EXERCISE

DATA LAYER WITH JDBC

EXERCISE OBJECTIVE

Create a program to create a logical data layer using JDBC. At the end we should observe the following:

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

<default config>

Projects | Files | Services | Favorites

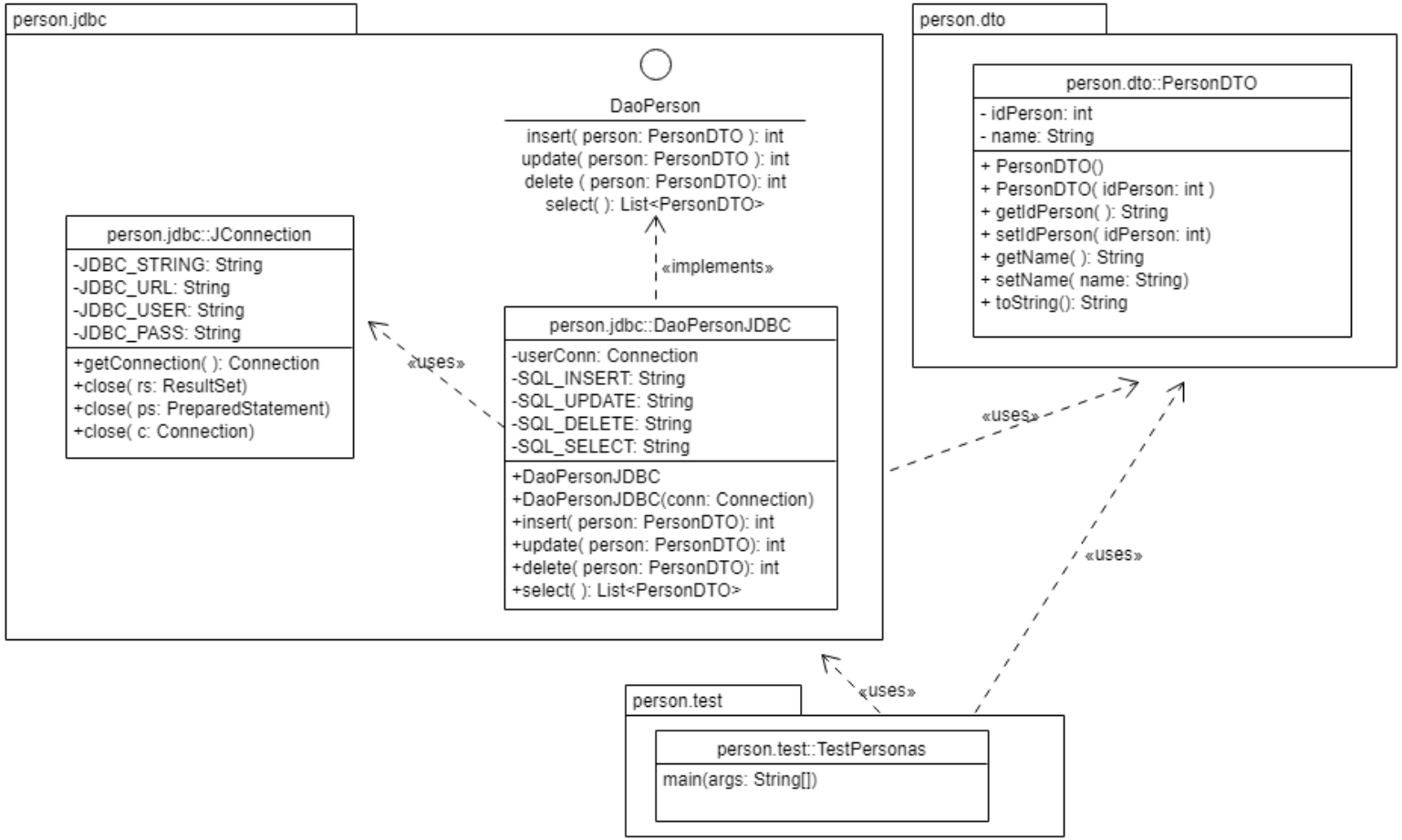
- PersonDataLayer
  - Source Packages
    - person.dto
      - PersonDTO.java
    - person.jdbc
      - DaoPerson.java
      - DaoPersonJdbc.java
      - JConnection.java
    - person.test
      - PersonTest.java
  - Test Packages
  - Other Sources
  - Dependencies
  - Java Dependencies
  - Project Files

