SPRING FRAMEWORK COURSE

EXERCISE

SPRING AND 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:

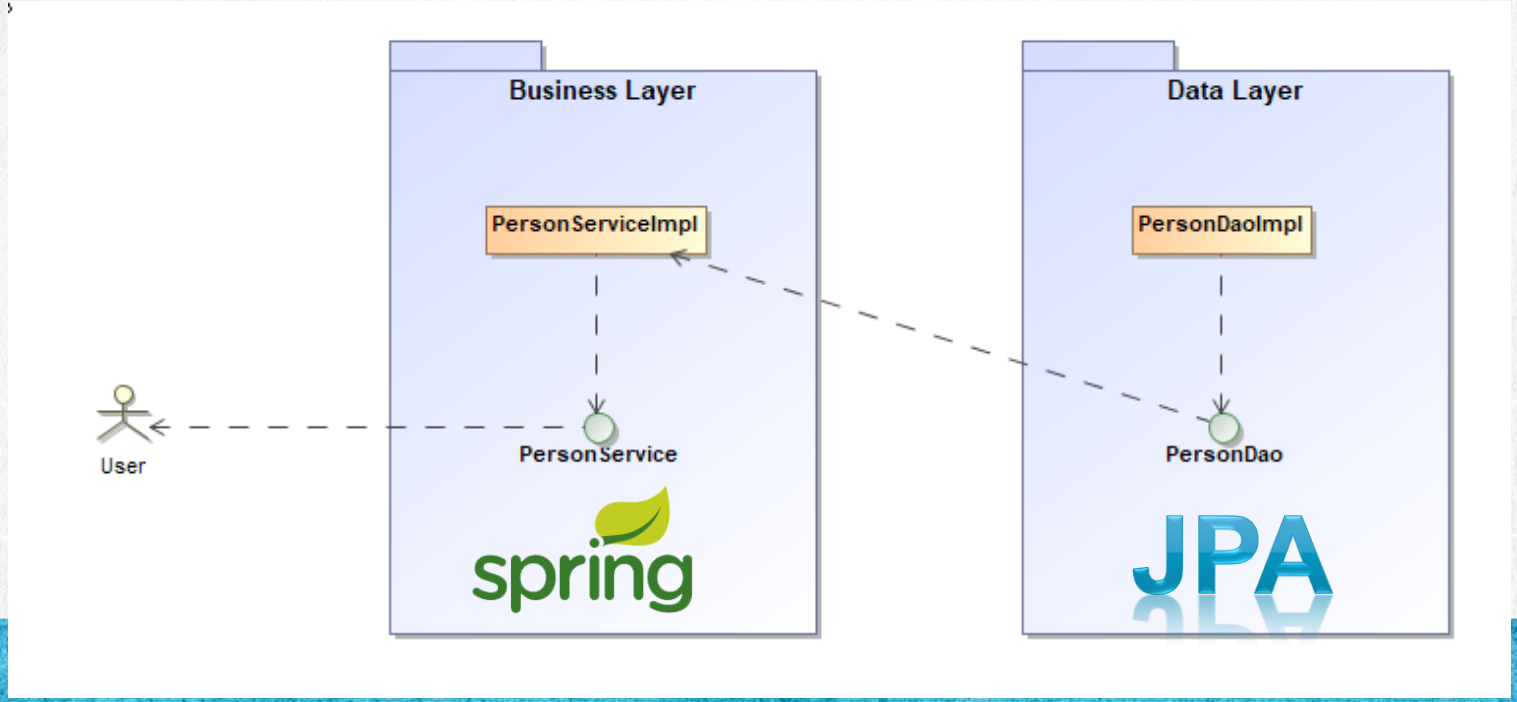
Output × Test Results

Retriever Output × Test (TestPersonService) ×

```
[EL Fine]: sql: ClientSession(1878225987)--Connection(981159997)--Thread(Thread[main,5,main])--SELECT id_person, NAME FROM PERSON
21:21:33 [main] INFO   - Person: Person{idPerson=1, name=John}
21:21:33 [main] INFO   - Person: Person{idPerson=2, name=Katty}
21:21:33 [main] INFO   - End of test testTransactions

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.961 s - in test.TestPersonService
21:21:33 [Thread-1] INFO  org.springframework.orm.jpa.LocalEntityManagerFactoryBean - Closing JPA EntityManagerFactory for persistence unit 'PersistenceUnit'
[EL Config]: connection: ServerSession(632475595)--Connection(981159997)--Thread(Thread[Thread-1,5,main])--disconnect
[EL Info]: connection: ServerSession(632475595)--Thread(Thread[Thread-1,5,main])--/file:/C:/Courses/Spring/Lesson10/spring-jpa/target/classes/_PersistenceUnit
[EL Config]: connection: ServerSession(632475595)--Connection(913746983)--Thread(Thread[Thread-1,5,main])--disconnect

Results:
```

CLASS DIAGRAM

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



DATABASE WITH MYSQL

- For this exercise we are going to use the MySql database so that you can observe how to work with data from a physically and non-embedded base such as H2.

- If you still do not have the installation of MySql, you can follow the installation guide:

Installation of MySql installation:

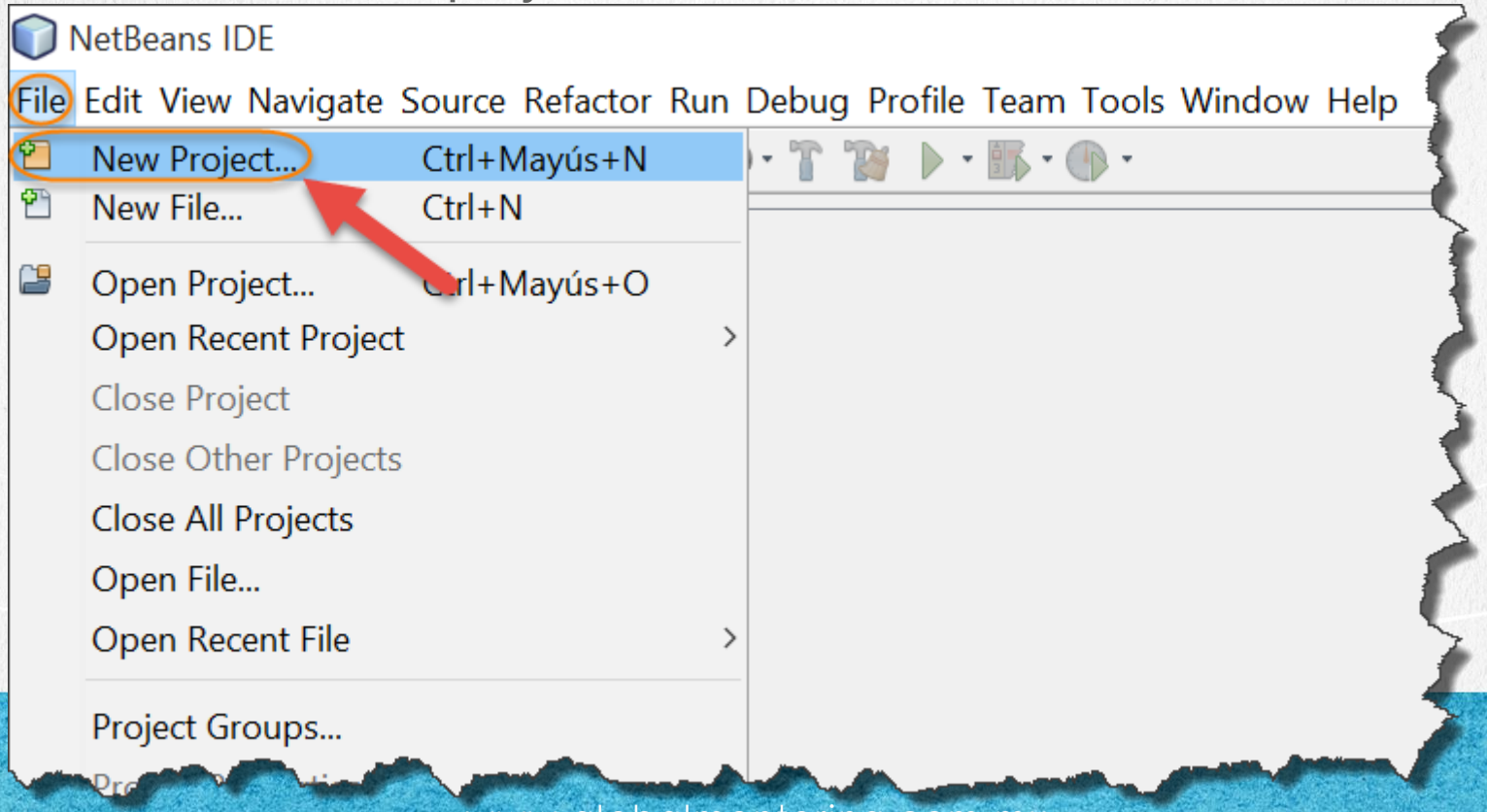
http://icursos.net/en/Installations/CJ-B-Exercise-06-MySqlInstalation.pdf

And creation of the database in MySql:

http://icursos.net/en/Installations/CJ-B-Exercise-03-MySqlDataBase.pdf

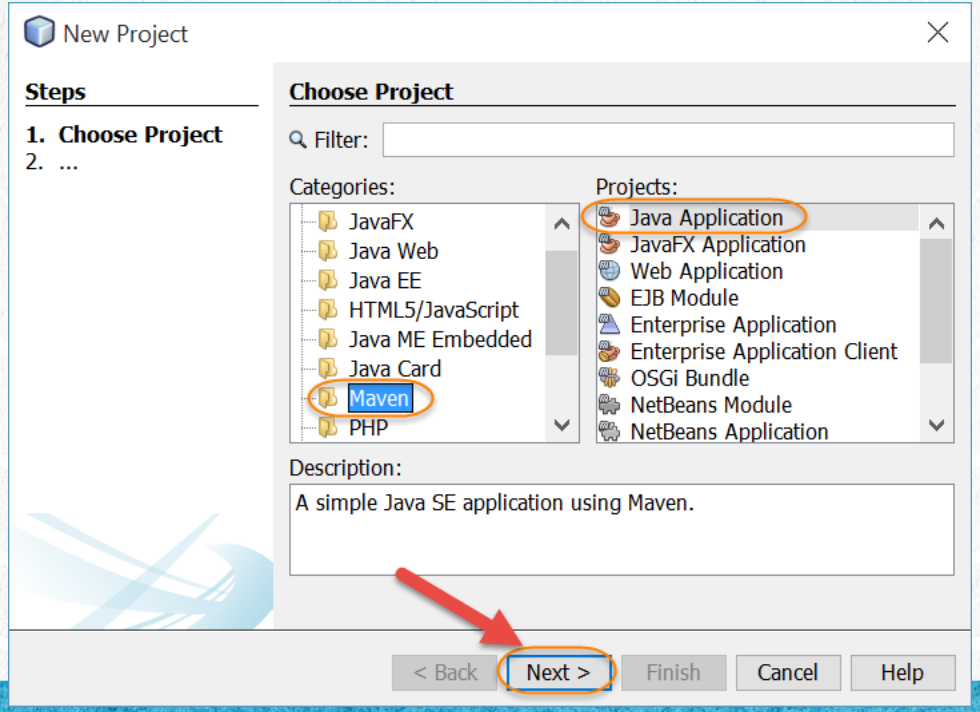
1. CREATE A NEW PROJECT

We created a new Maven project:



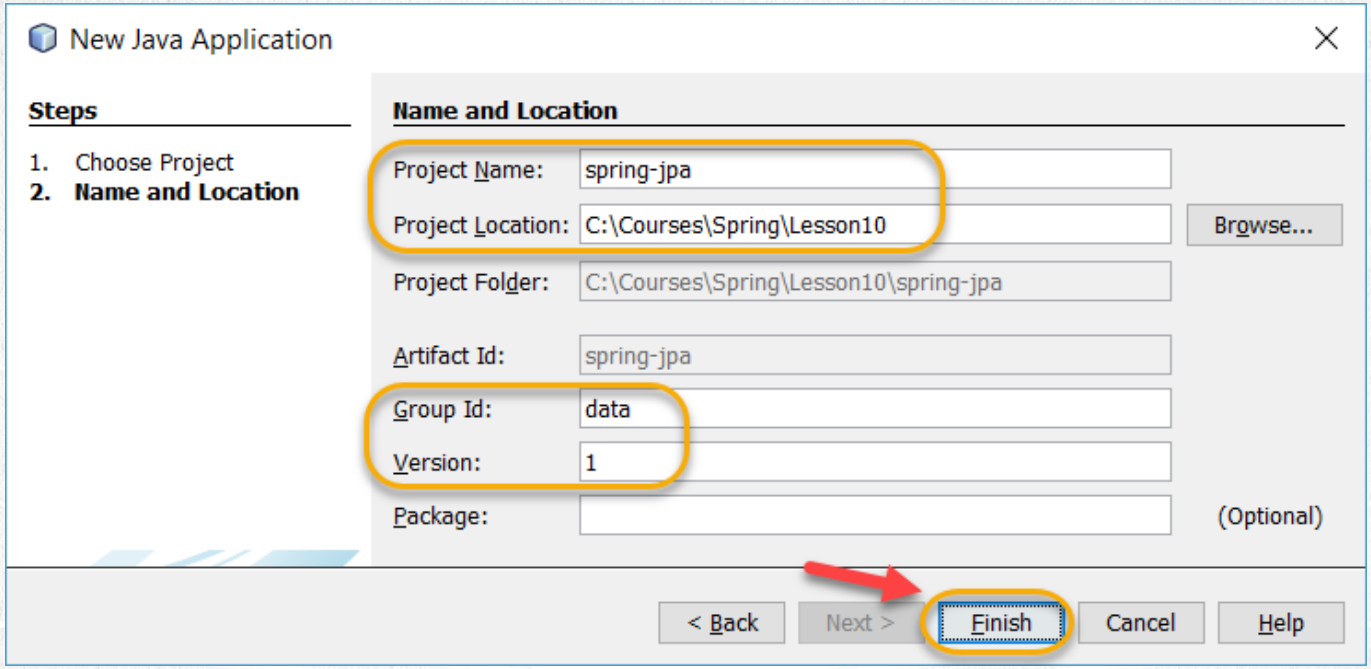
1. CREATE A NEW PROJECT

Selecting a new Maven project:



1. CREATE A NEW PROJECT

We write the following values:



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
- mysql
- junit
- log4j
- Jpa (Eclipse Implementation)

2. MODIFY THE CODE

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>data</groupId>
    <artifactId>spring-jpa</artifactId>
    <version>1</version>
    <packaging>jar</packaging>
    <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>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
```

2. MODIFY THE CODE

pom.xml:

Click to download

```
<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.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>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
</dependency>
```

2. MODIFY THE CODE

pom.xml:

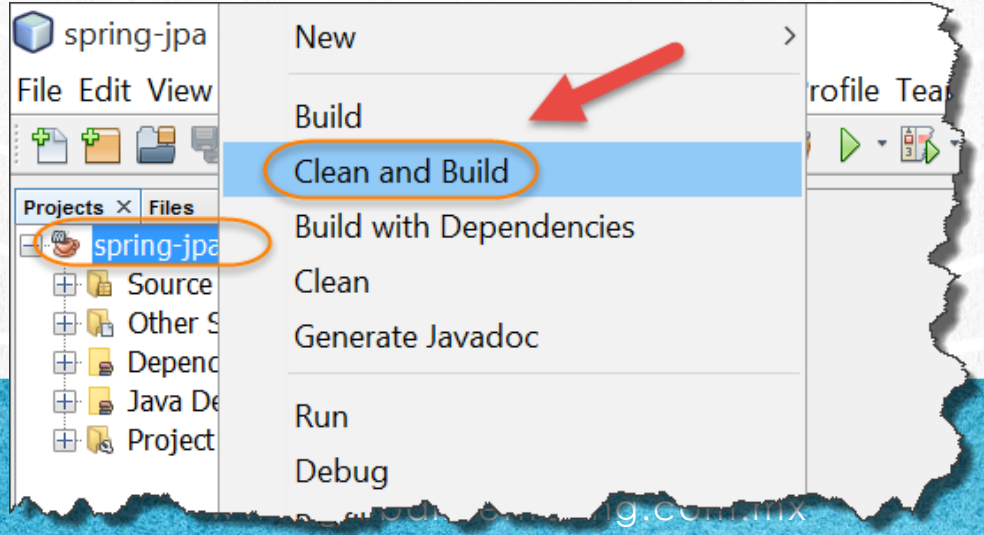
Click to download

```
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </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>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.0</version>
            </plugin>
        </plugins>
    </build>
</project>
```

3. MAVEN REPOSITORY UPDATE

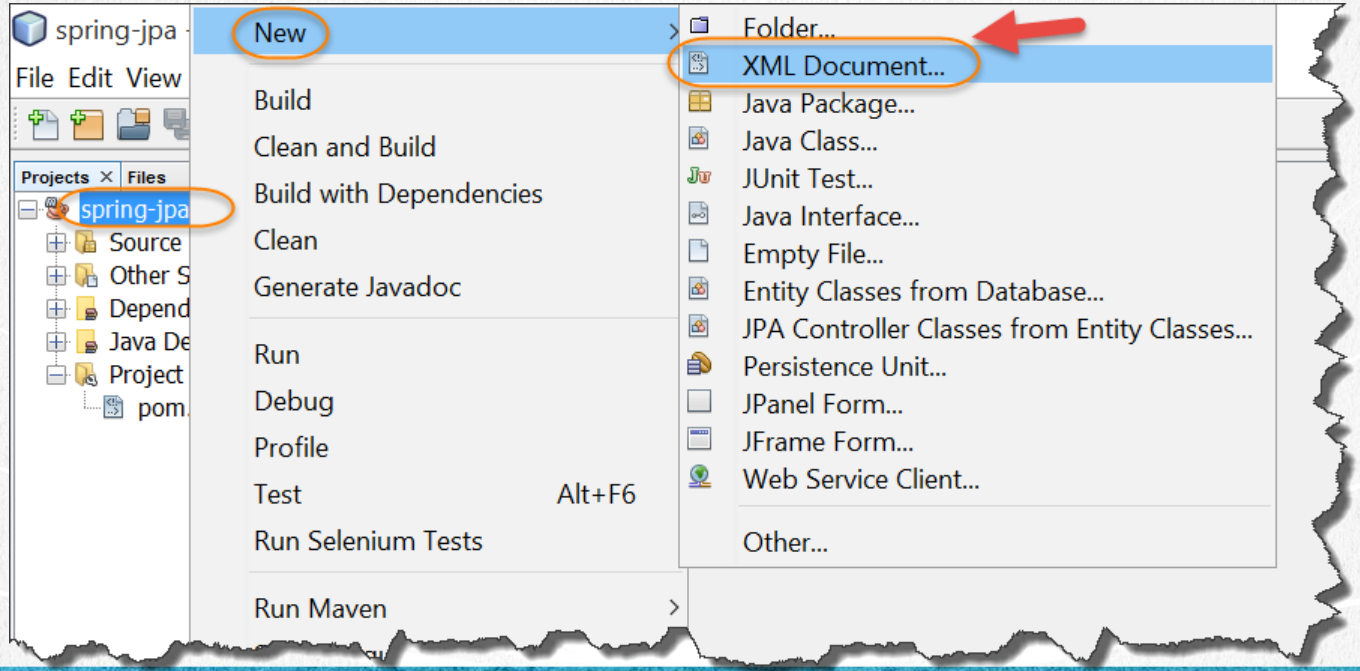
To update the Maven repository, which will be responsible for administering the .jar libraries of our project, it is enough to make clean & build our project

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



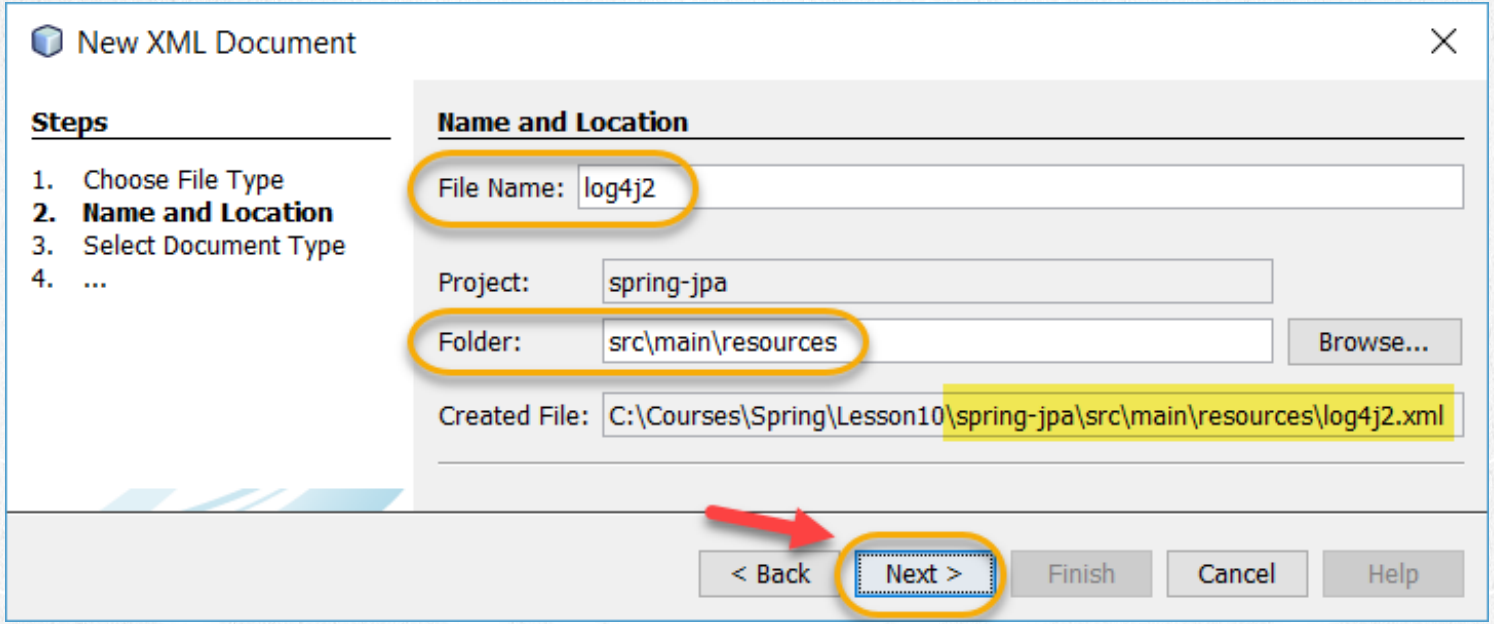
4. CREATE AN XML FILE

- We create the log4j2.xml file:



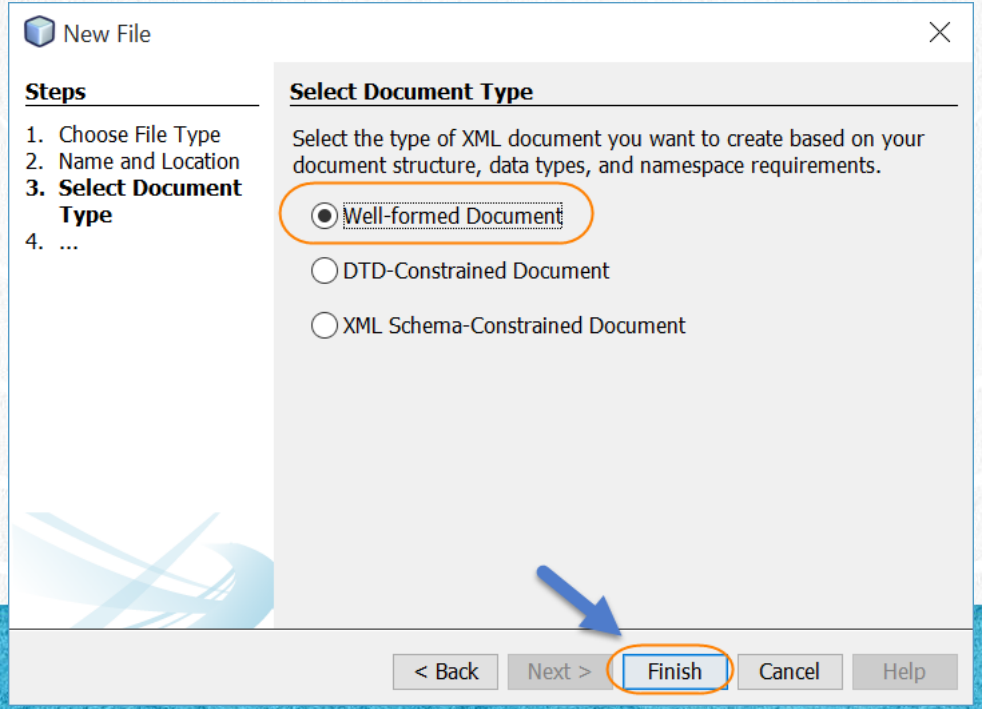
4. CREATE AN XML FILE

- 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 CODE

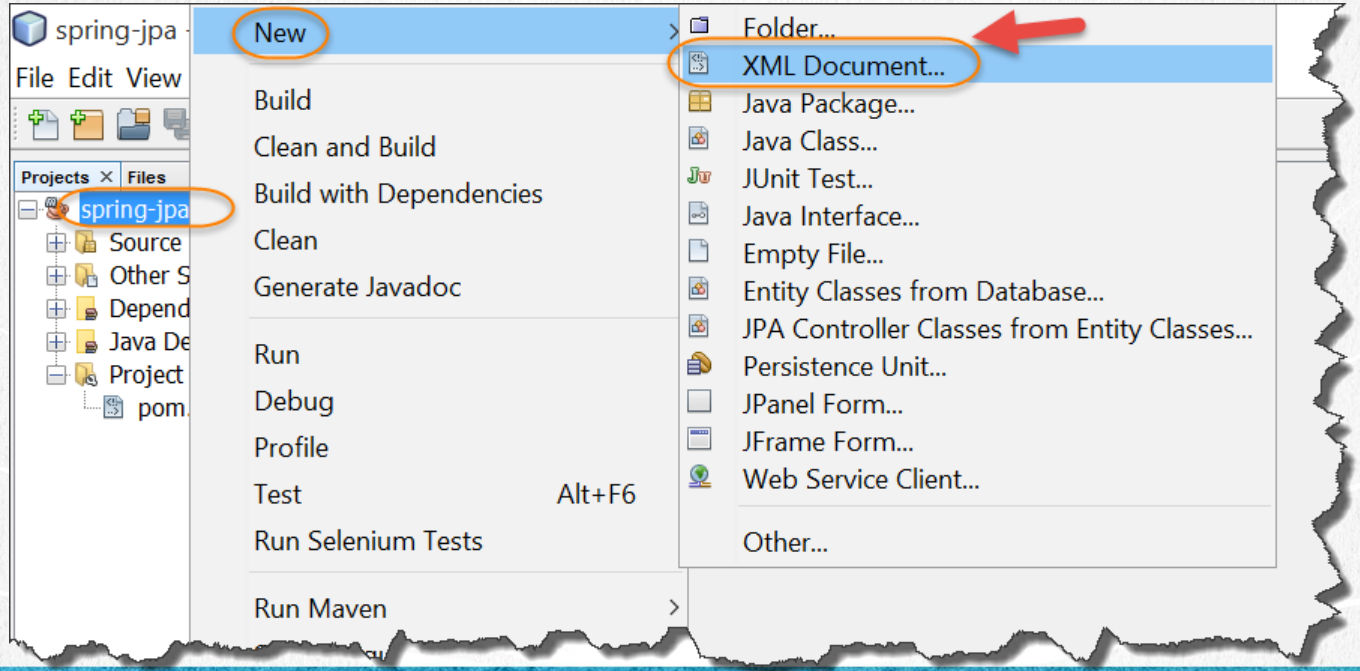
log4j2.xml:

Click to download

