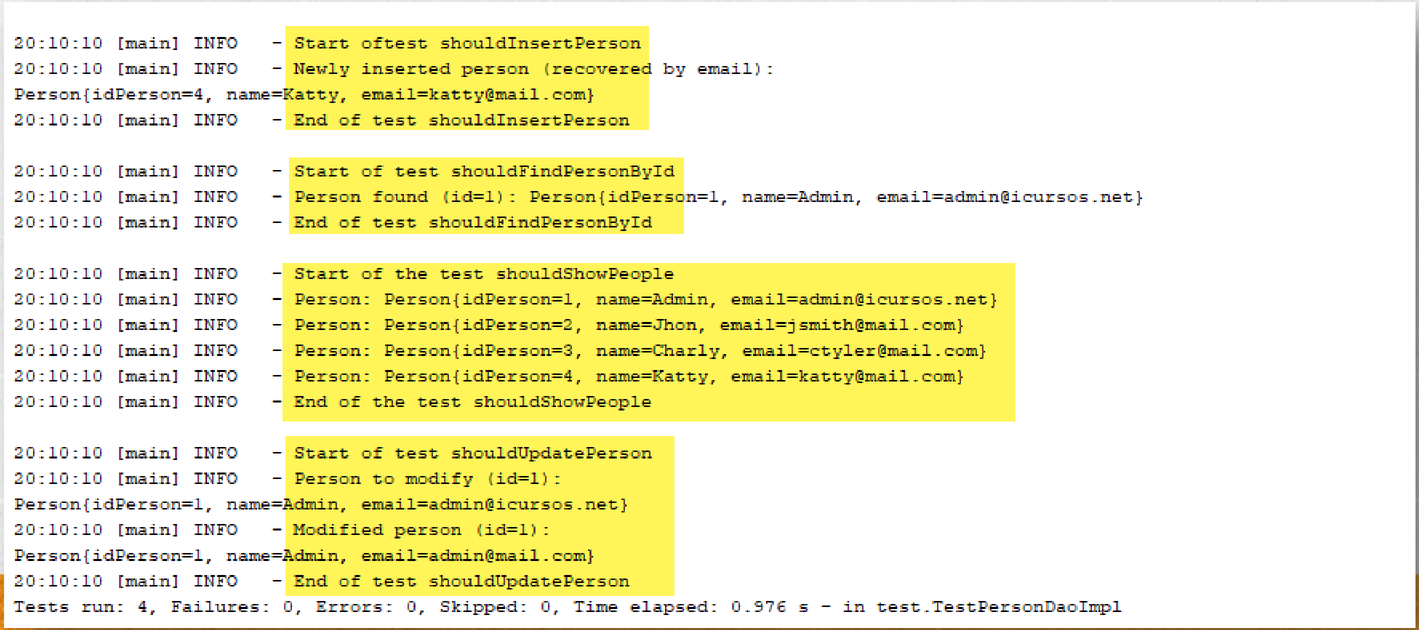
SPRING FRAMEWORK COURSE

EXERCISE

QUERIES WITH SPRING JDBC

EXERCISE OBJECTIVE

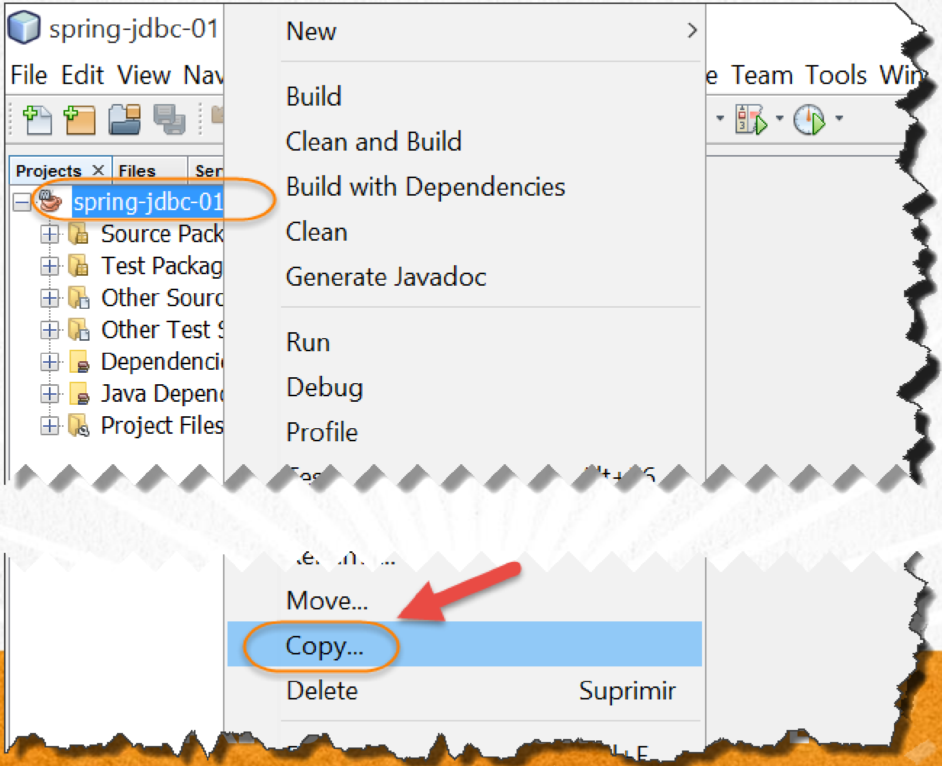
- The objective of the exercise is to add the cases of high, low, changes and selection of data from the table people to our Spring JDBC project. At the end we will see the following output:



```
20:10:10 [main] INFO   - Start oftest shouldInsertPerson
20:10:10 [main] INFO   - Newly inserted person (recovered by email):
Person{idPerson=4, name=Katty, email=katty@mail.com}
20:10:10 [main] INFO   - End of test shouldInsertPerson

20:10:10 [main] INFO   - Start of test shouldFindPersonById
20:10:10 [main] INFO   - Person found (id=1): Person{idPerson=1, name=Admin, email=admin@icursos.net}
20:10:10 [main] INFO   - End of test shouldFindPersonById

20:10:10 [main] INFO   - Start of the test shouldShowPeople
20:10:10 [main] INFO   - Person: Person{idPerson=1, name=Admin, email=admin@icursos.net}
20:10:10 [main] INFO   - Person: Person{idPerson=2, name=Jhon, email=jsmith@mail.com}
20:10:10 [main] INFO   - Person: Person{idPerson=3, name=Charly, email=ctyler@mail.com}
20:10:10 [main] INFO   - Person: Person{idPerson=4, name=Katty, email=katty@mail.com}
20:10:10 [main] INFO   - End of the test shouldShowPeople

20:10:10 [main] INFO   - Start of test shouldUpdatePerson
20:10:10 [main] INFO   - Person to modify (id=1):
Person{idPerson=1, name=Admin, email=admin@icursos.net}
20:10:10 [main] INFO   - Modified person (id=1):
Person{idPerson=1, name=Admin, email=admin@mail.com}
20:10:10 [main] INFO   - End of test shouldUpdatePerson
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.976 s - in test.TestPersonDaoImpl
```

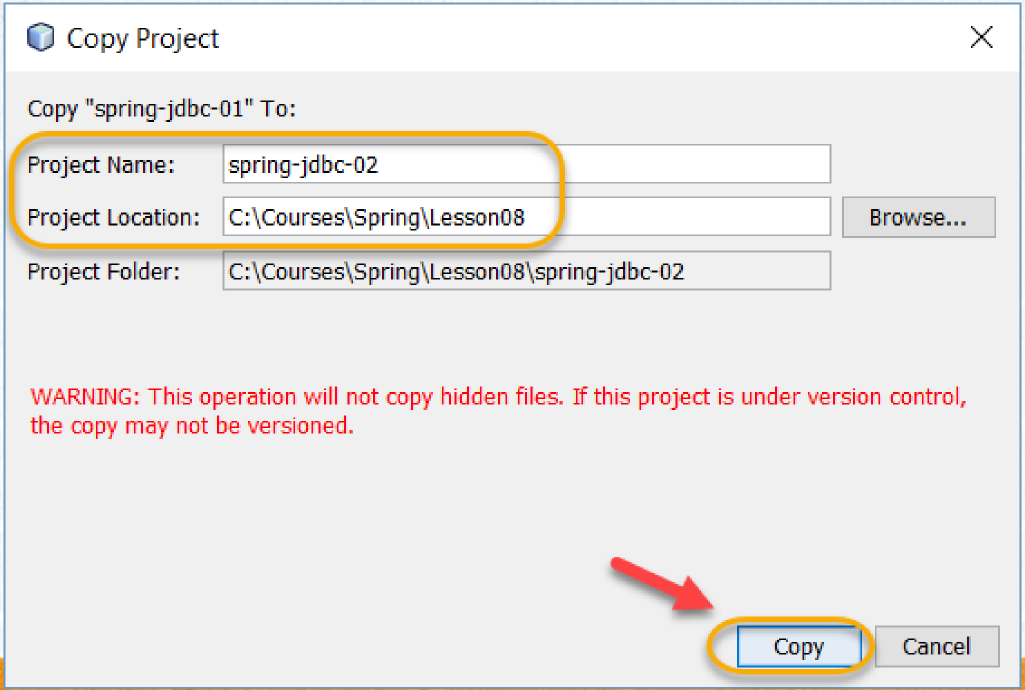
1. COPY THE PROJECT

We copy the project spring-jdbc-01:



1. COPY THE PROJECT

We copy the project spring-jdbc-01:



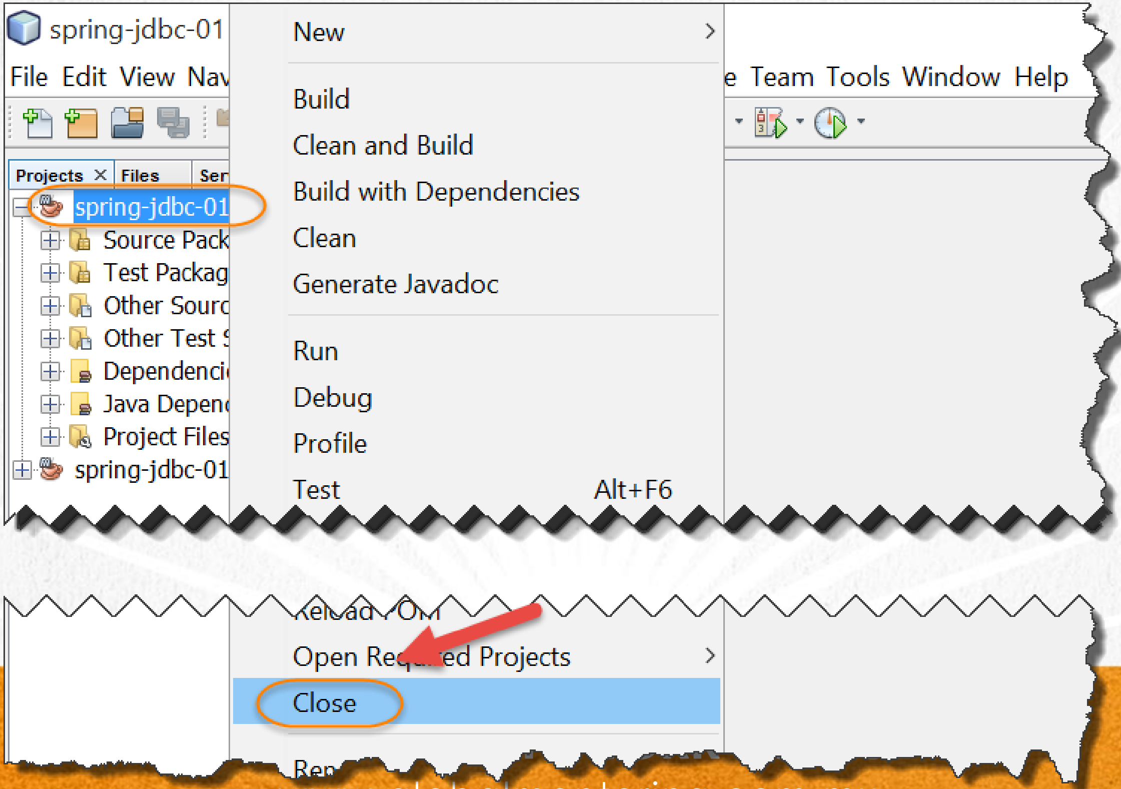
Copy "spring-jdbc-01" To:

Project Name:	spring-jdbc-02
Project Location:	C:\Courses\Spring\Lesson08
Project Folder:	C:\Courses\Spring\Lesson08\spring-jdbc-02

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

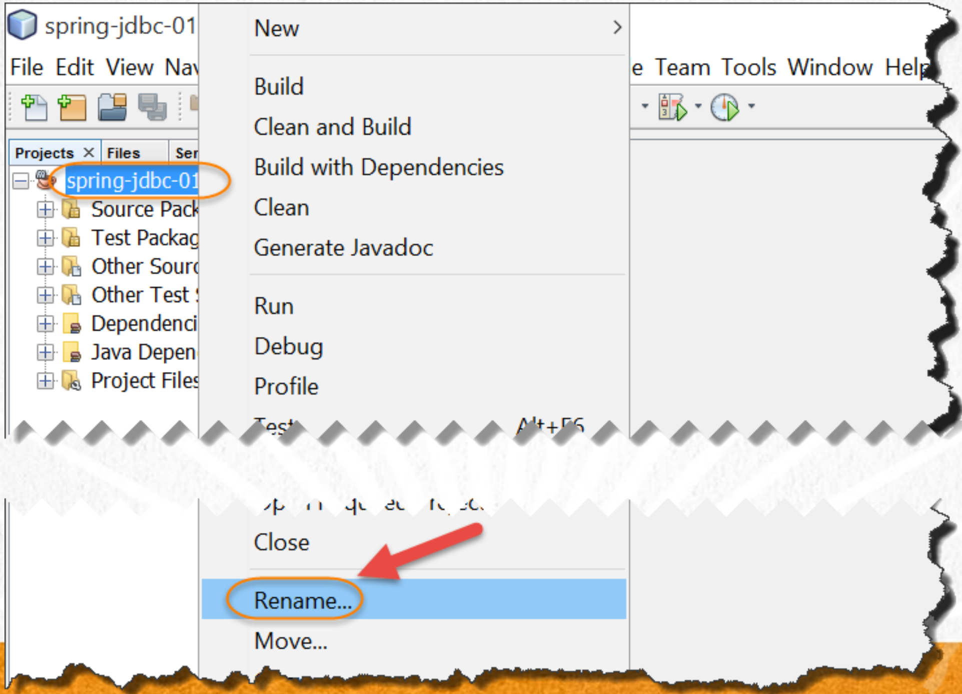
2. CLOSE THE PROJECT

We closed the previous project and we are left with the new one:



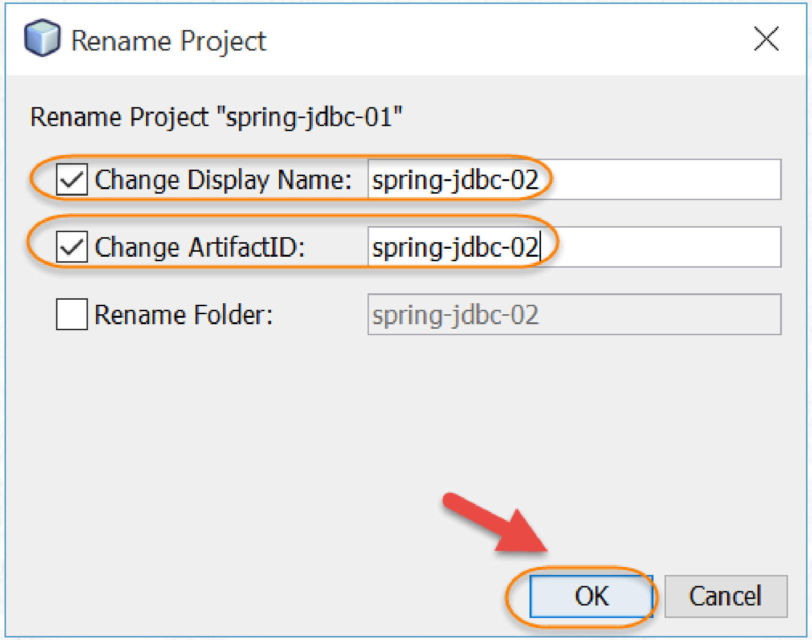
3. RENAME THE PROJECT

Rename the Project to spring-jdbc-02:



3. RENAME THE PROJECT

Rename the Project to spring-jdbc-02:



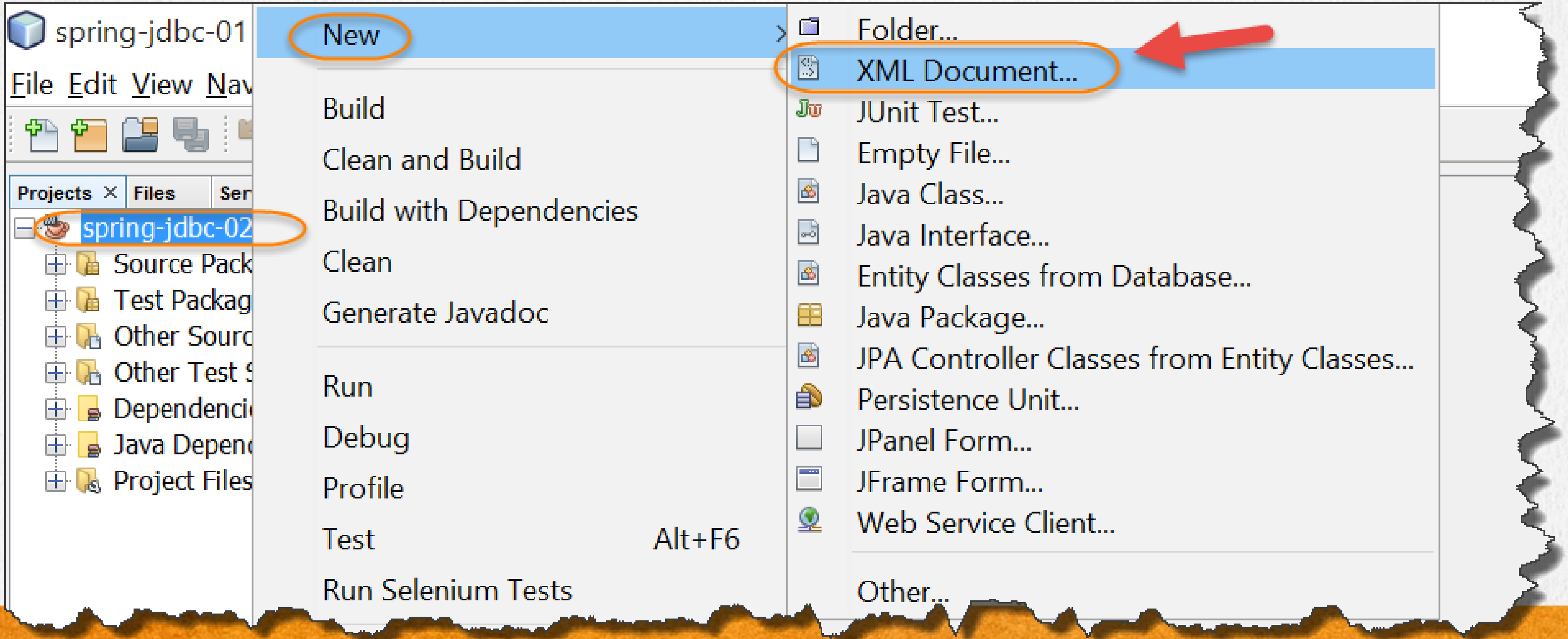
Rename Project

Rename Project "spring-jdbc-01"