PersonDTO.java | JConnection.java | DaoPerson.java | DaoPersonJdbc.java | PersonTest.java

Source | History

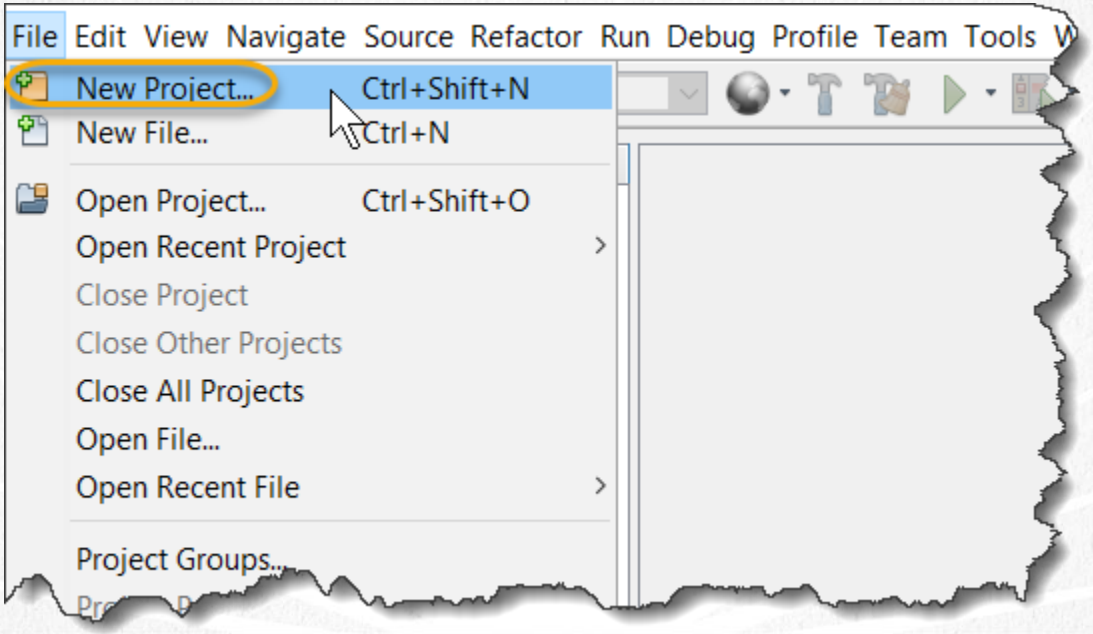
```java
package person.test;

import java.sql.SQLException;
import java.util.List;
import person.dto.PersonDTO;
import person.jdbc.*;

public class PersonTest {
        public static void main(String[] args) {
        //We use the interface type as reference to a specific class
        DaoPerson personaDao = new DaoPersonJdbc();

        //We create a new record
        //We use the DTO person class which is used
        //to transfer the information between the layers
        //it is not necessary to specify the primary key
        //since in this case it is of the autonumeric type
        //and the DB is in charge to assign a new value
        PersonDTO personDto = new PersonDTO();
        personDto.setName("Charly");
        //Utilizamos la capa DAO para persistir el objeto DTO
        try {
            //personaDao.insert(personDto);