```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>
        <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>
```

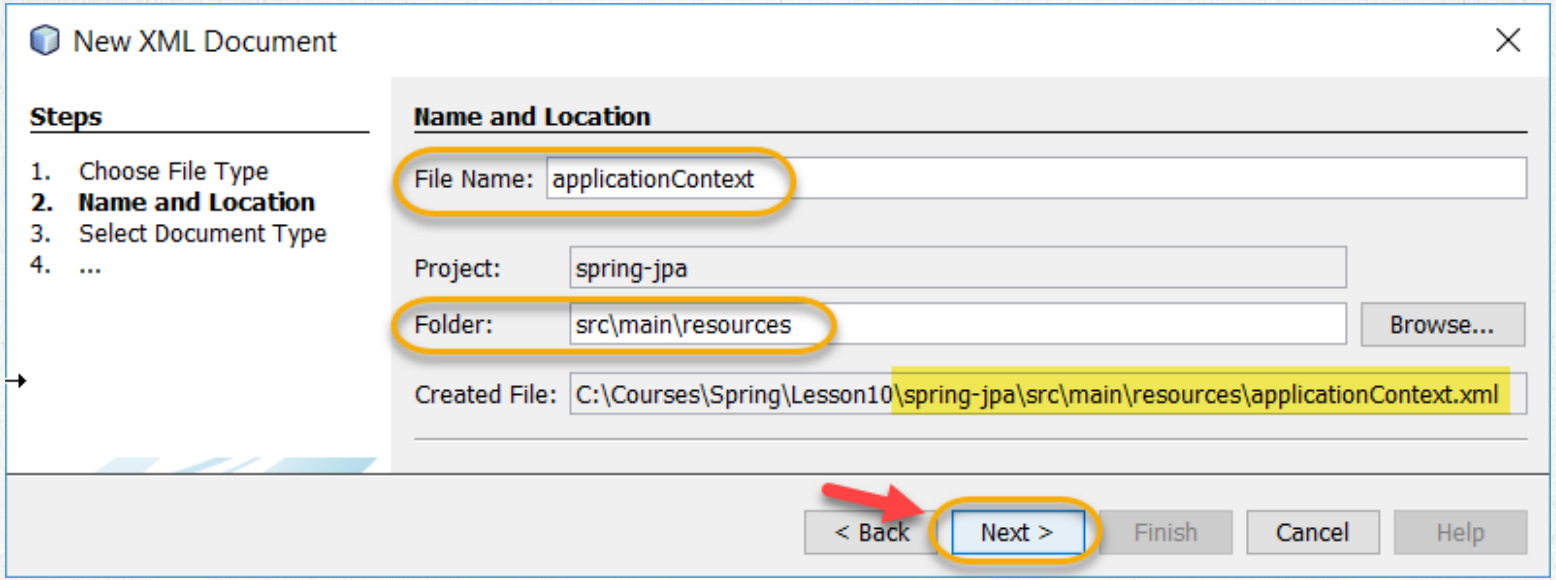
6. CREATE AN XML FILE

- We create the applicationContext.xml file :



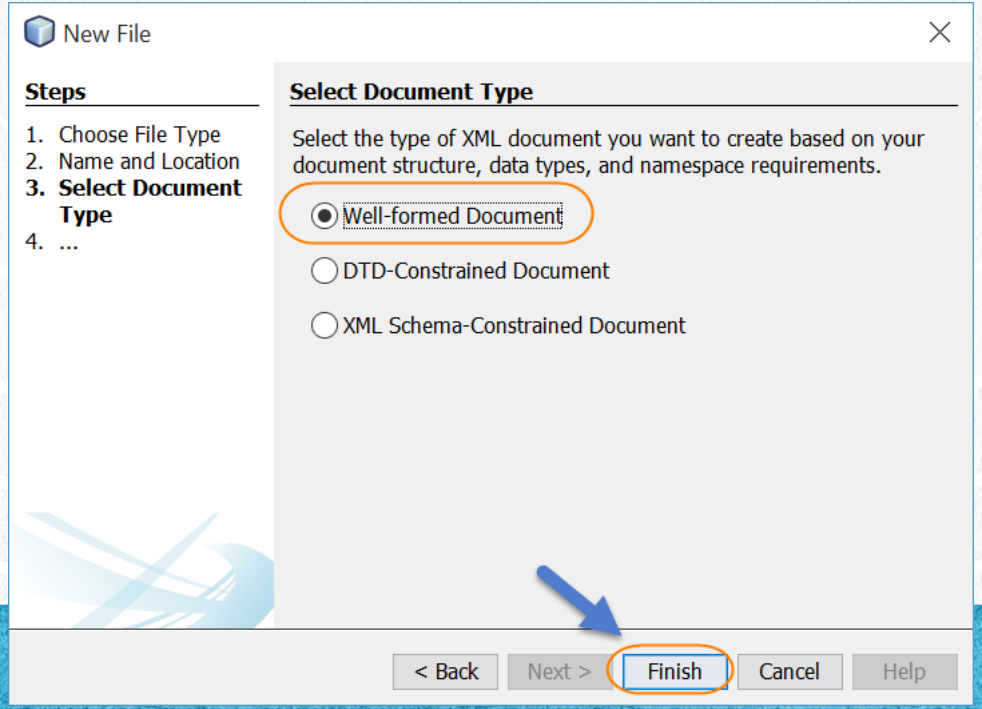
6. CREATE AN XML FILE

- We create the applicationContext.xml file:



6. CREATE AN XML FILE

- We created the applicationContext.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

applicationContext.xml:

Click to download

```xml
<?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"
       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/tx
                http://www.springframework.org/schema/tx/spring-tx.xsd">

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

    <!-- Get the injected entity manager at the Spring factory -->
    <bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor" />

    <!-- Definition Adapter EclipseLink JPA Vendor -->
    <bean id="jpaVendorAdapter" class="org.springframework.orm.jpa.vendor.EclipseLinkJpaVendorAdapter"
          p:databasePlatform="org.eclipse.persistence.platform.database.MySQLPlatform" p:showSql="true"/>
```

7. MODIFY THE FILE

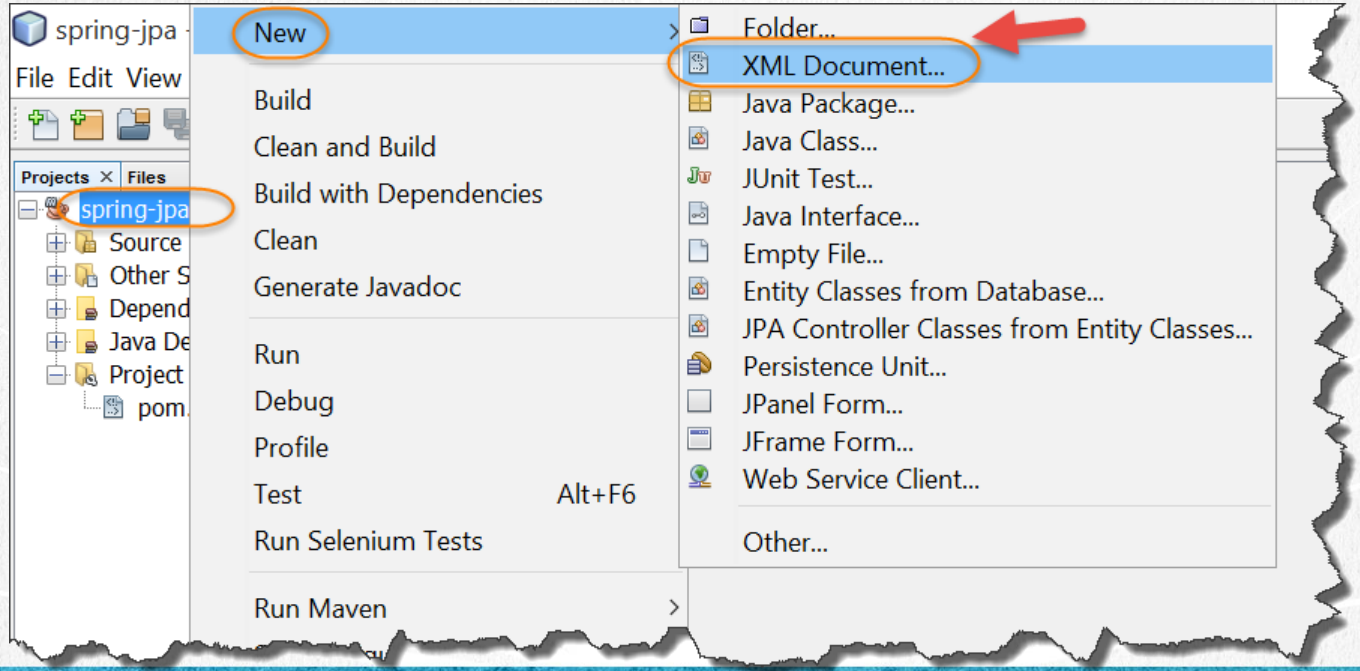
applicationContext.xml:

Click to download

```
    <!-- Entity Manager Factory -->
    <bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean"
          p:persistenceUnitName="PersistenceUnit"
          p:jpaVendorAdapter-ref="jpaVendorAdapter"/>

    <!-- Transaction Manager -->
    <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager"
          p:entityManagerFactory-ref="entityManagerFactory" />