- [✓] Change Display Name: spring-jdbc-02
- [✓] Change ArtifactID: spring-jdbc-02
- [ ] Rename Folder: spring-jdbc-02

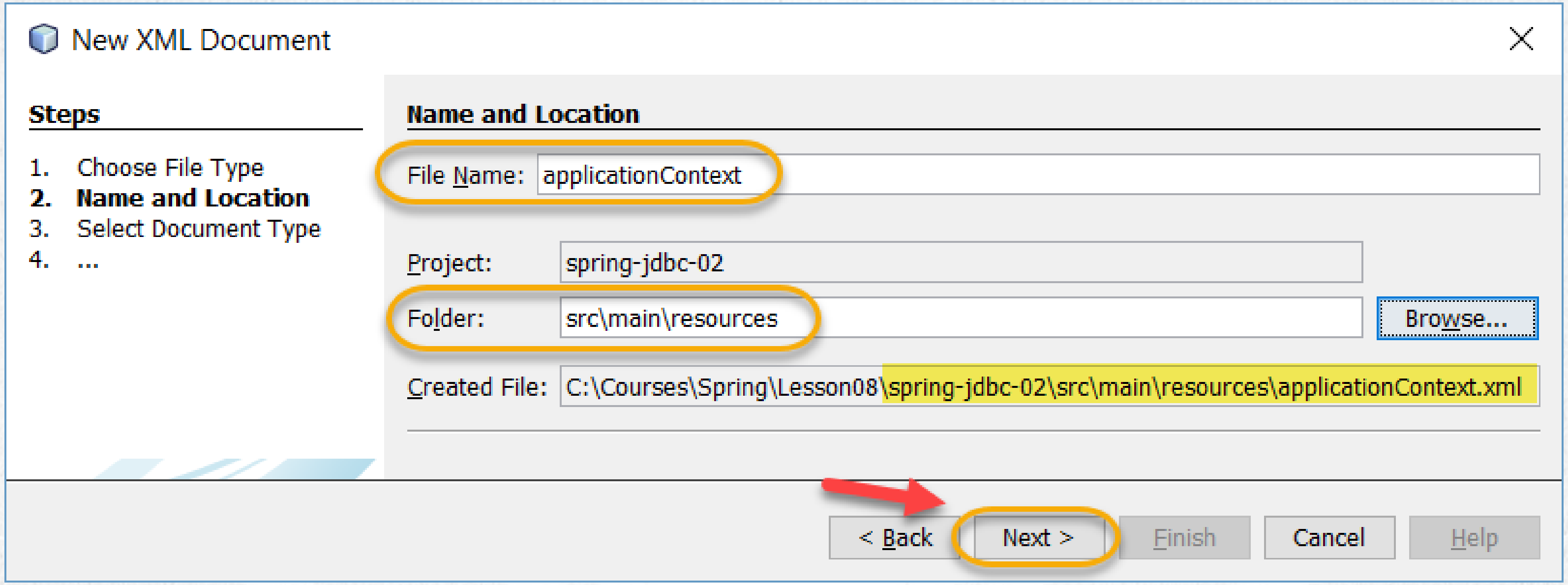
4. CREATE AN XML FILE

Create the applicationContext.xml file:



4. CREATE AN XML FILE

Create the applicationContext.xml file:



File Name: applicationContext

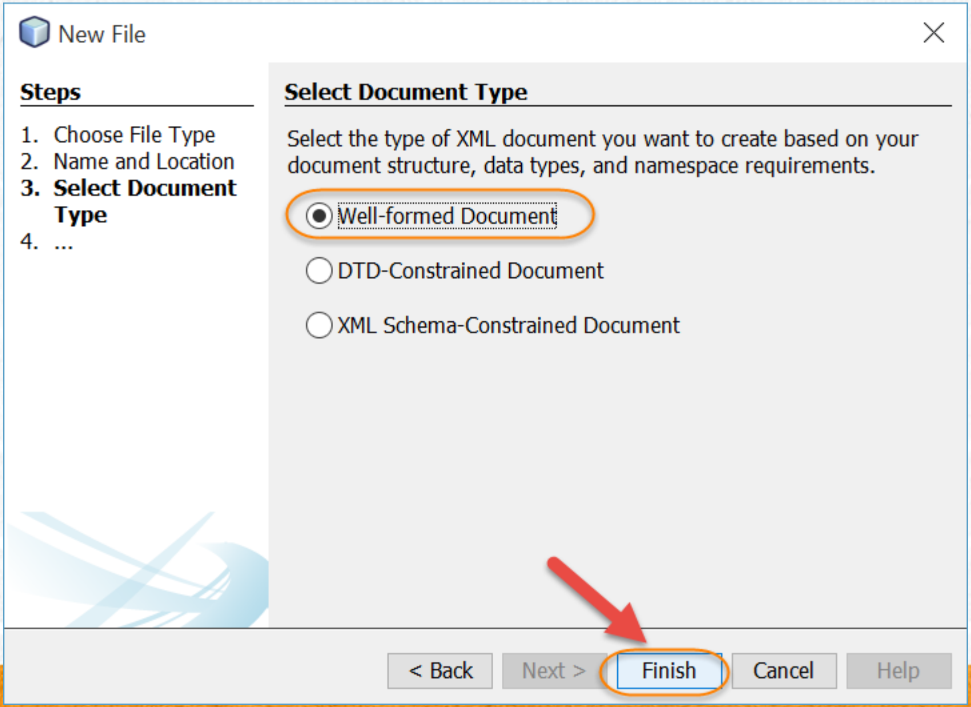
Project: spring-jdbc-02

Folder: src\main\resources

Created File: C:\Courses\Spring\Lesson08\spring-jdbc-02\src\main\resources\applicationContext.xml

4. CREATE AN XML FILE

Create the applicationContext.xml file:



New File

Steps

1. Choose File Type
2. Name and Location
3. **Select Document Type**
4. ...

Select Document Type

Select the type of XML document you want to create based on your document structure, data types, and namespace requirements.

- [✓ Well-formed Document]
- DTD-Constrained Document
- XML Schema-Constrained Document

5. MODIFY THE CODE

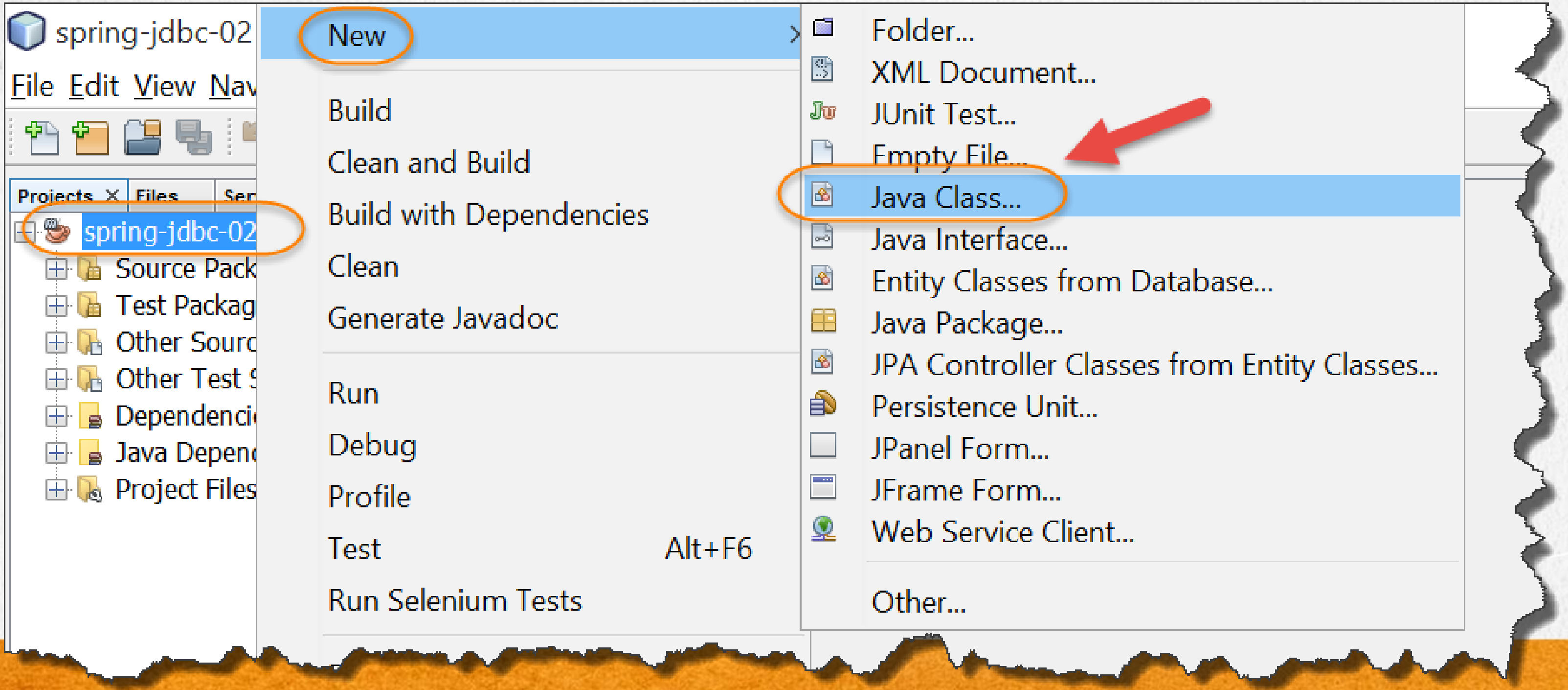
applicationContext.xml:

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"
       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">

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

</beans>
```

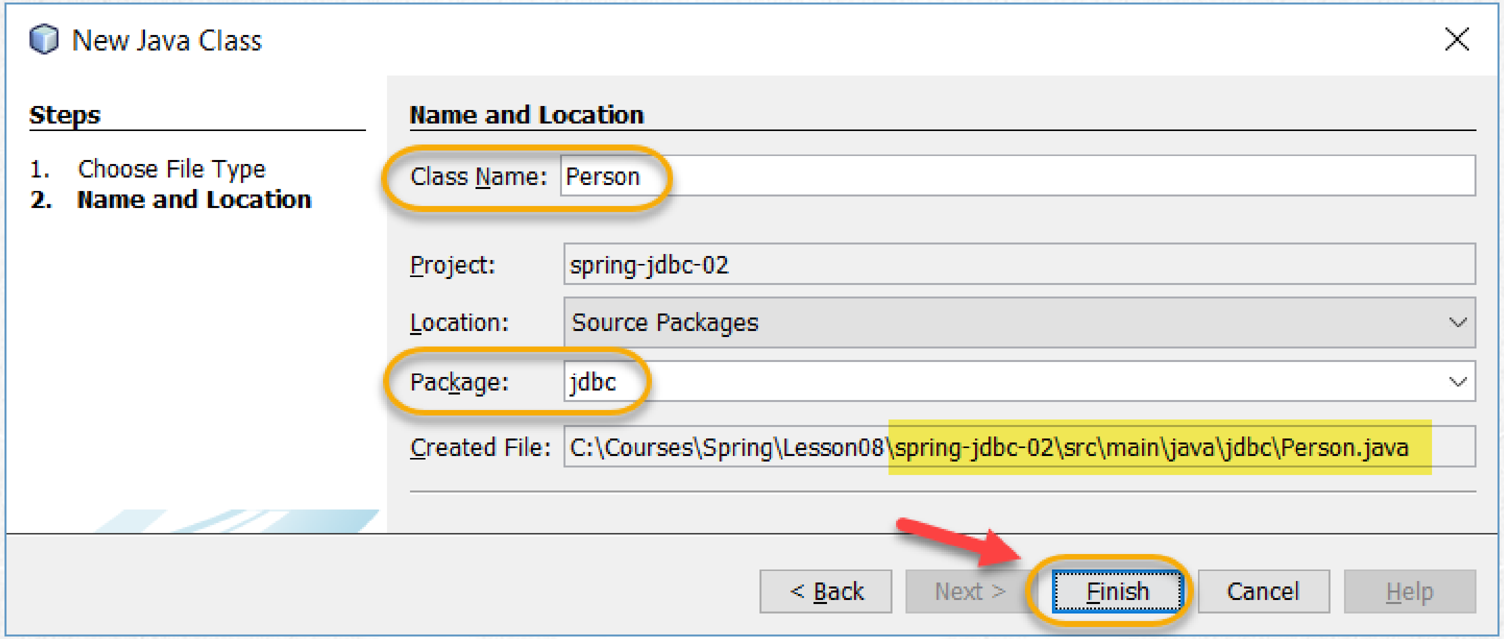
6. CREATE A NEW CLASS

We created the Person.java file:



6. CREATE A NEW CLASS

We created the Person.java file:



New Java Class

Steps

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

Name and Location

Class Name: Person

Project: spring-jdbc-02

Location: Source Packages

Package: jdbc

Created File: C:\Courses\Spring\Lesson08\spring-jdbc-02\src\main\java\jdbc\Person.java

7. MODIFY THE CODE

Person.java:

Click to download

```
package jdbc;

public class Person {

    private int idPerson;
    private String name;
    private String email;

    public Person() {
    }

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

    public int getIdPerson() {
        return idPerson;
    }

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

    public String getName() {
        return name;
    }
```

7. MODIFY THE CODE

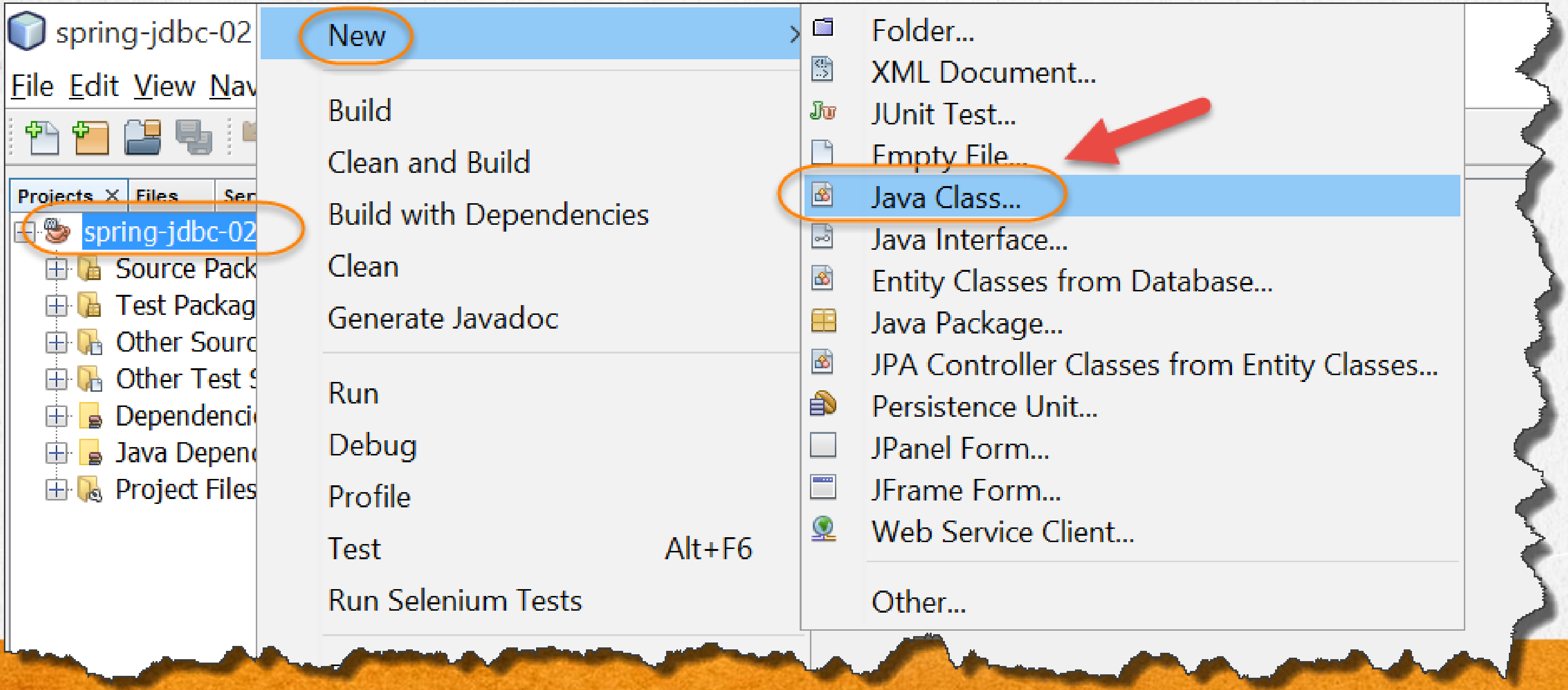
Person.java:

Click to download

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

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

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

}
```

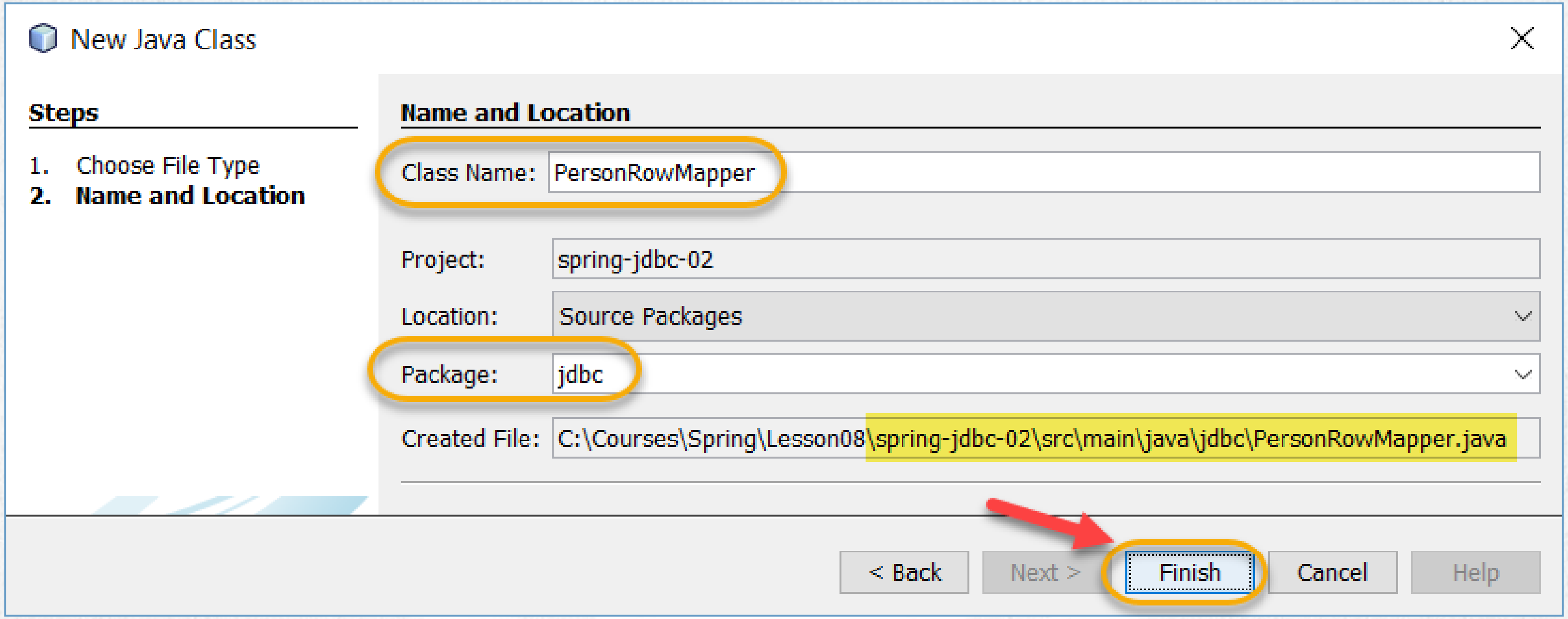
8. CREATE A NEW CLASS

We create the PersonRowMapper.java class:



8. CREATE A NEW CLASS

We create the PersonRowMapper.java class:



Class Name: PersonRowMapper

Project: spring-jdbc-02

Location: Source Packages

Package: jdbc

Created File: C:\Courses\Spring\Lesson08\spring-jdbc-02\src\main\java\jdbc\PersonRowMapper.java

9. MODIFY THE CODE

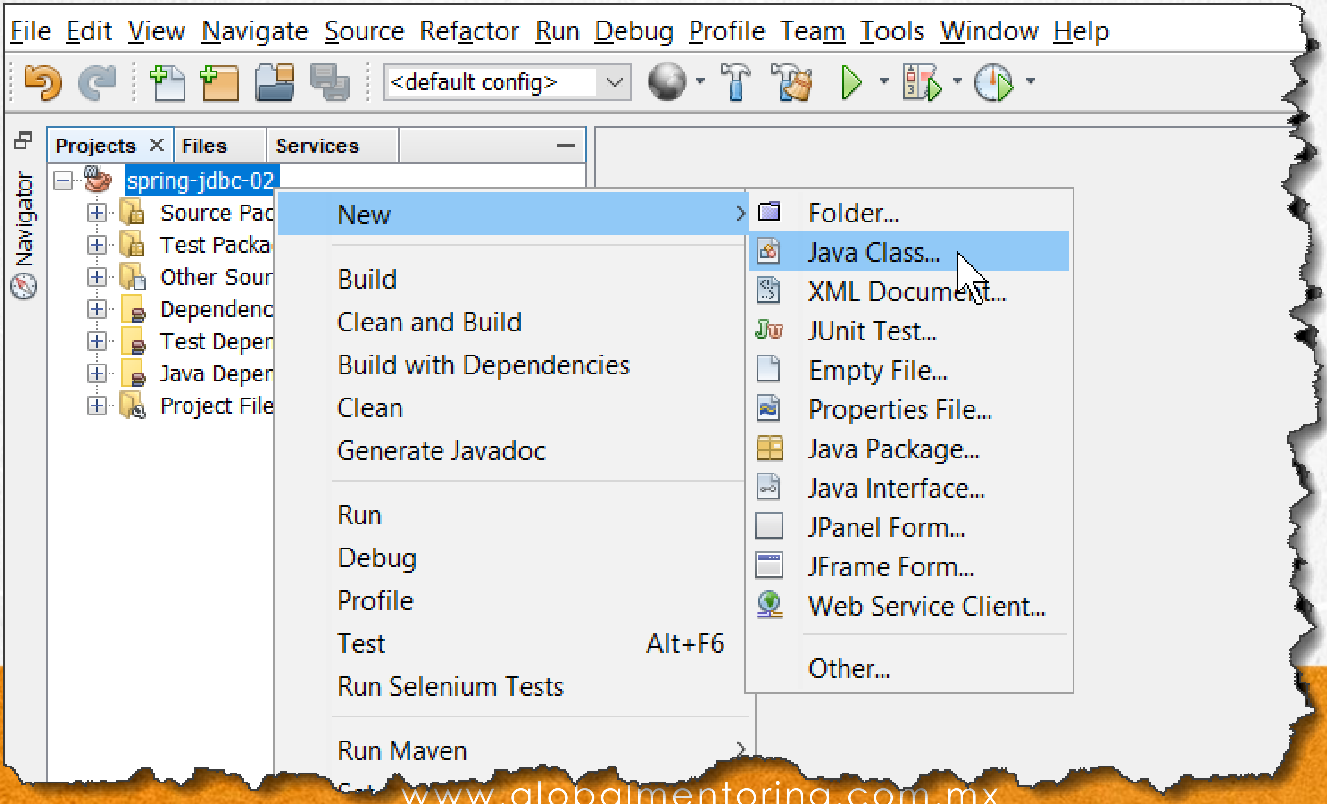
PersonRowMapper.java:

Click to download

```
package jdbc;

import java.sql.*;
import org.springframework.jdbc.core.RowMapper;

public class PersonRowMapper implements RowMapper<Person> {

    // Method that is called by the Spring template. This is a callback method
    @Override
    public Person mapRow(ResultSet rs, int rowNum) throws SQLException {
        //Creation of the person object for each record found in the resultSet
        Person person = new Person();
        person.setIdPerson(rs.getInt("id_person"));
        person.setName(rs.getString("name"));
        person.setEmail(rs.getString("email"));
        return person;
    }
}
```

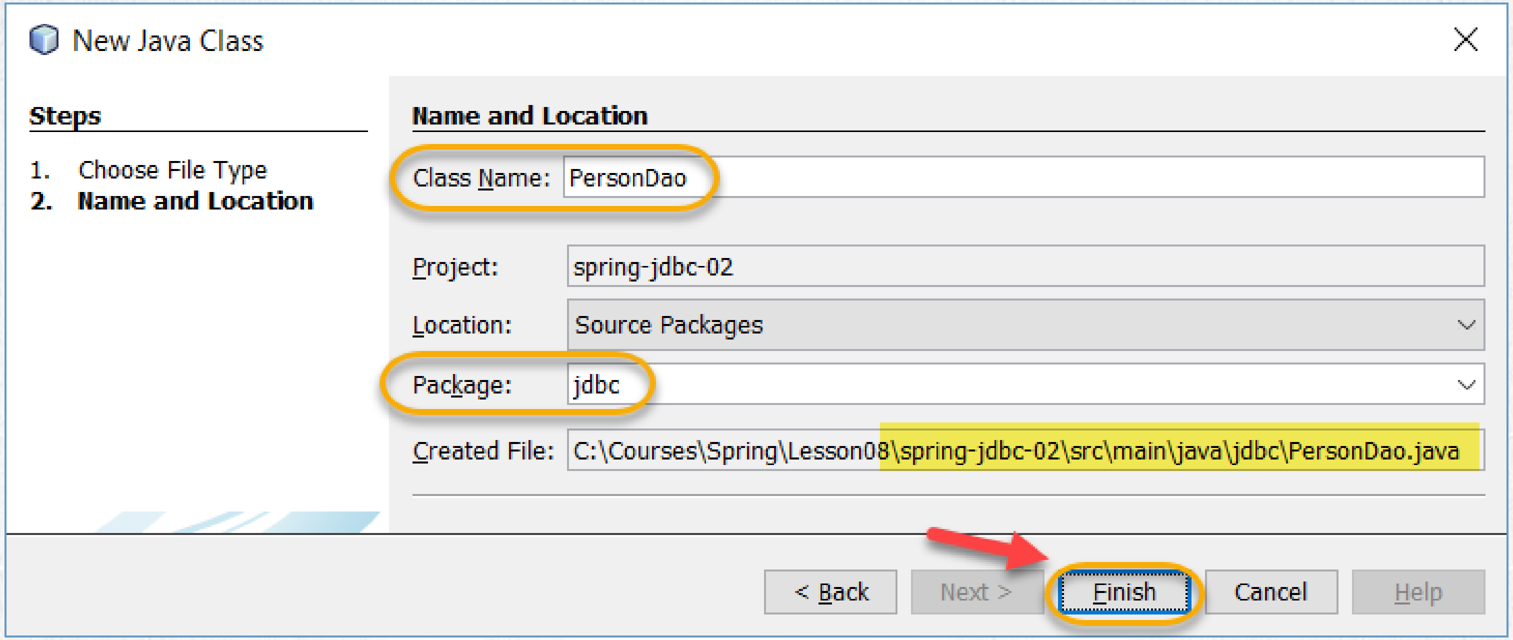
10. CREATE A JAVA CLASS

Create the PersonDao.java interface:



10. CREATE A JAVA CLASS

Create the PersonDao.java interface:


New Java Class

Steps

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

Name and Location

Class Name: PersonDao

Project: spring-jdbc-02

Location: Source Packages

Package: jdbc

Created File: C:\Courses\Spring\Lesson08\spring-jdbc-02\src\main\java\jdbc\PersonDao.java

