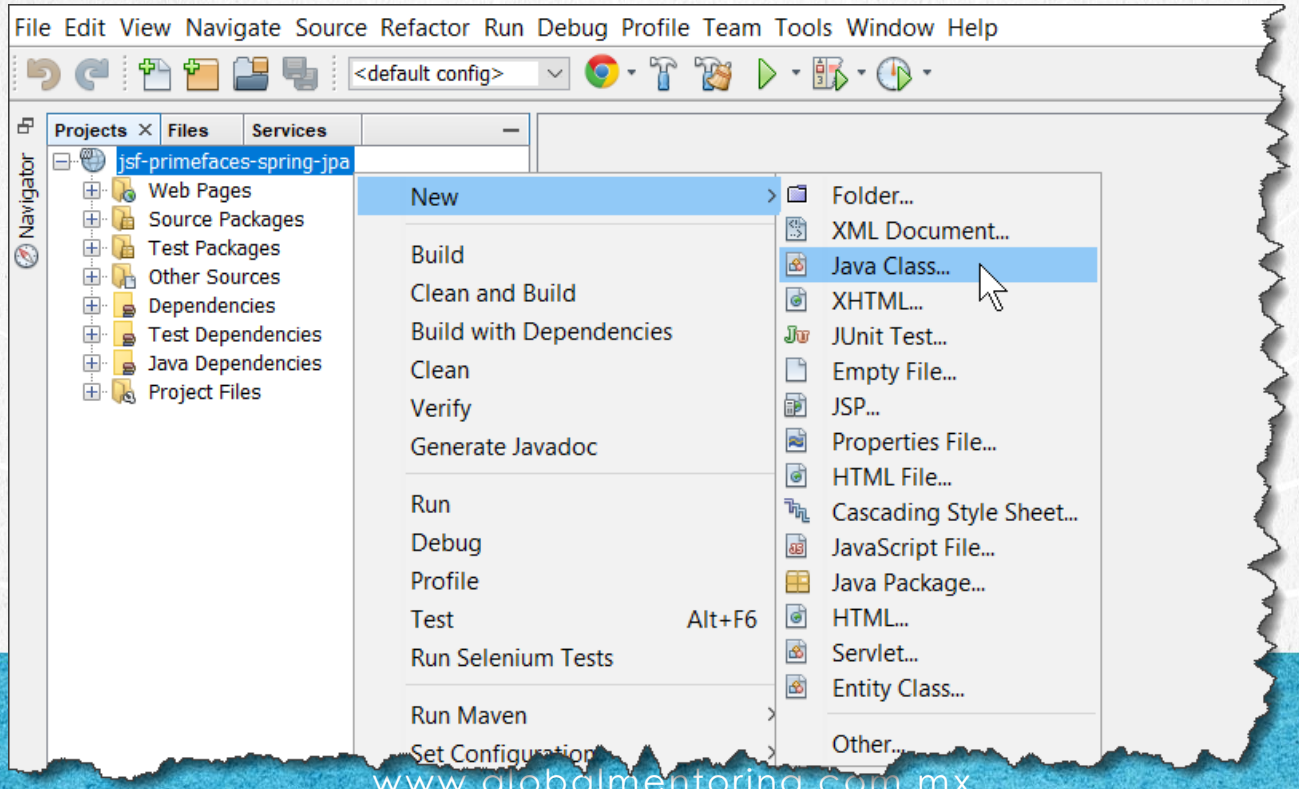
beans.xml:

Click to download

```
<beans 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/beans_2_0.xsd"
  bean-discovery-mode="all"
  version="2.0">

</beans>
```