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

</beans>
```

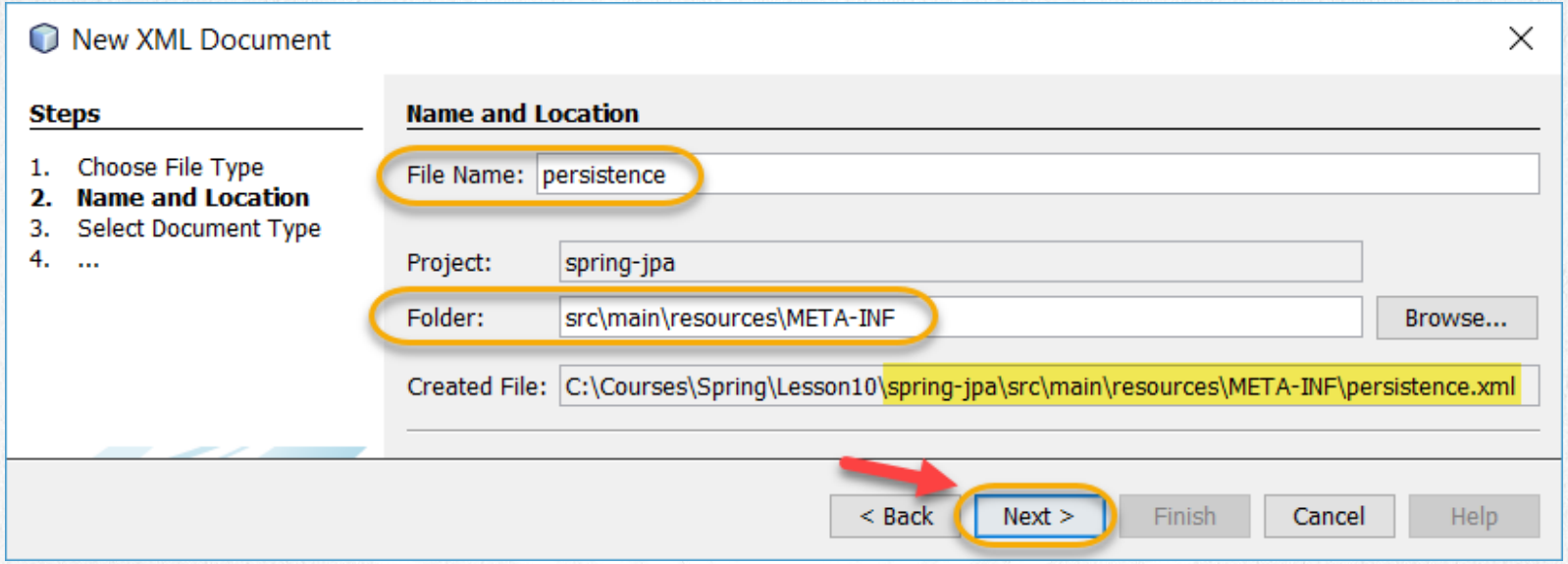
8. CREATE AN XML FILE

- We create the persistence.xml file:



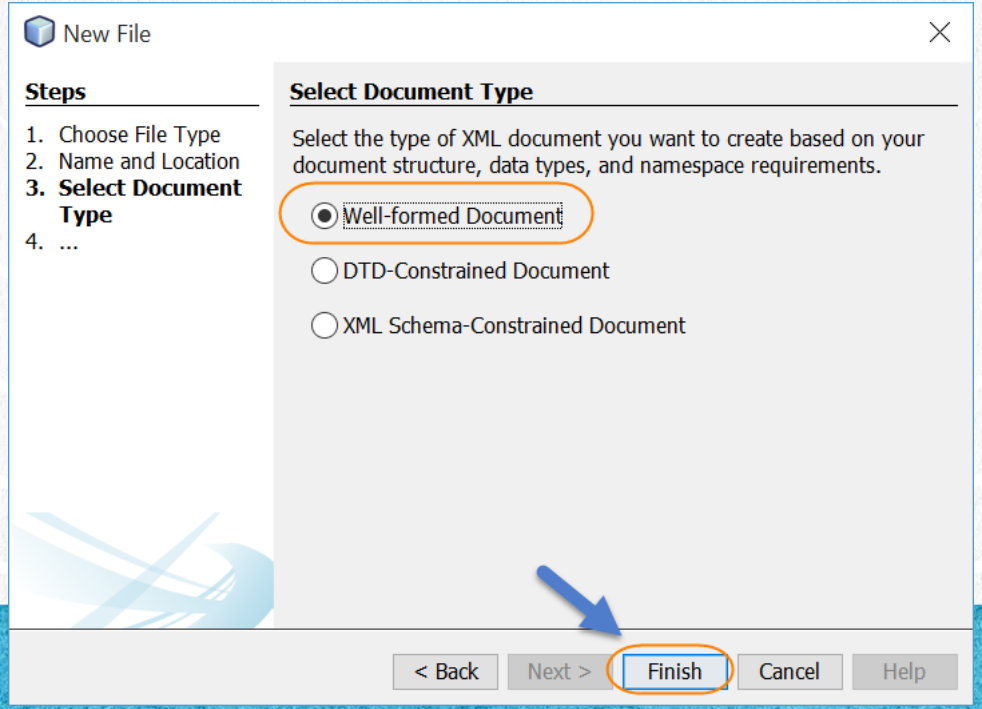
8. CREATE AN XML FILE

- We create the persistence.xml file:



8. 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:



9. MODIFY THE FILE

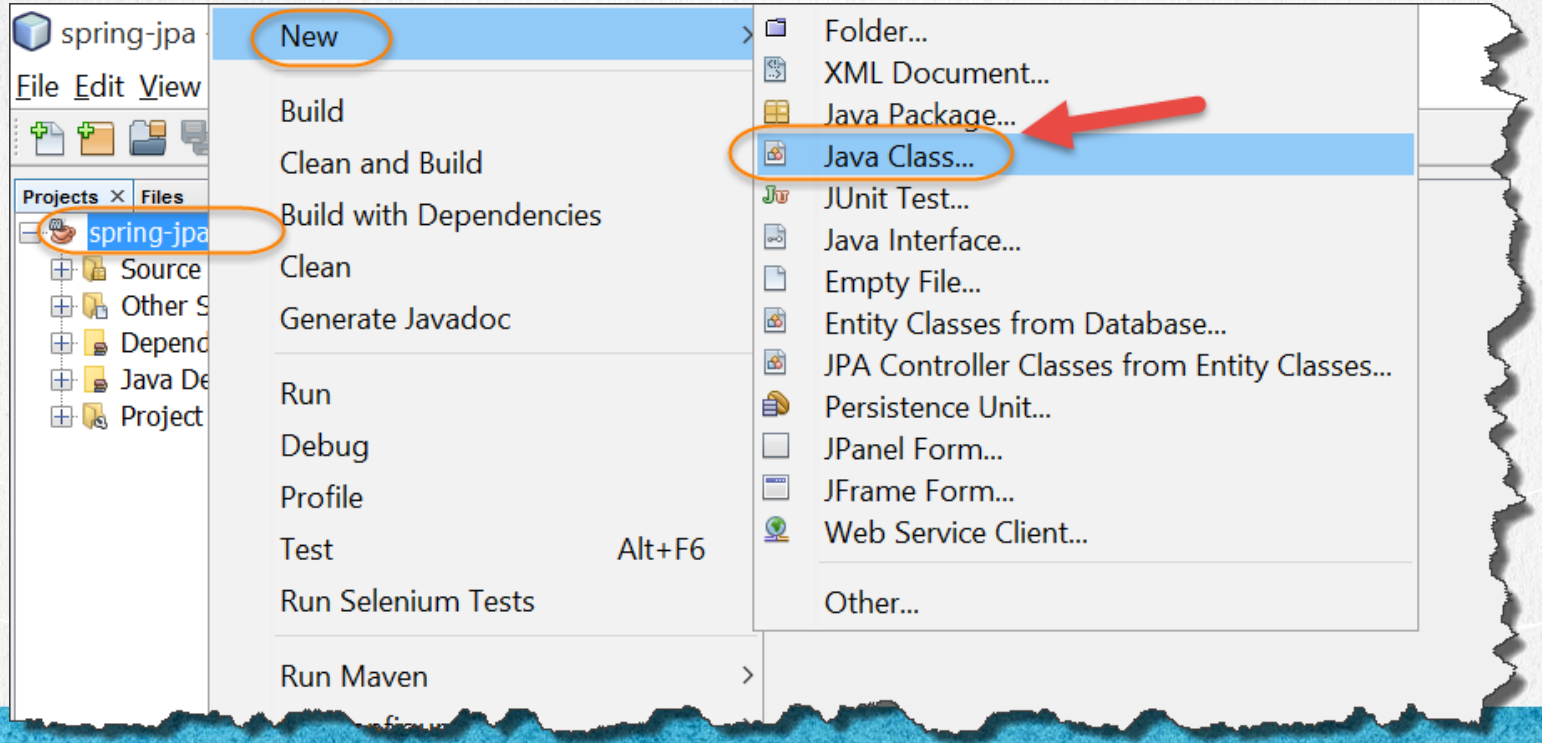
persistence.xml:

Click to download

```xml
<?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="RESOURCE_LOCAL">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <!--We list the Entity classes-->
        <class>domain.Person</class>
        <properties>
            <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/test?useSSL=false"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
            <property name="javax.persistence.jdbc.password" value="admin"/>
            <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>
```

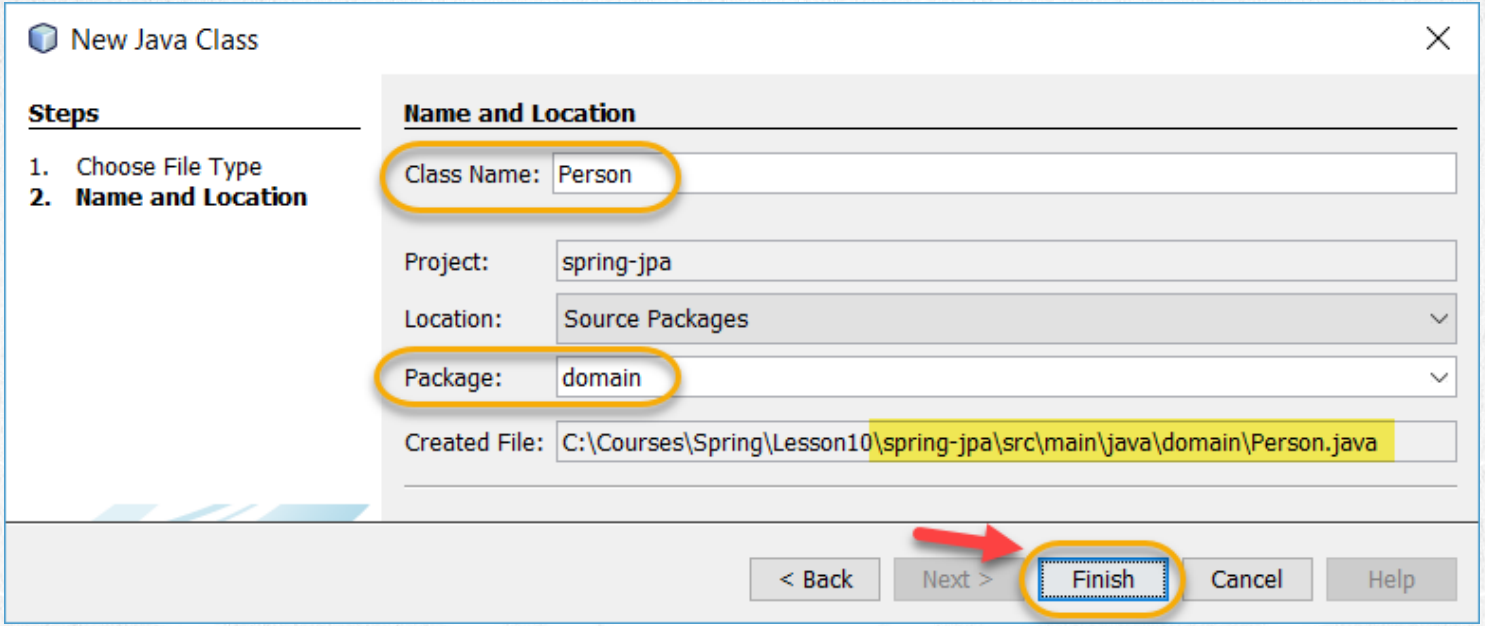
10. CREATE A NEW CLASS

Next we create the Person.java class:



10. CREATE A NEW CLASS

Next we create the Person.java class:



11. MODIFY THE FILE

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 FILE

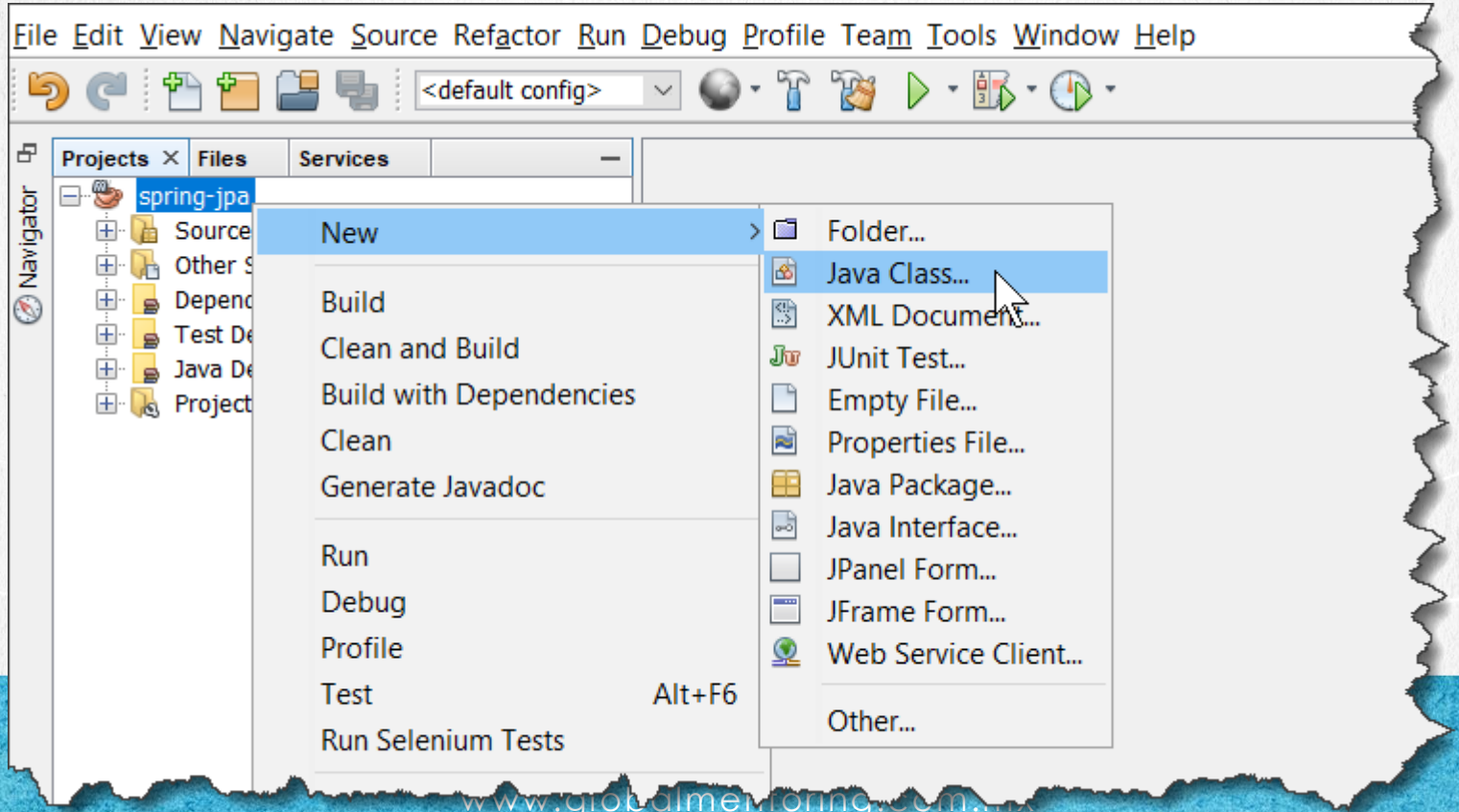
Person.java:

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 + '}';
    }

}
```