11. MODIFY THE CODE

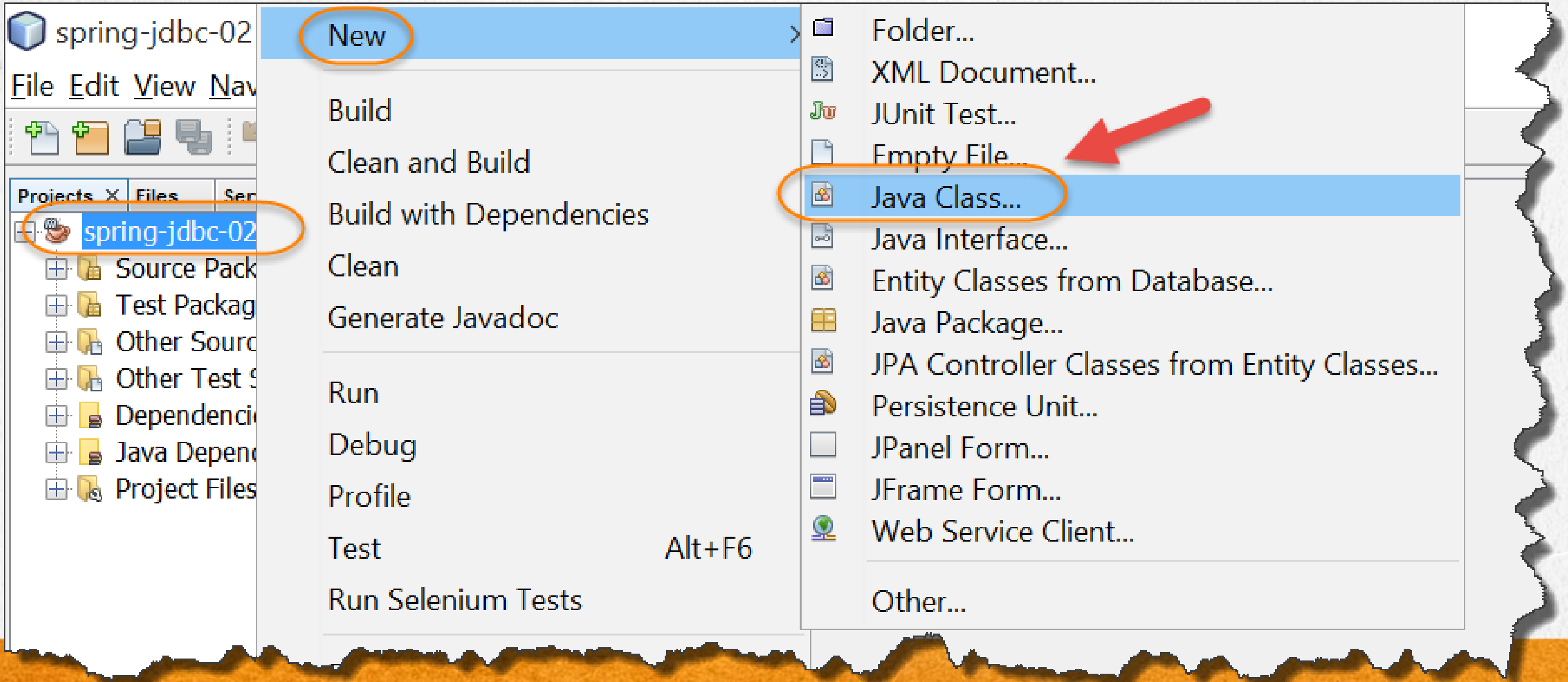
PersonDao.java:

Click to download

```
package jdbc;

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();

    int countPeople();

    Person getPersonByEmail(Person person);
}
```

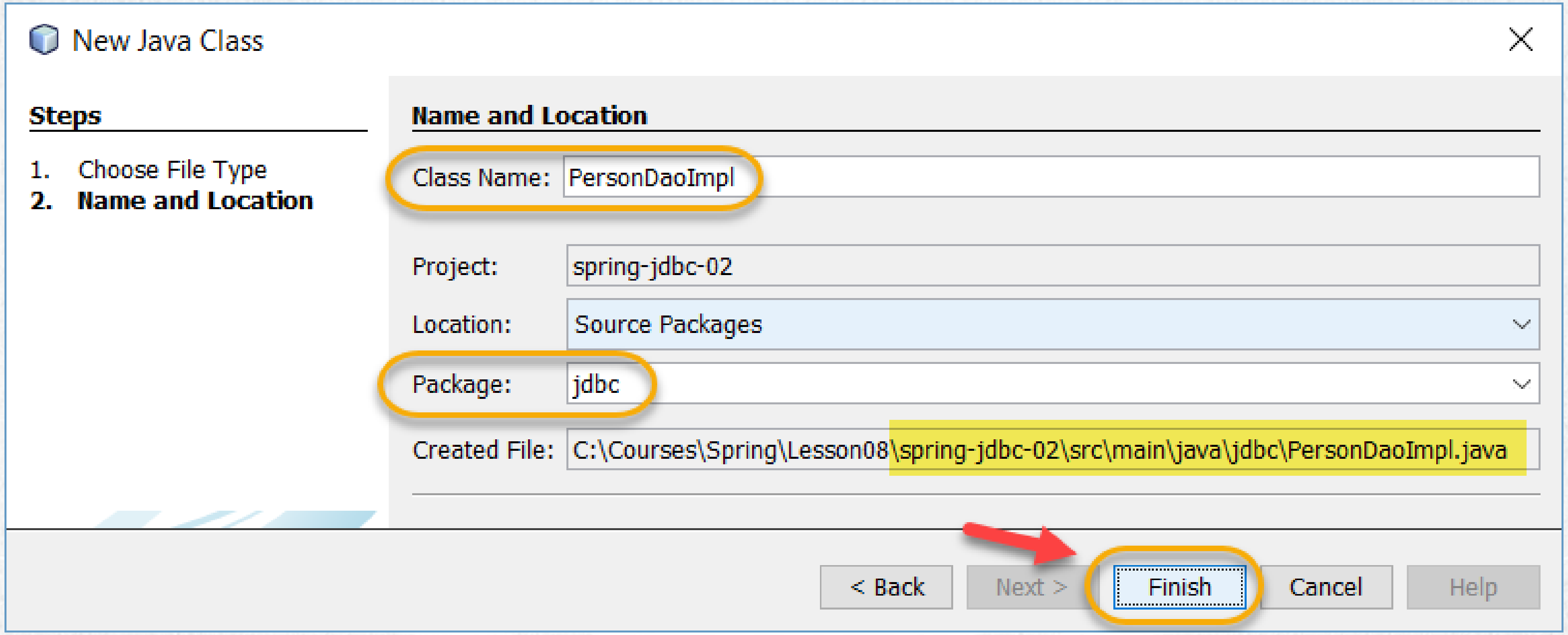
12. CREATE A NEW CLASS

We create the PersonDaoImpl.java class:



12. CREATE A NEW CLASS

We create the PersonDaoImpl.java class:



New Java Class

Steps

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

Name and Location

Class Name: PersonDaoImpl

Project: spring-jdbc-02

Location: Source Packages

Package: jdbc

Created File: C:\Courses\Spring\Lesson08\spring-jdbc-02\src\main\java\jdbc\PersonDaoImpl.java

13. MODIFY THE CODE

PersonaDaoImpl.java:

Click to download

```java
package jdbc;

import java.util.List;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.EmptyResultDataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.namedparam.BeanPropertySqlParameterSource;
import org.springframework.jdbc.core.namedparam.SqlParameterSource;
import org.springframework.stereotype.Repository;

@Repository
public class PersonDaoImpl implements PersonDao {

    private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

    private JdbcTemplate jdbcTemplate;
```

13. MODIFY THE CODE

PersonaDaoImpl.java:

Click to download

```
    @Autowired
    public void setDataSource(DataSource dataSource) {
        //It is not common to use the 2 templates, however if possible.
        //The difference is the handling of parameters by index or by name
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.namedParameterJdbcTemplate = new NamedParameterJdbcTemplate(dataSource);
    }

    // Query with Parameters by name
    // We omit the PK since it is autoincrementable
    private static final String SQL_INSERT_PERSON = "INSERT INTO person (name, email) values (:name, :email)";

    private static final String SQL_UPDATE_PERSON = "UPDATE person set name = :name, email = :email WHERE
id_person = :idPerson";

    private static final String SQL_DELETE_PERSON = "DELETE FROM person WHERE id_person = :idPerson";

    private static final String SQL_SELECT_PERSON = "SELECT id_person, name, email FROM person";

    // Parameters by index
    private static final String SQL_SELECT_PERSON_BY_ID = SQL_SELECT_PERSON + " WHERE id_person = ?";
```

13. MODIFY THE CODE

PersonaDaoImpl.java:

Click to download

```
    @Override
    public List<Person> findAllPeople() {
        RowMapper<Person> personRowMapper = BeanPropertyRowMapper.newInstance(Person.class);
        return this.jdbcTemplate.query(SQL_SELECT_PERSON, personRowMapper);
    }

    @Override
    public int countPeople() {
        String sql = "SELECT count(*) FROM person";
        return this.jdbcTemplate.queryForObject(sql, Integer.class);
    }

    @Override
    public Person findPersonById(int idPerson) {
        Person person;
        try {
            //Utilizamos la clase PersonaRowMapper
            person = jdbcTemplate.queryForObject(SQL_SELECT_PERSON_BY_ID, new PersonRowMapper(), idPerson);
        } catch (EmptyResultDataAccessException e) {
            e.printStackTrace(System.out);
            person = null;
        }
        return person;
    }
```