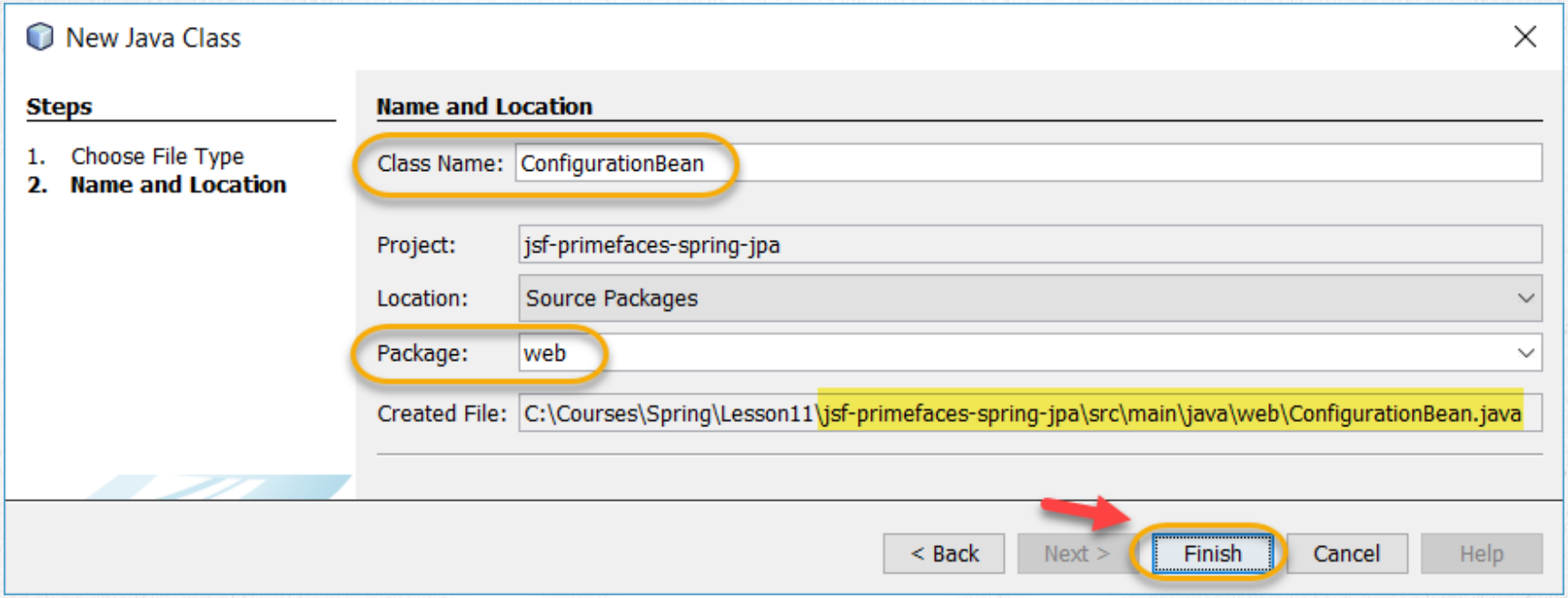
34. CREATE A JAVA CLASS

Create a new class. This file indicates the version we will use of JSF:



34. CREATE A JAVA CLASS

Create a new class. This file indicates the version we will use of JSF:



35. MODIFY THE FILE

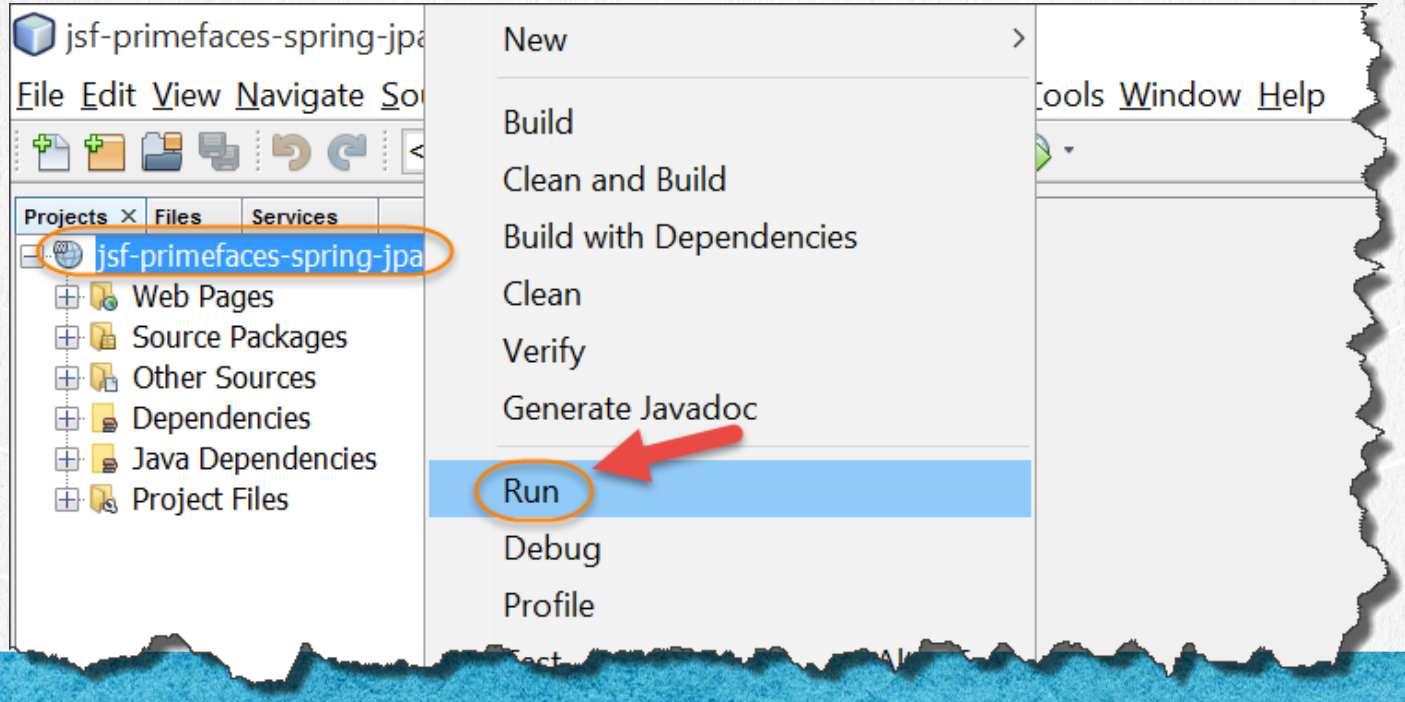
ConfigurationBean.java:

Click to download

```
package web;

import javax.faces.annotation.FacesConfig;
import static javax.faces.annotation.FacesConfig.Version.JSF_2_3;

@FacesConfig(
// Activates CDI
version = JSF_2_3
)
public class ConfigurationBean {

}
```

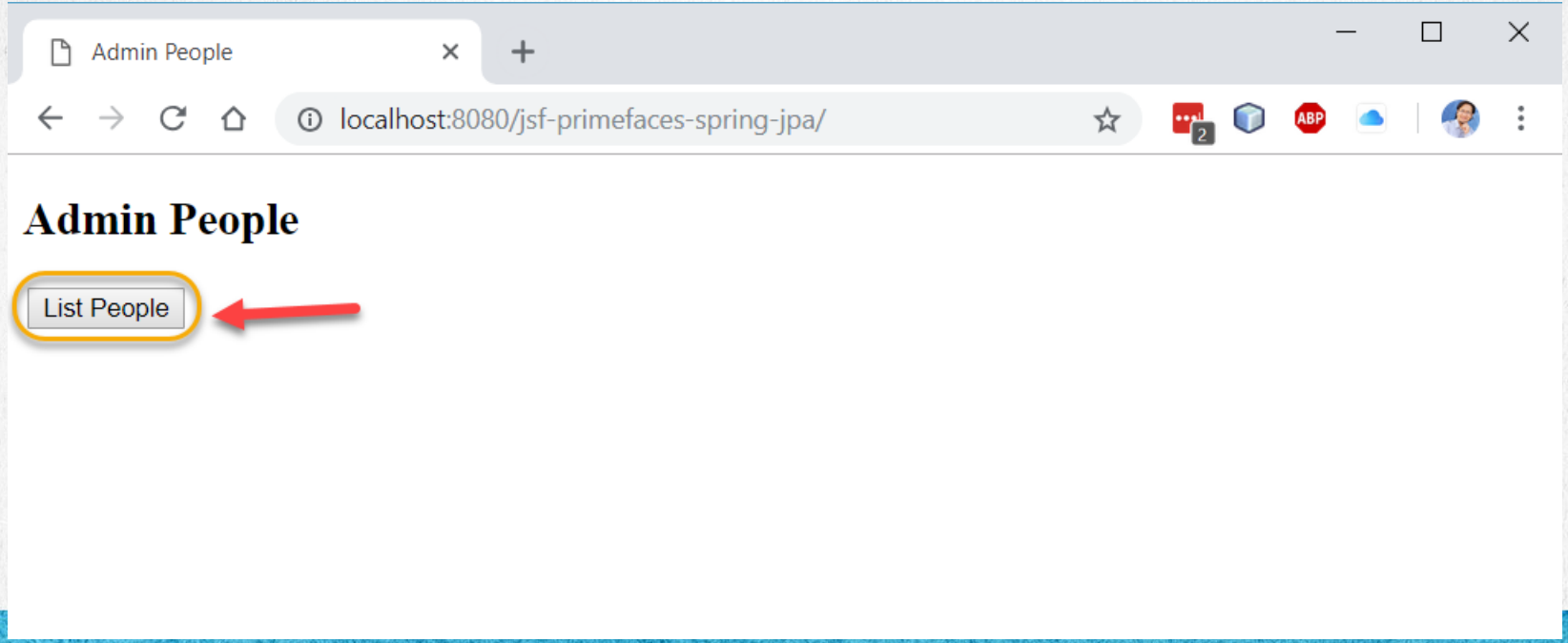
36. EXECUTE THE PROJECT

Execute the project:



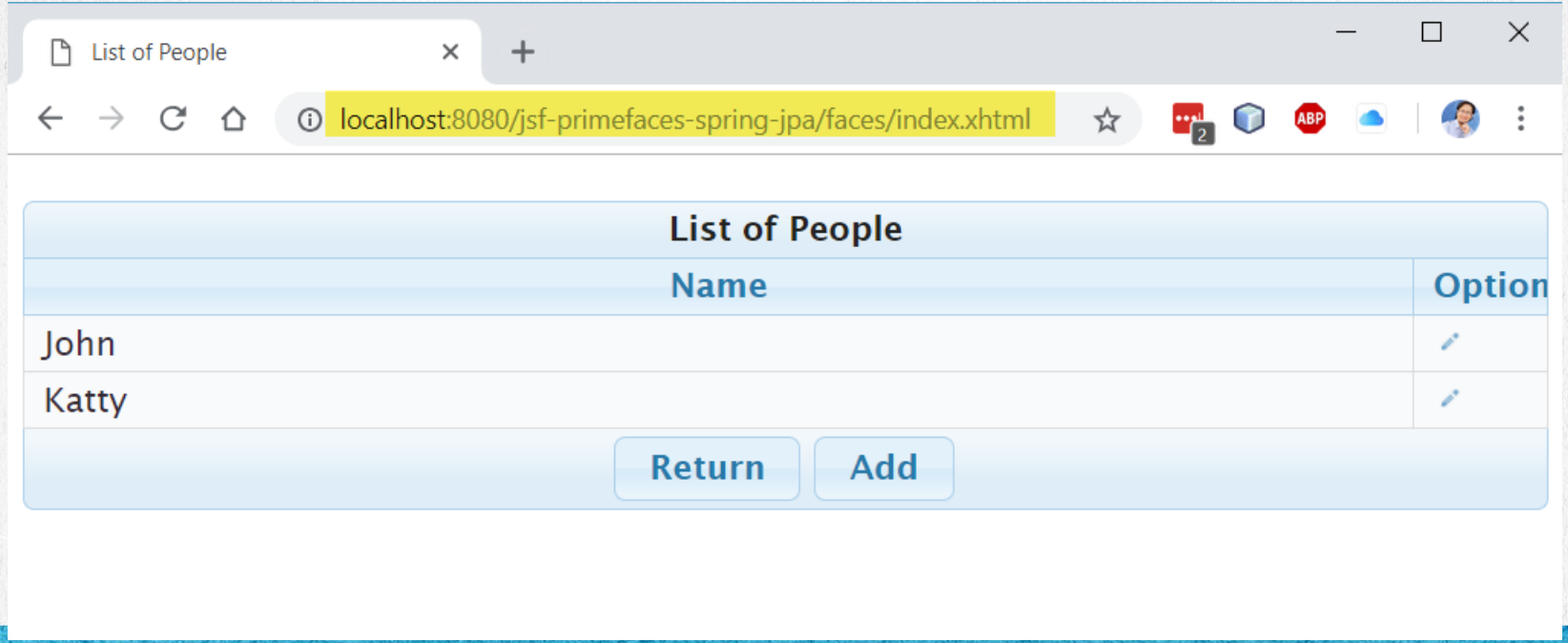
36. EXECUTE THE PROJECT

The result is as follows:



36. EXECUTE THE PROJECT

The result is as follows:



EXTRA TASKS

Test each of the use cases:
Add (click on the Add button)
Modify (click on the options column)
Delete (right click and delete)

Check the Glassfish server log and check that everything works properly.

EXERCISE CONCLUSION

With this exercise we have made the integration of JSF / PrimeFaces, Spring and JPA.

This application of 3 layers, presentation layer, business and data respectively, gives us an overview of how to solve one of the most used architectures in Java business developments using Spring Framework.

Each of the layers performs a very specific task, however Spring is the framework responsible for orchestrating the integration of other technologies.

We can take this project as a basis for our own web business projects with Java and Spring Framework.

ONLINE COURSE

SPRING FRAMEWORK

By: Eng. Ubaldo Acosta