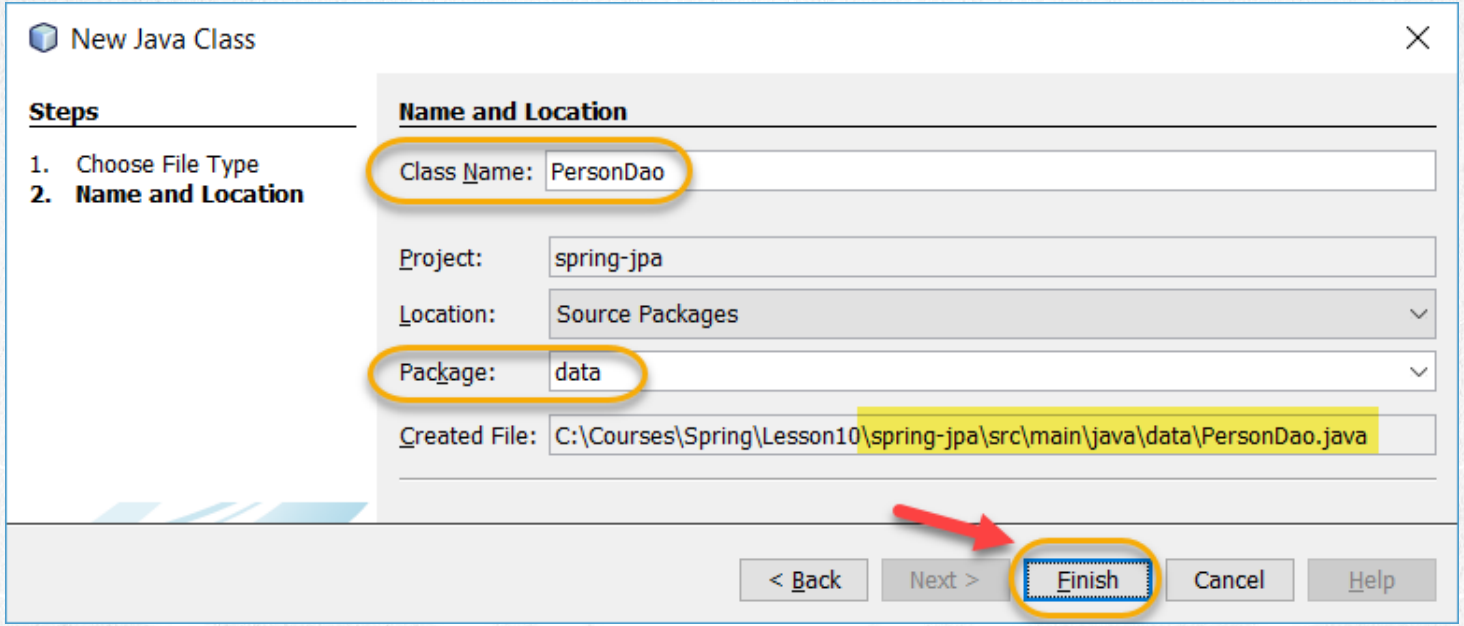
12. CREATE A NEW CLASS

Next we create the PersonDao.java interface:



12. CREATE A NEW CLASS

Next we create the PersonDao.java interface:



13. MODIFY THE FILE

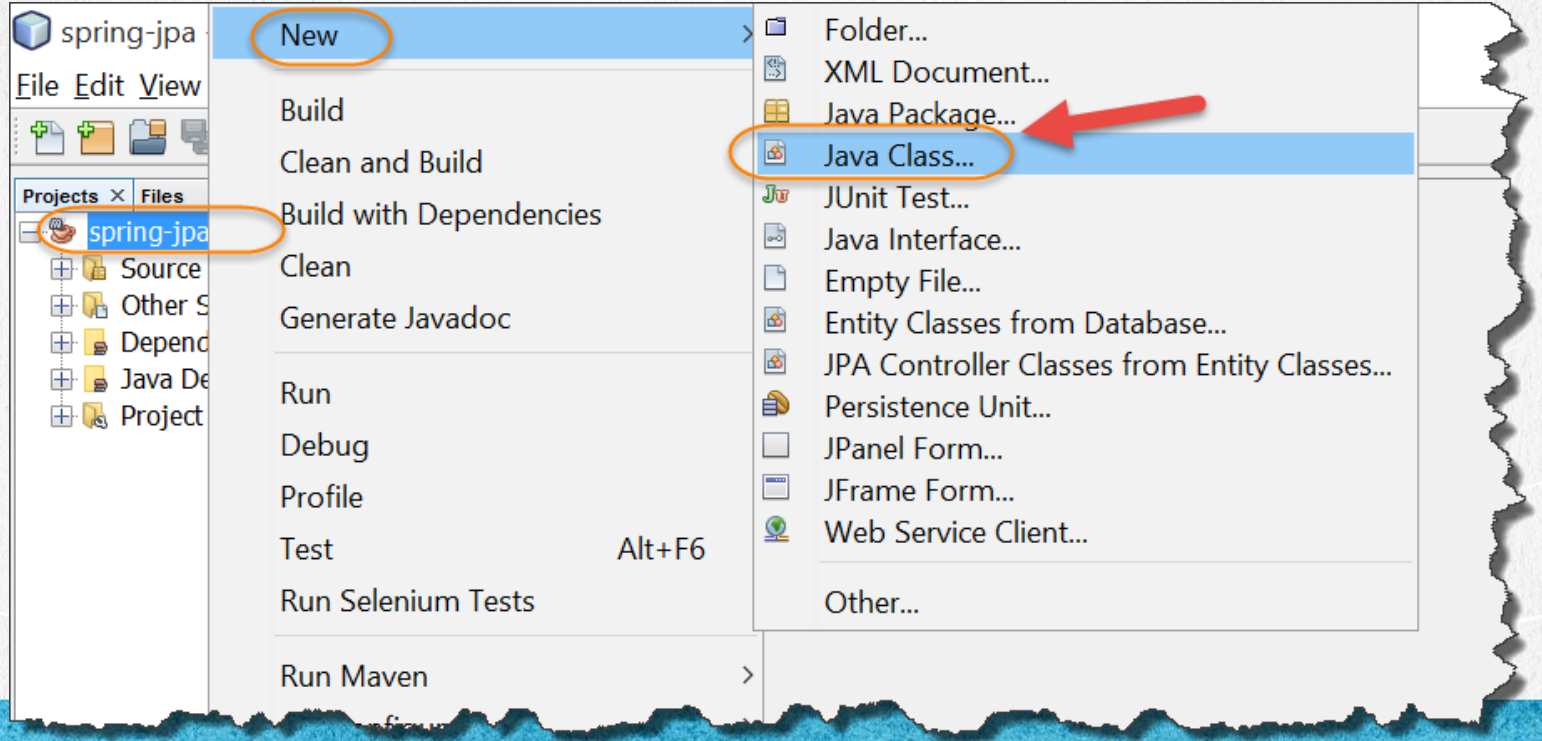
PersonDao.java:

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();
}
```

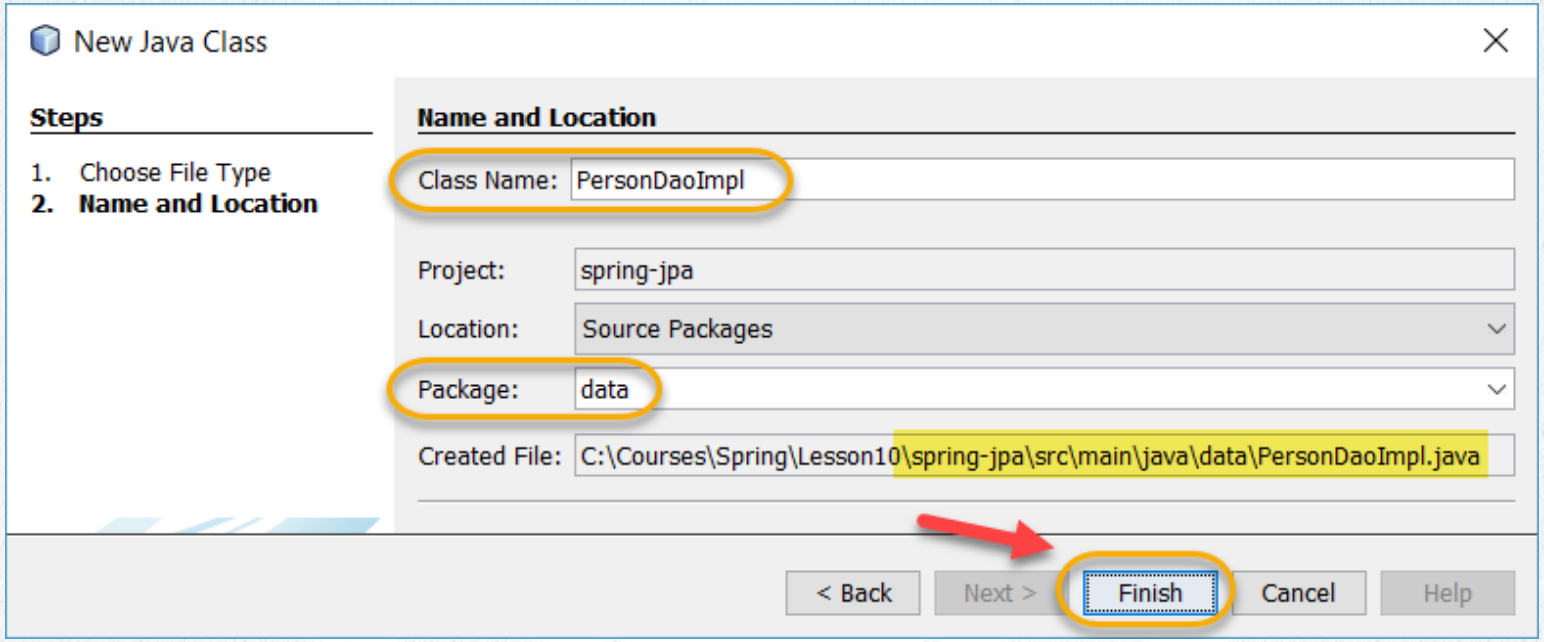
14. CREATE A NEW CLASS

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

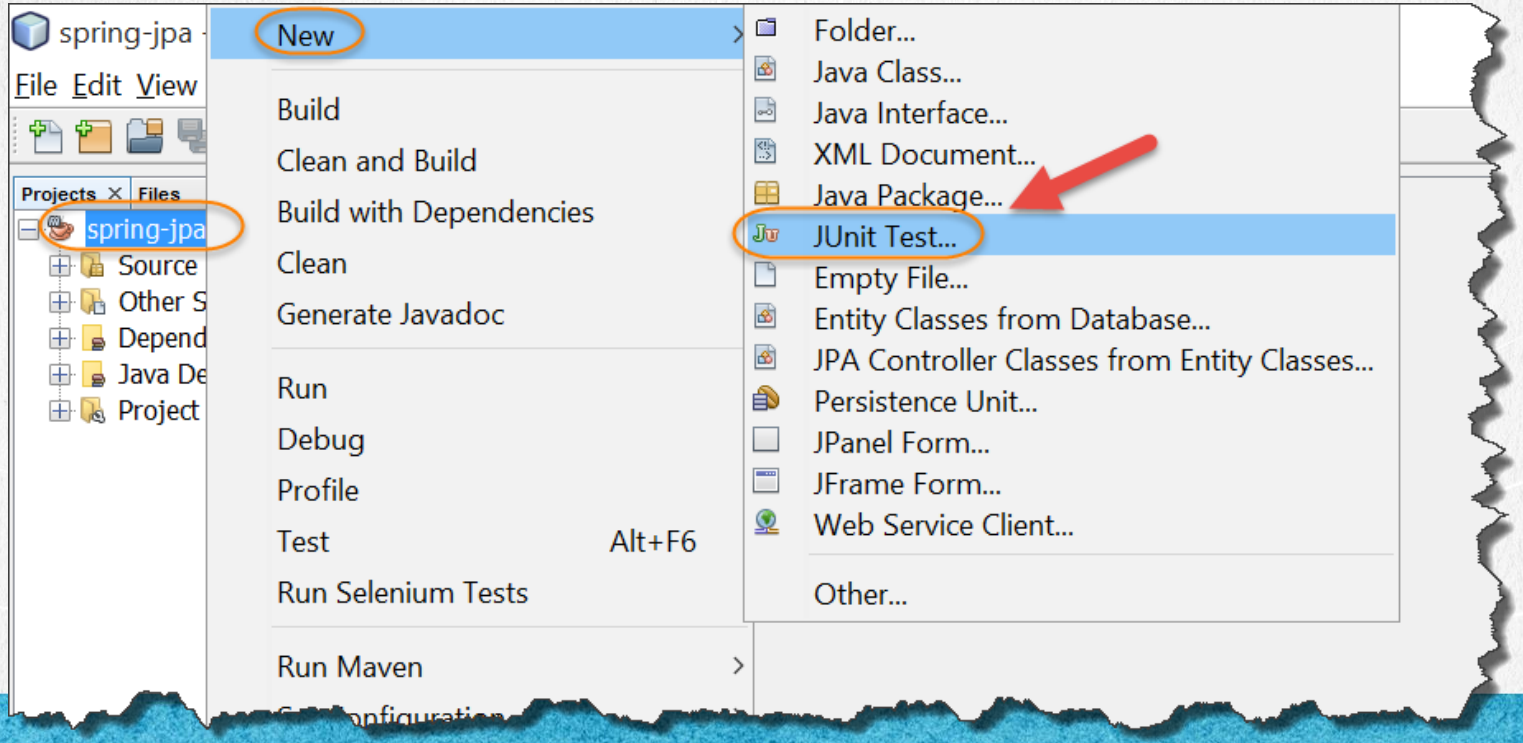
PersonDaoImpl.java:

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();
    }
}
```