13. MODIFY THE CODE

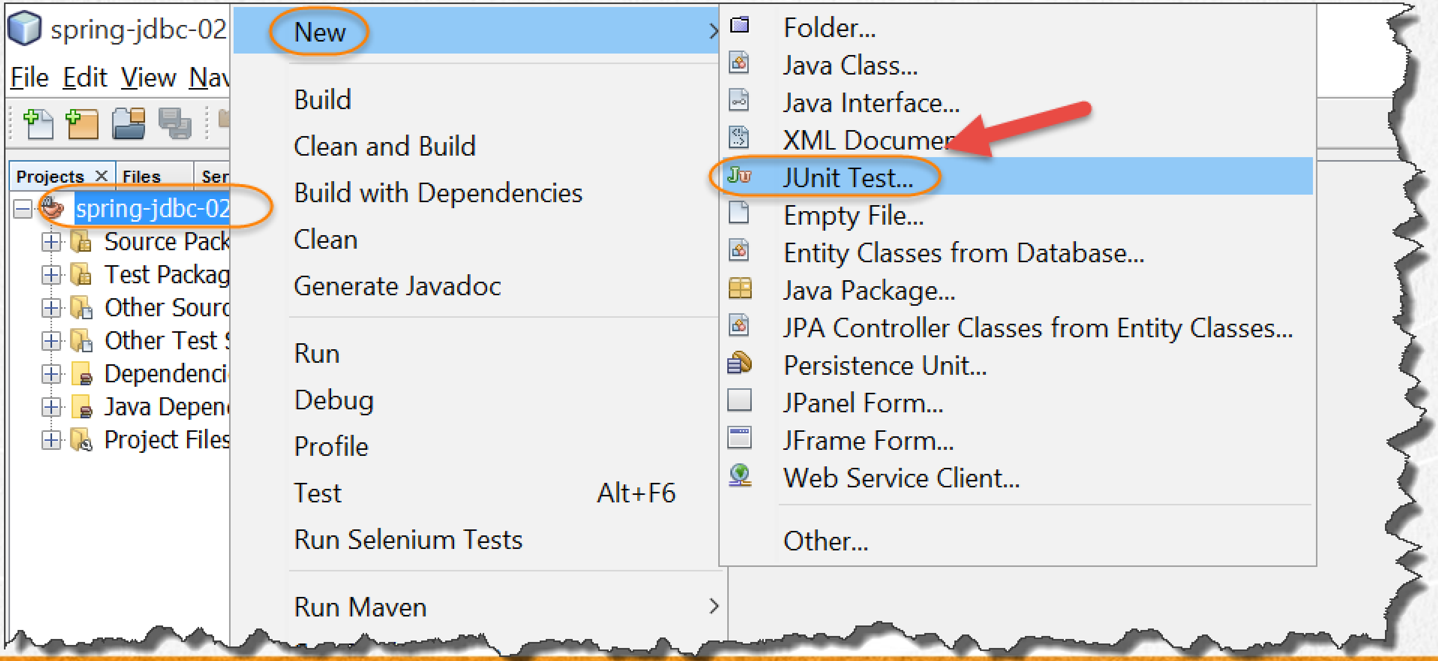
PersonaDaoImpl.java:

Click to download

```
    @Override
    public void insertPerson(Person person) {
        SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(person);
        this.namedParameterJdbcTemplate.update(SQL_INSERT_PERSON, parameterSource);
    }

    @Override
    public void updatePerson(Person person) {
        SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(person);
        this.namedParameterJdbcTemplate.update(SQL_UPDATE_PERSON, parameterSource);
    }

    @Override
    public void deletePerson(Person person) {
        SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(person);
        this.namedParameterJdbcTemplate.update(SQL_DELETE_PERSON, parameterSource);
    }

    @Override
    public Person getPersonByEmail(Person person) {
        String sql = "SELECT * FROM person WHERE email = :email";
        SqlParameterSource namedParameters = new BeanPropertySqlParameterSource(person);
        return this.namedParameterJdbcTemplate.queryForObject(sql, namedParameters, new PersonRowMapper());
    }
}
```

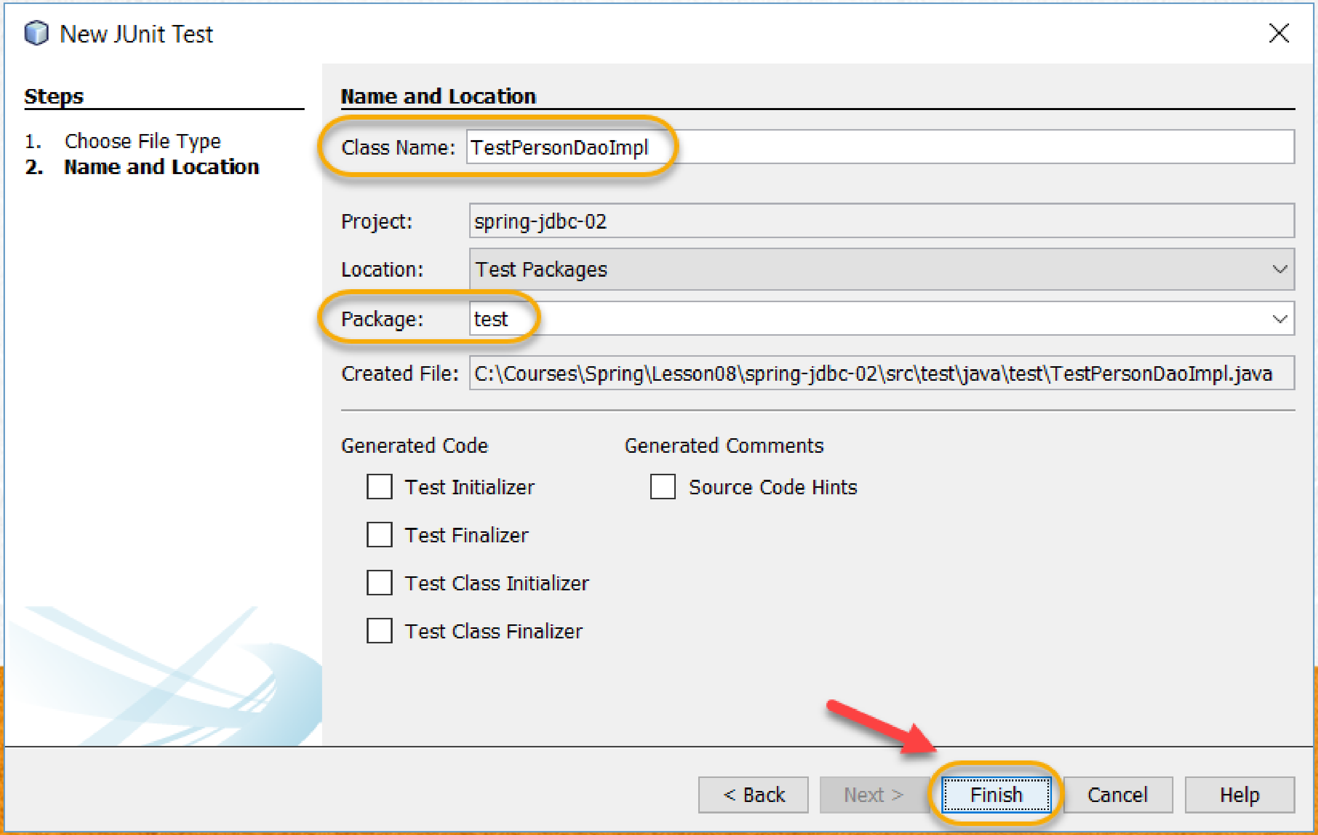
14. CREATE A JAVA CLASS

We created the TestPersonDaoImpl.java file:



14. CREATE A JAVA CLASS

We created the TestPersonDaoImpl.java file:



New JUnit Test

Steps

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

Name and Location

Class Name: TestPersonDaoImpl

Project: spring-jdbc-02

Location: Test Packages

Package: test

Created File: C:\Courses\Spring\Lesson08\spring-jdbc-02\src\test\java\test\TestPersonDaoImpl.java

Generated Code
- [ ] Test Initializer
- [ ] Test Finalizer
- [ ] Test Class Initializer
- [ ] Test Class Finalizer

Generated Comments
- [ ] Source Code Hints

15. MODIFY THE CODE

TestPersonDaoImpl.java:

Click to download

```
package test;

import java.util.List;
import jdbc.Person;
import jdbc.PersonDao;
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:datasource-test.xml", "classpath:applicationContext.xml"})
public class TestPersonDaoImpl {

    private final Logger logger = LogManager.getRootLogger();

    @Autowired
    private PersonDao personDao;
```

15. 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);
    }
}
```

15. MODIFY THE CODE

TestPersonDaoImpl.java:

Click to download

```
    @Test
    public void shouldFindPersonById() {
        try {
            System.out.println();
            logger.info("Start of test shouldFindPersonById");
            int idPerson = 1;
            Person person = personDao.findPersonById(idPerson);

            //According to the recovered person, it should be the same as the record 1
            assertEquals("Admin", person.getName());

            //Print the object
            logger.info("Person found (id=" + idPerson + "): " + person);
            logger.info("End of test shouldFindPersonById");
        } catch (Exception e) {
            logger.error("Error JBDC", e);
        }
    }
```

15. MODIFY THE CODE

TestPersonDaoImpl.java:

Click to download

```
@Test
public void shouldInsertPerson() {
    try {
        System.out.println();
        logger.info("Start oftest shouldInsertPerson");
        // The data script has 3 records
        assertEquals(3, personDao.countPeople());
        Person person = new Person();
        person.setName("Katty");
        person.setEmail("katty@mail.com");
        personDao.insertPerson(person);

        //We retrieve the newly inserted person by email
        person = personDao.getPersonByEmail(person);
        logger.info("Newly inserted person (recovered by email): \n" + person);
        // There should already be four people
        assertEquals(4, personDao.countPeople());
        logger.info("End of test shouldInsertPerson");
    } catch (Exception e) {
        logger.error("Error JBDC", e);
    }
}
```

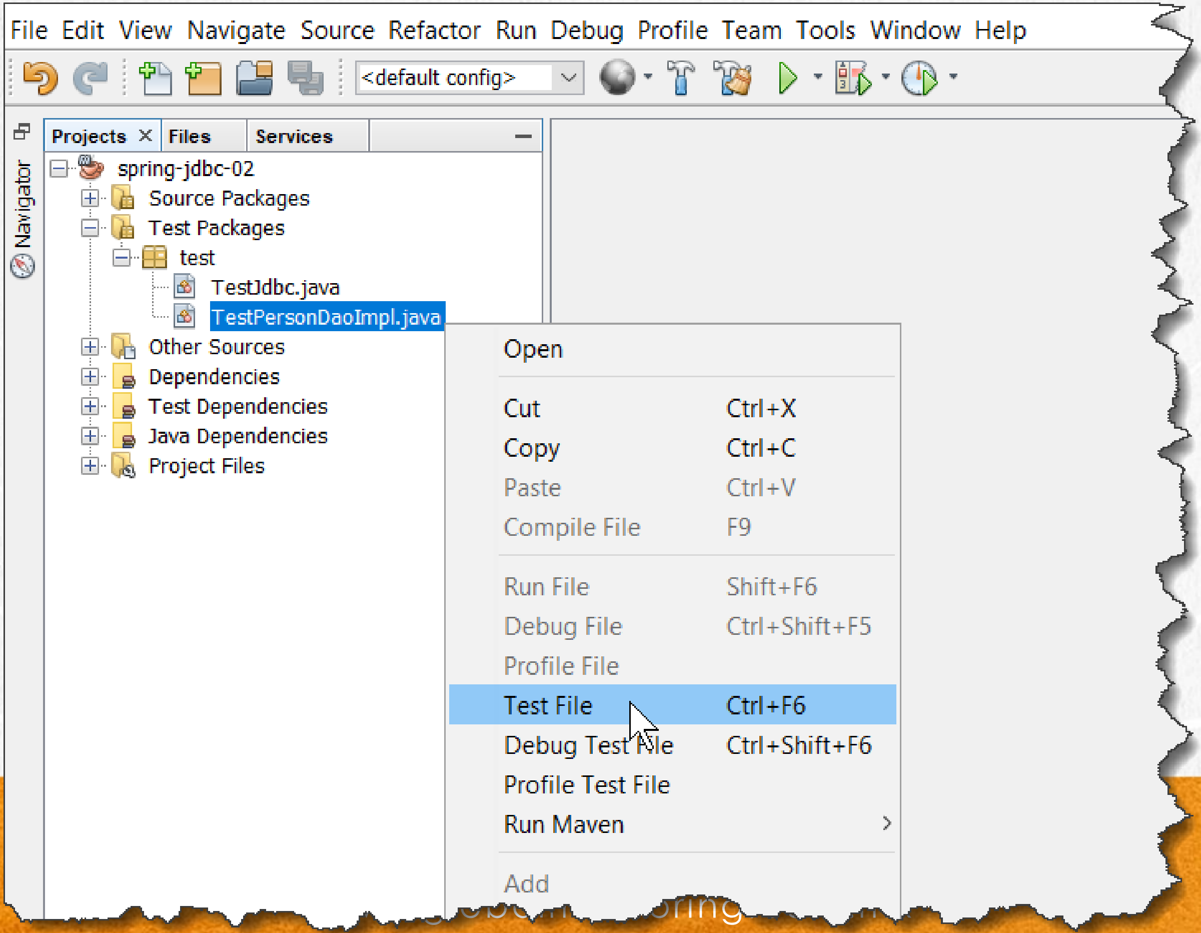
15. MODIFY THE CODE

TestPersonDaoImpl.java:

Click to download

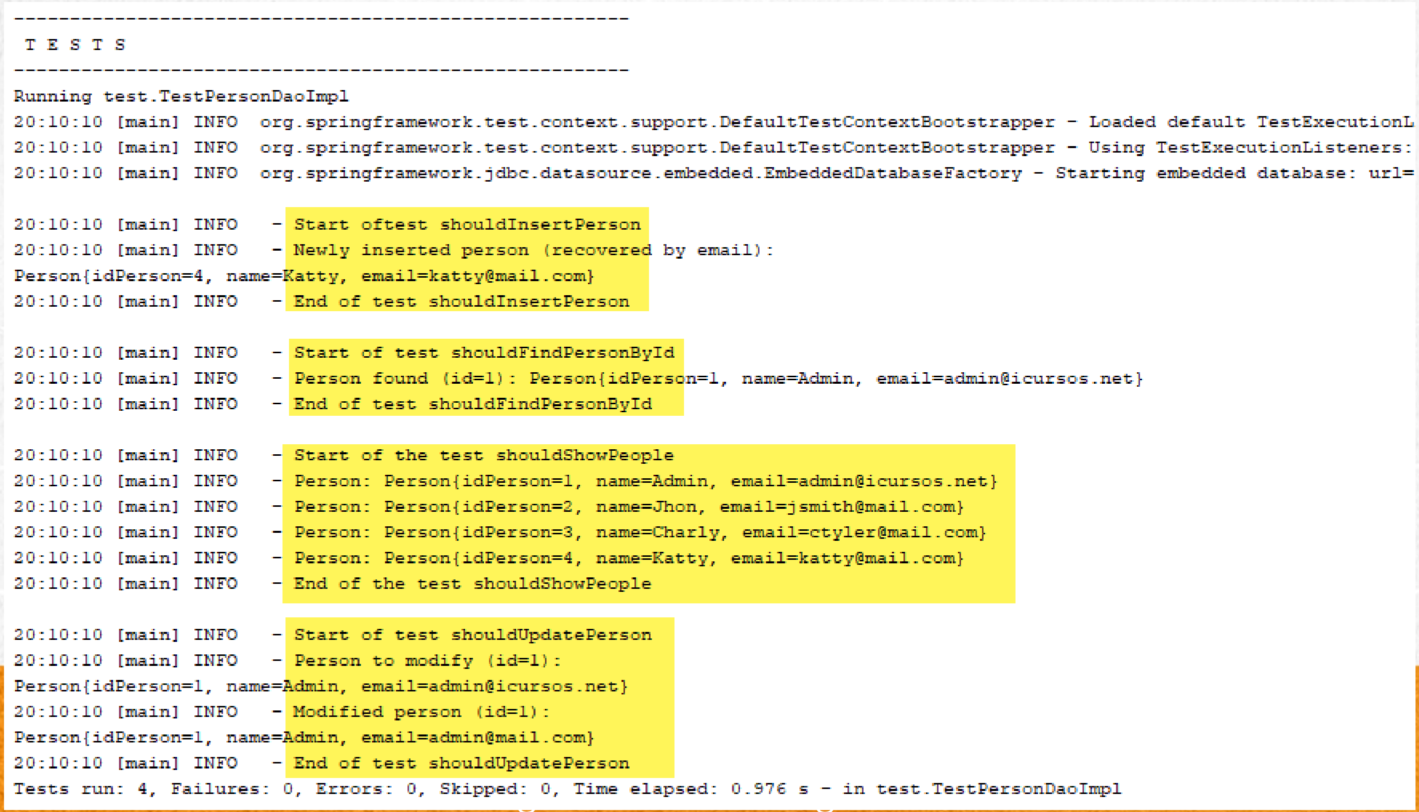
```
    @Test
    public void shouldUpdatePerson() {
        try {
            System.out.println();
            logger.info("Start of test shouldUpdatePerson");
            int idPerson = 1;
            Person person = personDao.findPersonById(idPerson);
            logger.info("Person to modify (id=" + idPerson + "): \n" + person);
            //Update the email
            person.setEmail("admin@mail.com");
            personDao.updatePerson(person);
            //We read the user again
            person = personDao.findPersonById(idPerson);
            //According to the person recovered, it should be the same as the record 1
            assertEquals("admin@mail.com", person.getEmail());
            //We print the whole object
            logger.info("Modified person (id=" + idPerson + "): \n" + person);
            logger.info("End of test shouldUpdatePerson");
        } catch (Exception e) {
            logger.error("Error JBDC", e);
        }
    }
}
```

16. EXECUTE THE PROJECT



16. EXECUTE THE PROJECT

The output of the project is as follows:



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running test.TestPersonDaoImpl
20:10:10 [main] INFO  org.springframework.test.context.support.DefaultTestContextBootstrapper - Loaded default TestExecutionL
20:10:10 [main] INFO  org.springframework.test.context.support.DefaultTestContextBootstrapper - Using TestExecutionListeners:
20:10:10 [main] INFO  org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseFactory - Starting embedded database: url=

20:10:10 [main] INFO   - Start oftest shouldInsertPerson
20:10:10 [main] INFO   - Newly inserted person (recovered by email):
Person{idPerson=4, name=Katty, email=katty@mail.com}
20:10:10 [main] INFO   - End of test shouldInsertPerson

20:10:10 [main] INFO   - Start of test shouldFindPersonById
20:10:10 [main] INFO   - Person found (id=1): Person{idPerson=1, name=Admin, email=admin@icursos.net}
20:10:10 [main] INFO   - End of test shouldFindPersonById

20:10:10 [main] INFO   - Start of the test shouldShowPeople
20:10:10 [main] INFO   - Person: Person{idPerson=1, name=Admin, email=admin@icursos.net}
20:10:10 [main] INFO   - Person: Person{idPerson=2, name=Jhon, email=jsmith@mail.com}
20:10:10 [main] INFO   - Person: Person{idPerson=3, name=Charly, email=ctyler@mail.com}
20:10:10 [main] INFO   - Person: Person{idPerson=4, name=Katty, email=katty@mail.com}
20:10:10 [main] INFO   - End of the test shouldShowPeople

20:10:10 [main] INFO   - Start of test shouldUpdatePerson
20:10:10 [main] INFO   - Person to modify (id=1):
Person{idPerson=1, name=Admin, email=admin@icursos.net}
20:10:10 [main] INFO   - Modified person (id=1):
Person{idPerson=1, name=Admin, email=admin@mail.com}
20:10:10 [main] INFO   - End of test shouldUpdatePerson
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.976 s - in test.TestPersonDaoImpl
```

EXERCISE CONCLUSION

With this exercise we have implemented the use of Spring JDBC applying design patterns such as Entity class (Person.java), DAO class (PersonDao.java), as well as the implementation of this design pattern (PersonDaoImpl.java).

Later we created a unit test to check the methods of the PersonDaoImp.java class.

CURSO ONLINE

SPRING FRAMEWORK

Por: Ing. Ubaldo Acosta

Global Mentoring

Experiencia y Conocimiento para tu vida

SPRING FRAMEWORK COURSE

www.globalmentoring.com.mx