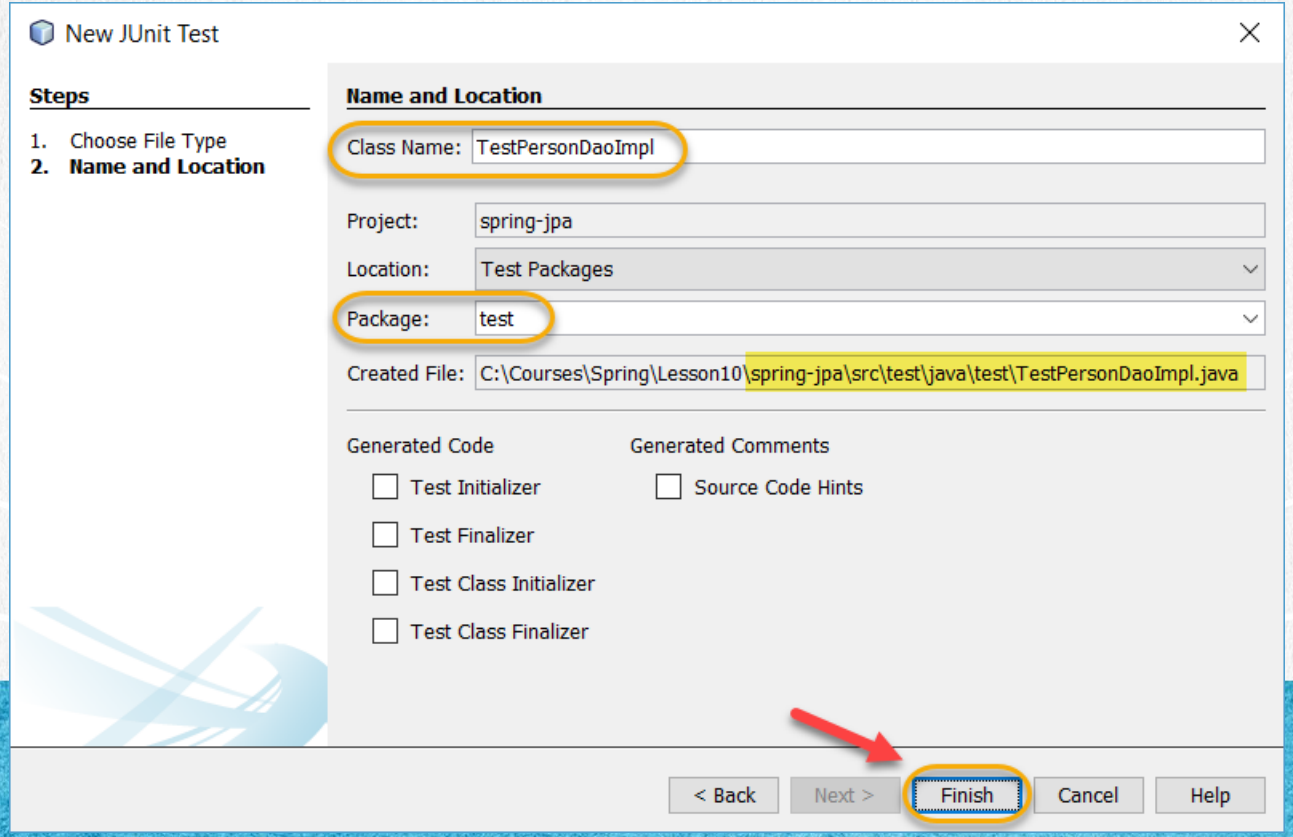
16. CREATE A NEW CLASS

We create the JUnit class called TestPersonDaoImpl.java:



16. CREATE A NEW CLASS

We create the JUnit class called TestPersonDaoImpl.java:



17. MODIFY THE CODE

TestPersonDaoImpl.java:

Click to download

```
package test;

import data.PersonDao;
import domain.Person;
import java.util.List;
import org.apache.logging.log4j.*;
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;

@ExtendWith(SpringExtension.class)
@ContextConfiguration(locations = {"classpath:applicationContext.xml"})
public class TestPersonDaoImpl {

    private final Logger logger = LogManager.getRootLogger();

    @Autowired
    private PersonDao personDao;
```

17. MODIFY THE CODE

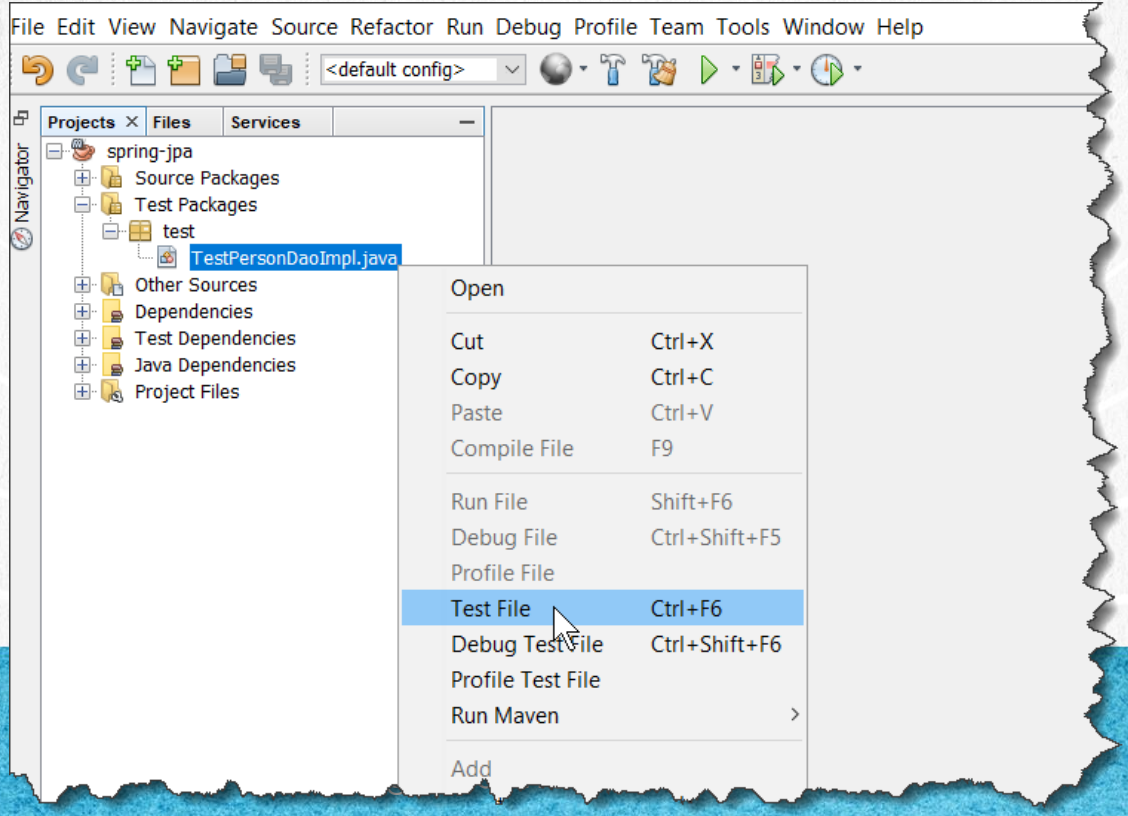
TestPersonDaoImpl.java:

Click to download

```
    @Test
    public void shouldShowPeople() {
        try {
            System.out.println();
            logger.info("Start of the test shouldShowPeople");

            List<Person> people = personDao.findAllPeople();

            int peopleCounter = 0;
            for (Person person : people) {
                logger.info("Person: " + person);
                peopleCounter++;
            }

            //According to the number of people recovered, it should be the same as the table
            assertEquals(peopleCounter, personDao.countPeople());

            logger.info("End of the test shouldShowPeople");
        } catch (Exception e) {
            logger.error("Error JBDC", e);
        }
    }
}
```

18. EXECUTE THE PROJECT

We execute the project. If no data in MySql can add data using MySql Workbench:



18. EXECUTE THE PROJECT

We execute the project. The result is similar to the following:

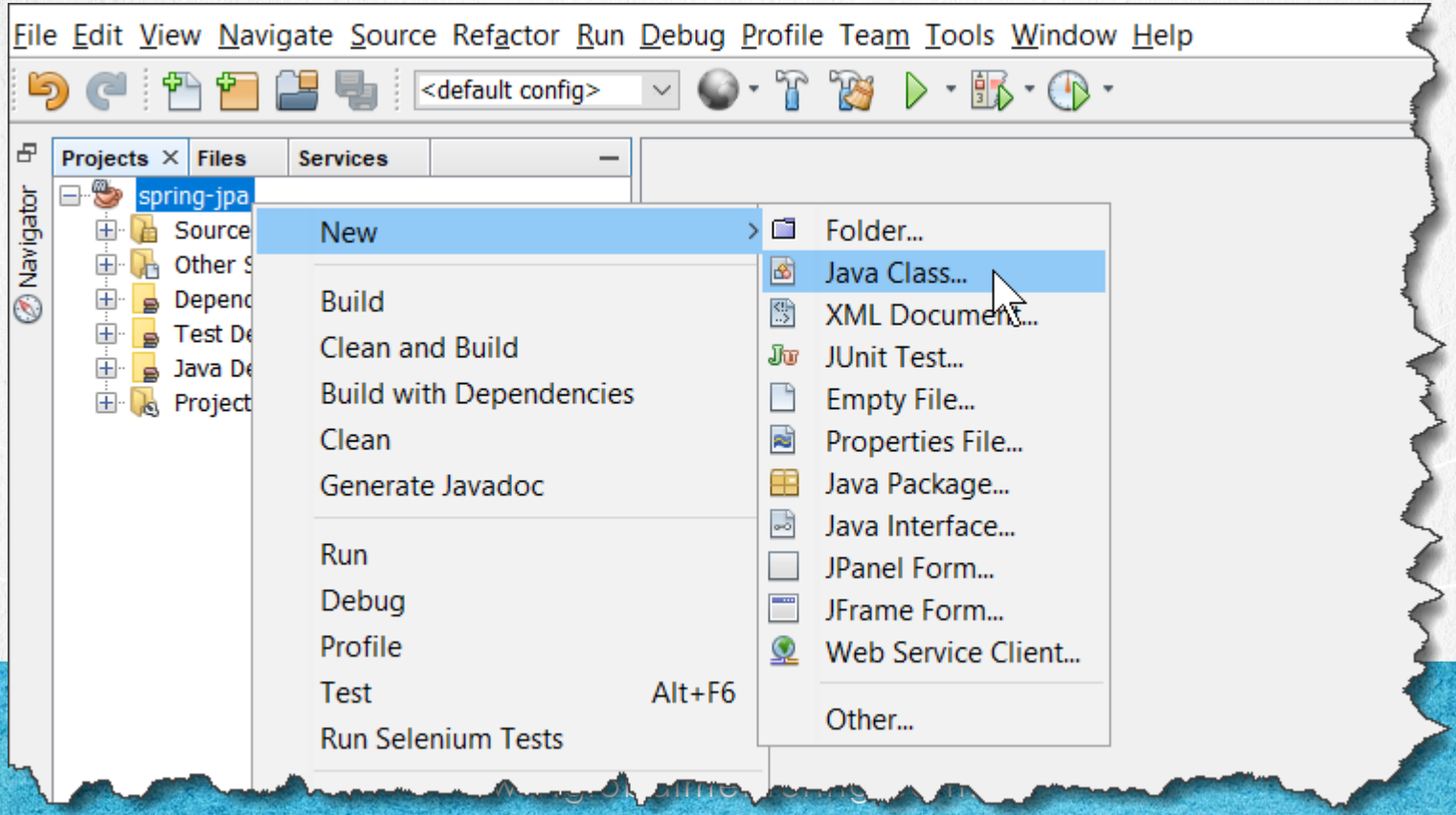
Output × Test Results

Retriever Output × Test (TestPersonDaoImpl) ×

```
[EL Info]: connection: ServerSession(695530017)--Thread(Thread[main,5,main])--/file:/C:/Courses/Spring/Lesson10/spring-jpa/target/classes/_P
[EL Fine]: sql: ServerSession(695530017)--Connection(49547843)--Thread(Thread[main,5,main])--SELECT id_person, NAME FROM PERSON
21:02:12 [main] INFO   - Person: Person{idPerson=1, name=John}
21:02:12 [main] INFO   - Person: Person{idPerson=2, name=Katty}
[EL Fine]: sql: ServerSession(695530017)--Connection(49547843)--Thread(Thread[main,5,main])--SELECT COUNT(id_person) FROM PERSON
21:02:12 [main] INFO   - End of the test shouldShowPeople
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.924 s - in test.TestPersonDaoImpl
21:02:12 [Thread-1] INFO  org.springframework.orm.jpa.LocalEntityManagerFactoryBean - Closing JPA EntityManagerFactory for persistence unit
[EL Config]: connection: ServerSession(695530017)--Connection(49547843)--Thread(Thread[Thread-1,5,main])--disconnect
[EL Info]: connection: ServerSession(695530017)--Thread(Thread[Thread-1,5,main])--/file:/C:/Courses/Spring/Lesson10/spring-jpa/target/classe
[EL Config]: connection: ServerSession(695530017)--Connection(600017090)--Thread(Thread[Thread-1,5,main])--disconnect

Results:
```

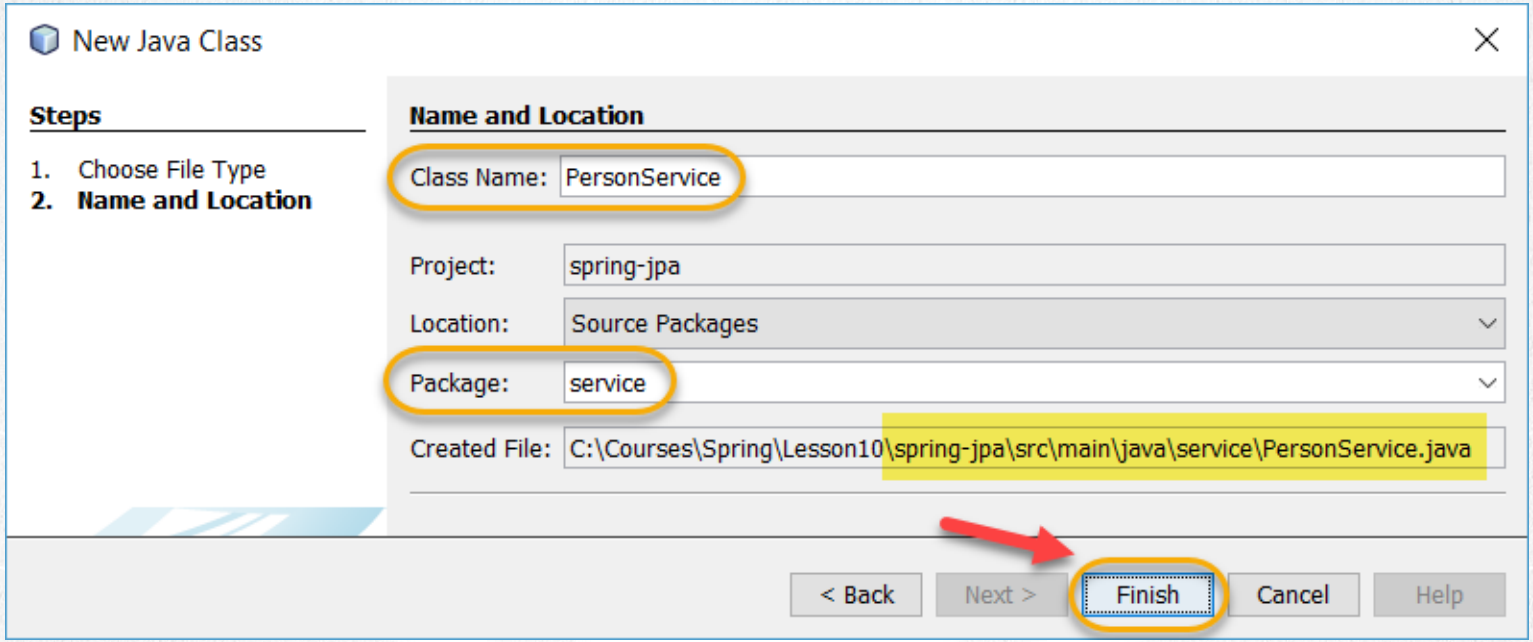
19. CREATE A NEW CLASS

Next we create the PersonService.java interface:



19. CREATE A NEW CLASS

Next we create the PersonService.java interface:



20. MODIFY THE CODE

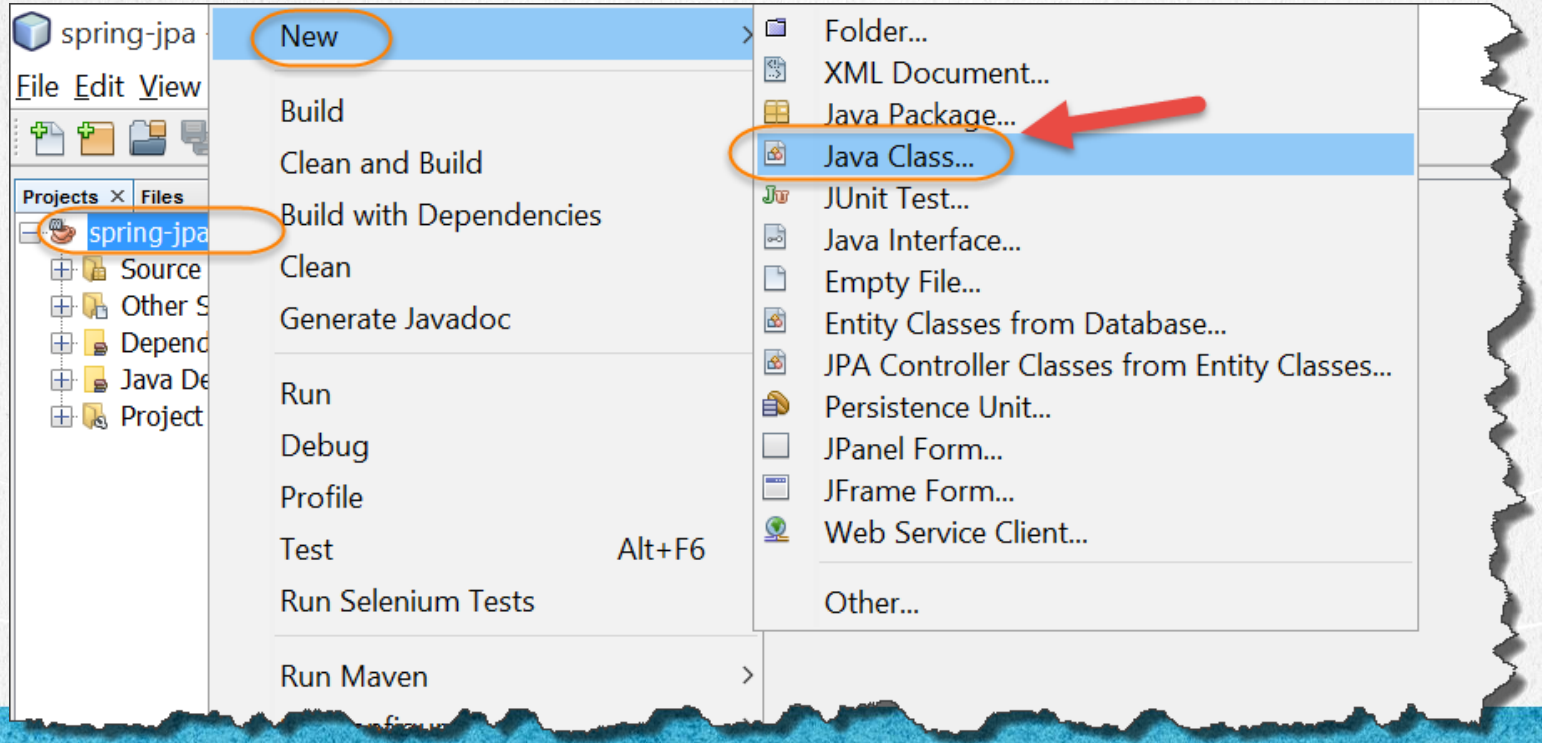
PersonService.java:

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);
}
```

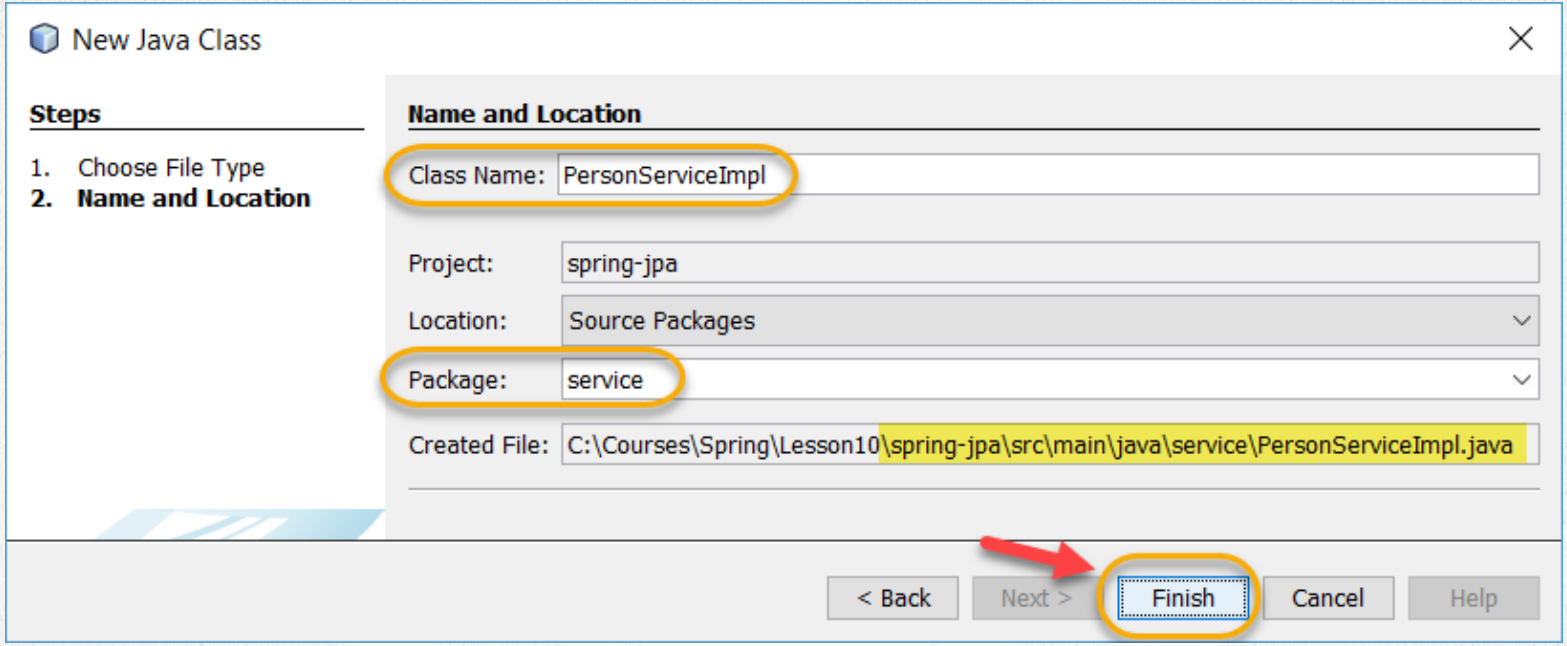
21. CREATE A NEW CLASS

We create the PersonServiceImpl.java class:



21. CREATE A NEW CLASS

We create the PersonServiceImpl.java class:



22. MODIFY THE CODE

PersonServiceImpl.java:

Click to download

```
package service;

import data.PersonDao;
import domain.Person;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.*;

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

    @Autowired
    private PersonDao personDao;

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

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

22. MODIFY THE CODE

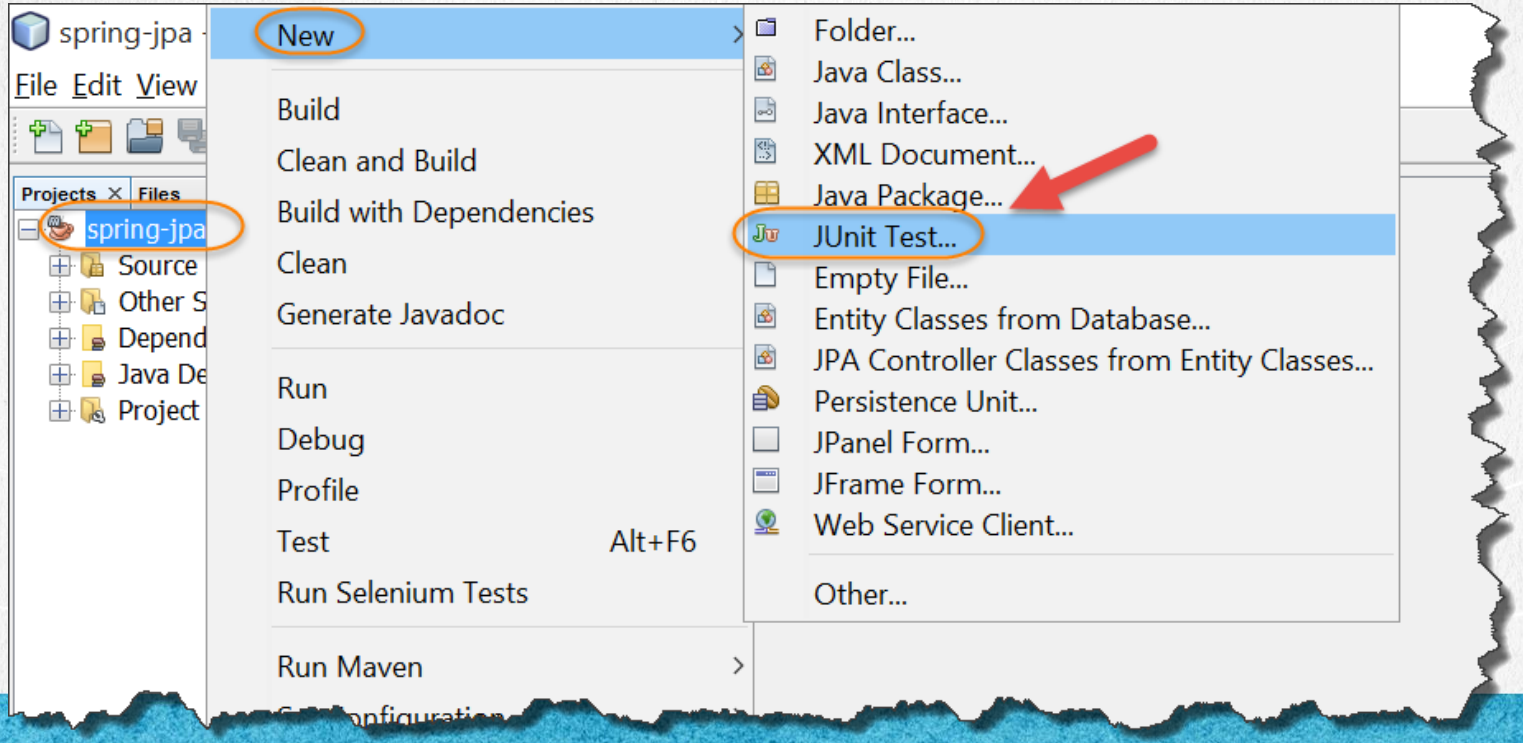
PersonServiceImpl.java:

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);
    }
}
```

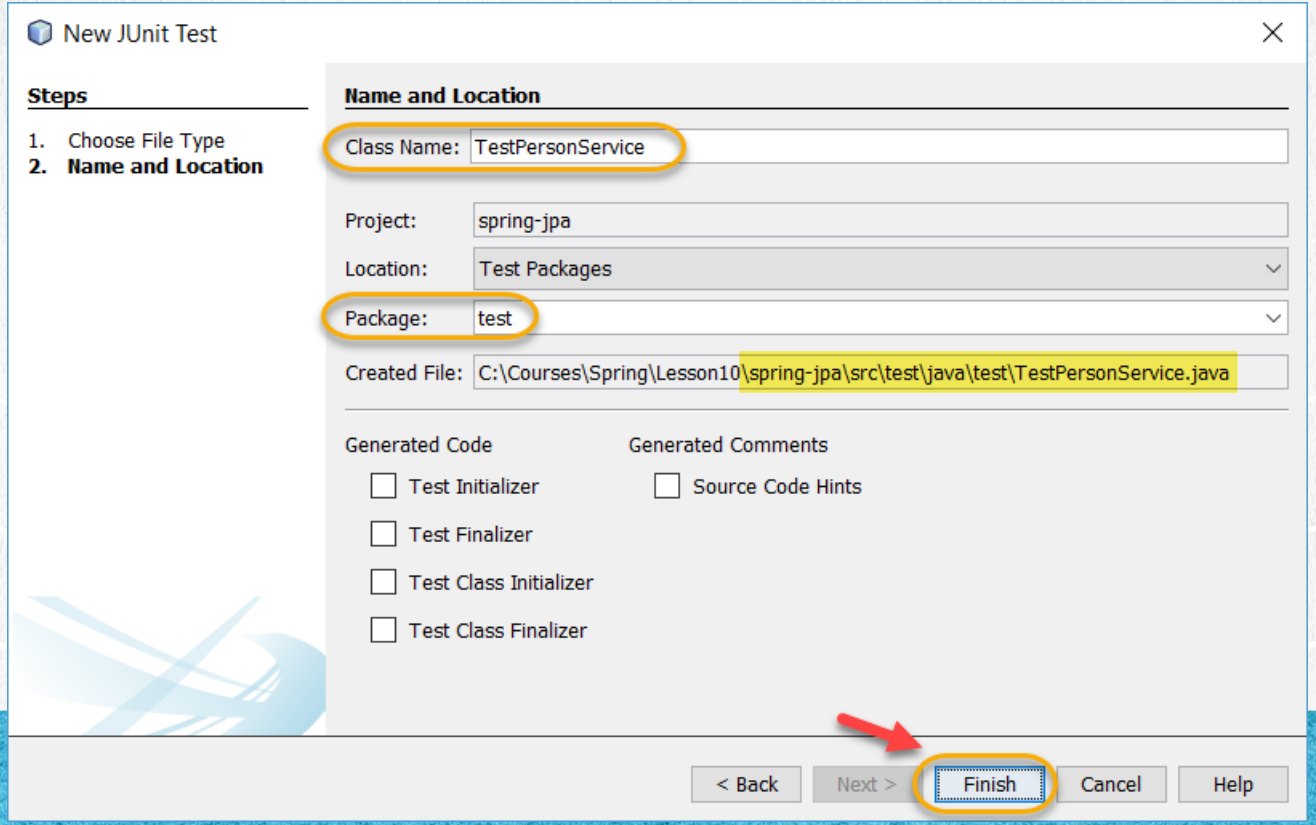
23. CREATE A NEW CLASS

We create the JUnit class called TestPersonService.java:



23. CREATE A NEW CLASS

We create the JUnit class called TestPersonService.java:



24. MODIFY THE CODE

TestPersonService.java:

Click to download

```
package test;

import domain.Person;
import java.util.List;
import org.apache.logging.log4j.*;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import service.PersonService;

@ExtendWith(SpringExtension.class)
@ContextConfiguration(locations = {"classpath:applicationContext.xml"})
public class TestPersonService {

    private final Logger logger = LogManager.getRootLogger();

    @Autowired
    private PersonService personService;
```

24. MODIFY THE CODE

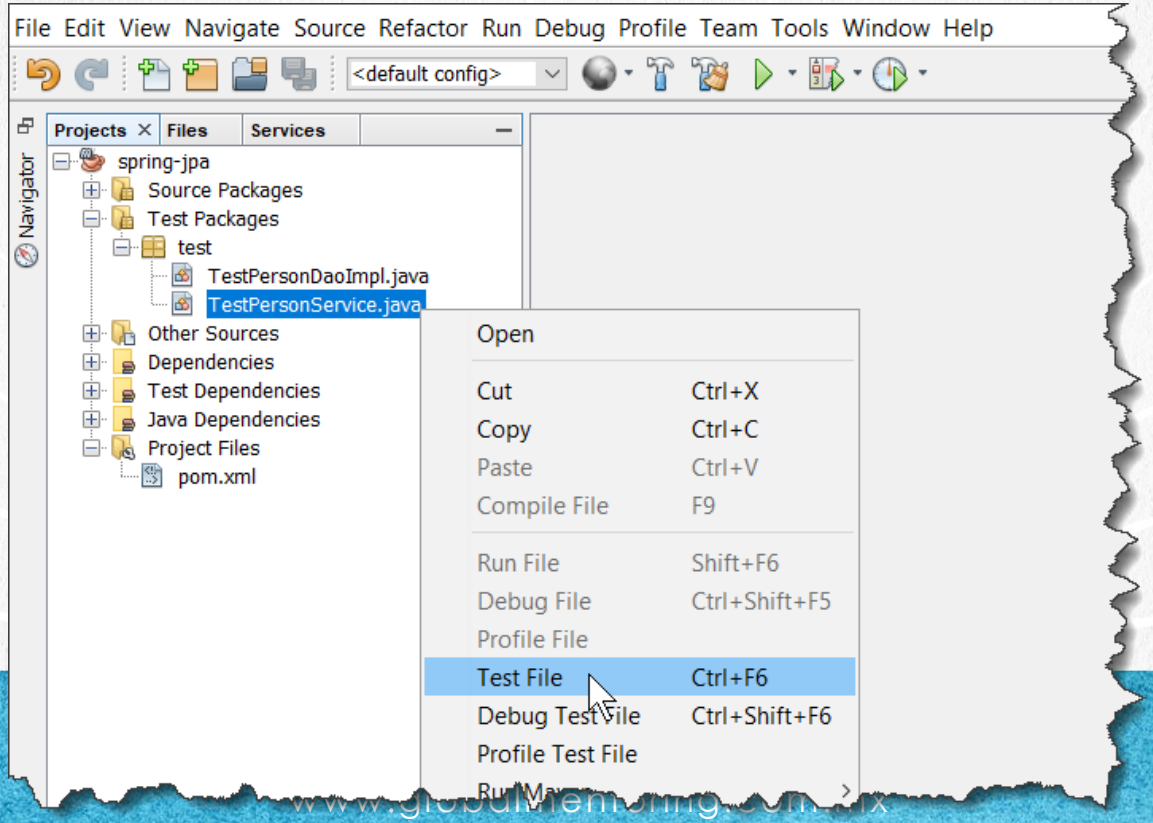
TestPersonService.java:

Click to download

```
    @Test
    public void testShoudShowPeople() {
        try {
            System.out.println();
            logger.info("Start of test testTransactions");
            logger.info("List People:");
            this.displayPeople();
            logger.info("End of test testTransactions");
            System.out.println();
        } catch (Exception e) {
            logger.error("Error Service: ", e);
        }
    }

    private int displayPeople() {
        List<Person> persons = personService.listPeople();
        int counPeople = 0;
        for (Person person : persons) {
            logger.info("Person: " + person);
            counPeople++;
        }
        return counPeople;
    }
}
```

25. EXECUTE THE PROJECT

We execute the project. If no data can add data using MySql Workbench:



25. EXECUTE THE PROJECT

We execute the project. The result is similar to the following:

Output × Test Results

Retriever Output × Test (TestPersonService) ×

```
[EL Fine]: sql: ClientSession(1878225987)--Connection(981159997)--Thread(Thread[main,5,main])--SELECT id_person, NAME FROM PERSON
21:21:33 [main] INFO   - Person: Person{idPerson=1, name=John}
21:21:33 [main] INFO   - Person: Person{idPerson=2, name=Katty}
21:21:33 [main] INFO   - End of test testTransactions

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.961 s - in test.TestPersonService
21:21:33 [Thread-1] INFO  org.springframework.orm.jpa.LocalEntityManagerFactoryBean - Closing JPA EntityManagerFactory for persistence unit 'PersistenceUnit'
[EL Config]: connection: ServerSession(632475595)--Connection(981159997)--Thread(Thread[Thread-1,5,main])--disconnect
[EL Info]: connection: ServerSession(632475595)--Thread(Thread[Thread-1,5,main])--/file:/C:/Courses/Spring/Lesson10/spring-jpa/target/classes/_PersistenceUnit
[EL Config]: connection: ServerSession(632475595)--Connection(913746983)--Thread(Thread[Thread-1,5,main])--disconnect

Results:
```

EXERCISE CONCLUSION

With this exercise we have made the integration of Spring and JPA.

We also create the data layer and the service layer of our project. So up to this point we can now create a presentation layer with frameworks like Servlets / JSPs, JSF, Spring MVC or any other technology for the presentation layer.

We create several configuration files, since each of the technologies requires a file in order to work correctly, and we join the technologies from the applicationContext.xml file.

Finally, we created two unit tests to test both the data layer, as well as the service layer of our project. With this we are ready to create one or more presentation layers, as required.

ONLINE COURSE

SPRING FRAMEWORK

By: Eng. Ubaldo Acosta