```

CLASS DIAGRAM



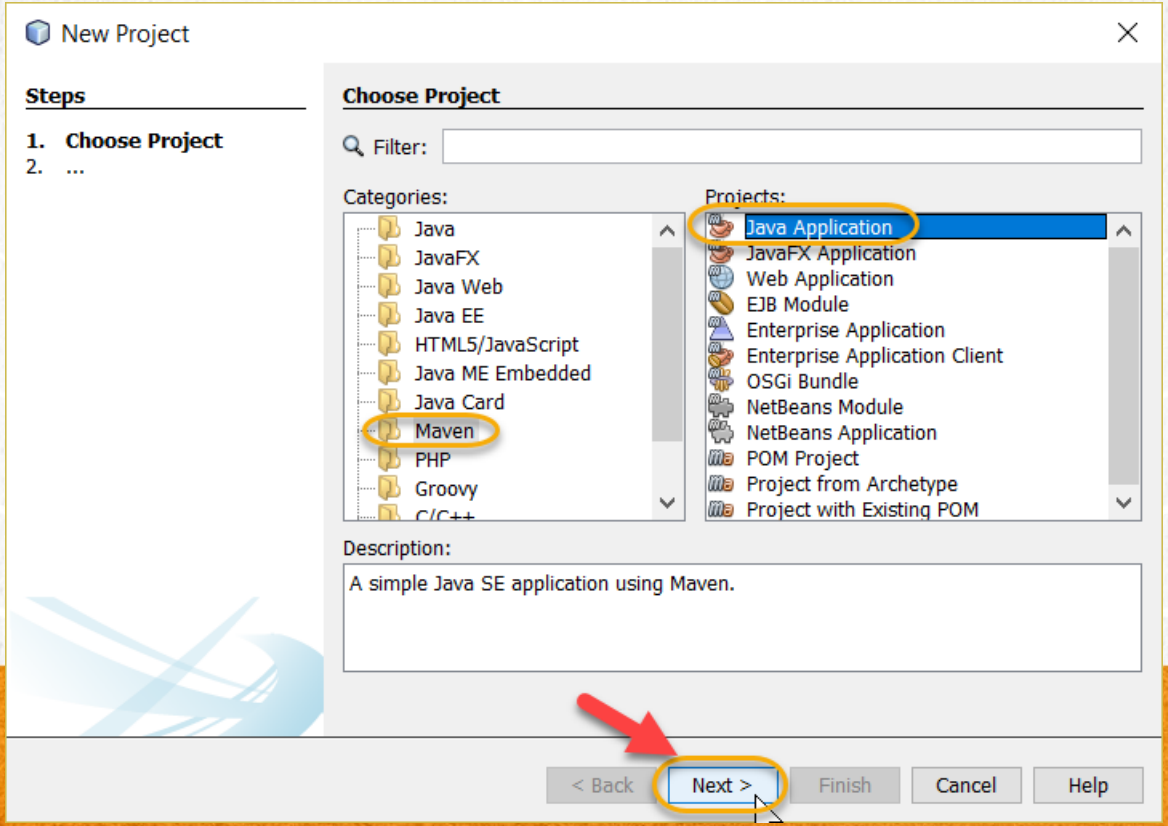
1. CREATE A NEW PROJECT

Create a new project:



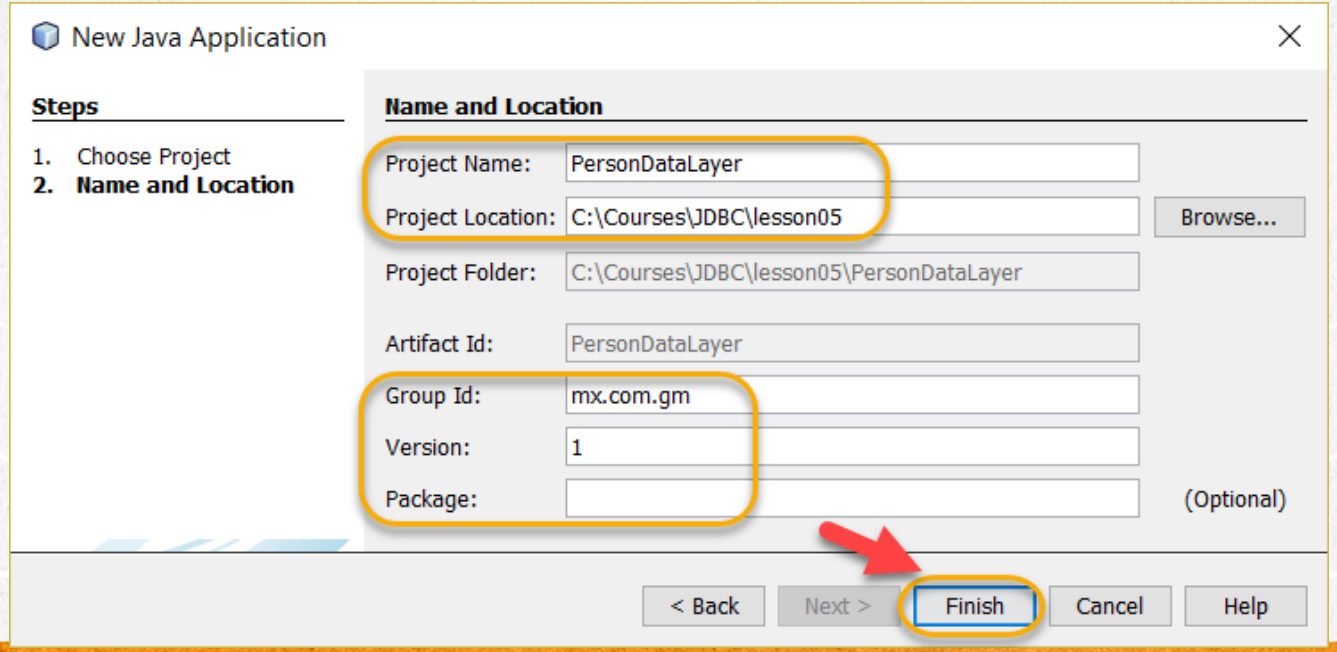
1. CREATE A NEW PROJECT

Create a new project:



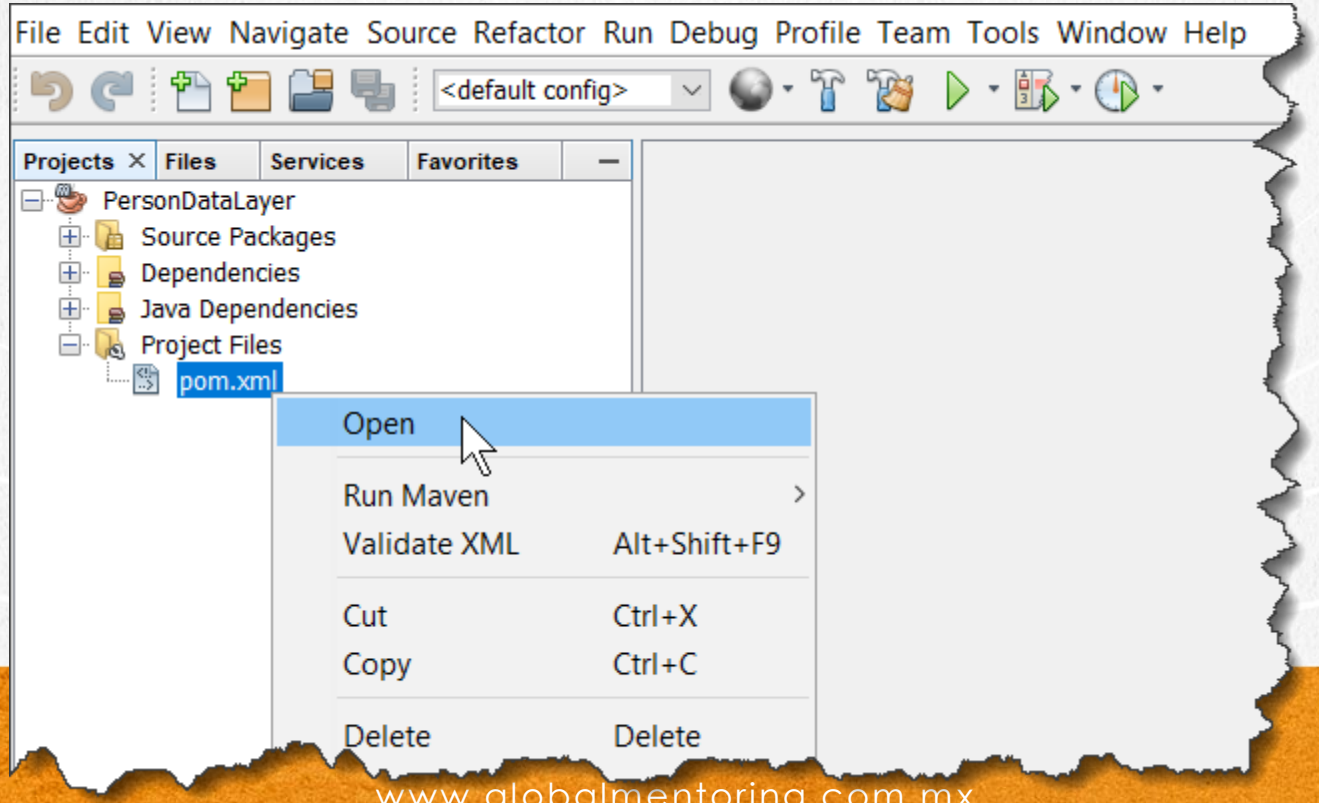
1. CREATE A NEW PROJECT

Create a new project:



2. MODIFY THE POM.XML

Modify the pom.xml to add the mysql.jar:



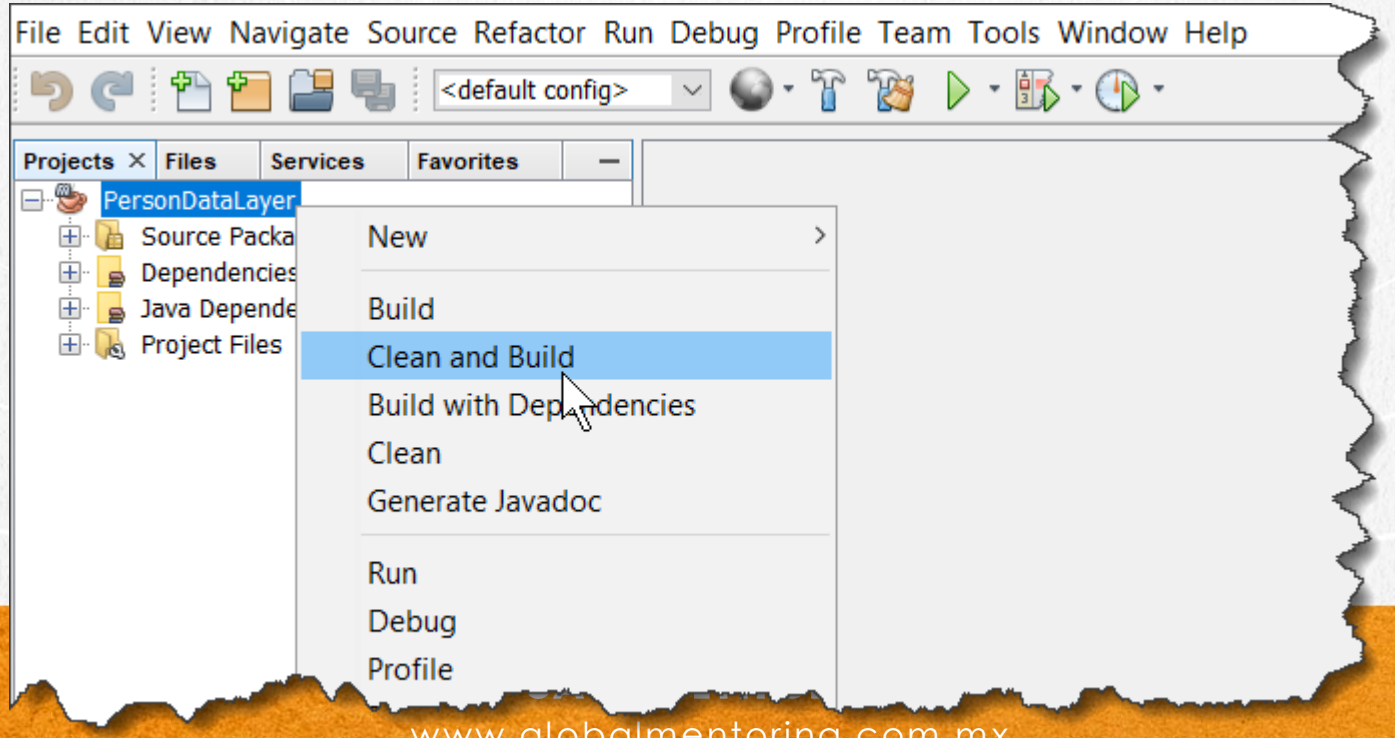
2. MODIFY THE CODE

pom.xml:

```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>mx.com.gm</groupId>
    <artifactId>PersonDataLayer</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>
    </properties>
    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.46</version>
        </dependency>
    </dependencies>
</project>
```

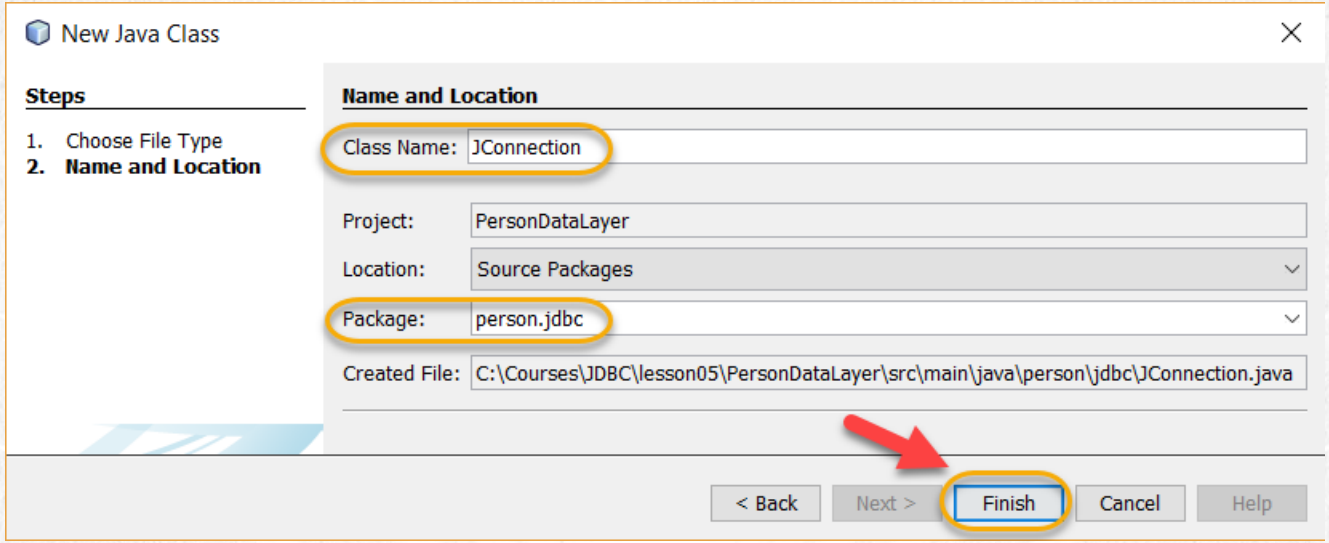
3. CLEAN & BUILD

Execute the clean & build option:



4. CREATE A NEW CLASS

Create a new class:



5. MODIFY THE CODE

JConnection.java:

```
package person.jdbc;

import java.sql.*;

public class JConnection {

    private static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    private static final String JDBC_URL = "jdbc:mysql://localhost/test?useSSL=false";
    private static final String JDBC_USER = "root";
    private static final String JDBC_PASS = "admin";
    private static Driver driver;

    public static synchronized Connection getConnection() throws SQLException {
        if (driver == null) {
            try {
                Class jdbcDriverClass = Class.forName(JDBC_DRIVER);
                driver =  (Driver) jdbcDriverClass.newInstance();
                DriverManager.registerDriver(driver);
            } catch (Exception e) {
                System.out.println("Failure to load the JDBC driver");
                e.printStackTrace(System.out);
            }
        }
        return DriverManager.getConnection(JDBC_URL, JDBC_USER, JDBC_PASS);
    }
```

5. MODIFY THE CODE

JConnection.java:

```
//Close the resultSet object
public static void close(ResultSet rs) {
    try {
        if (rs != null) {
            rs.close();
        }
    } catch (SQLException sqle) {
        sqle.printStackTrace(System.out);
    }
}

//Close the PrepareStatement object
public static void close(PreparedStatement stmt) {
    try {
        if (stmt != null) {
            stmt.close();
        }
    } catch (SQLException sqle) {
        sqle.printStackTrace(System.out);
    }
}
```

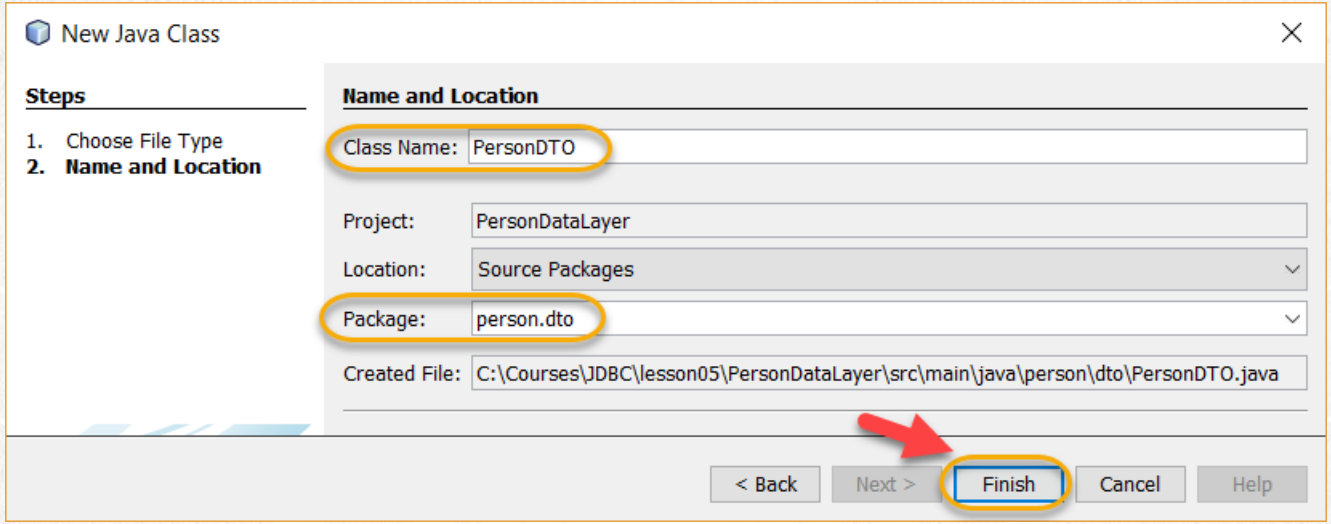
5. MODIFY THE CODE

JConnection.java:

```
    //Close the connection object
    public static void close(Connection conn) {
        try {
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException sqle) {
            sqle.printStackTrace(System.out);
        }
    }
}
```

6. CREATE A NEW CLASS

Create a new class:



7. MODIFY THE CODE

PersonDTO.java:

```
package person.dto;

public class PersonDTO {

    private int idPerson;
    private String name;

    public PersonDTO(){}

    public PersonDTO(int personId){
        this.idPerson = personId;
    }

    public int getIdPerson() {
        return idPerson;
    }

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

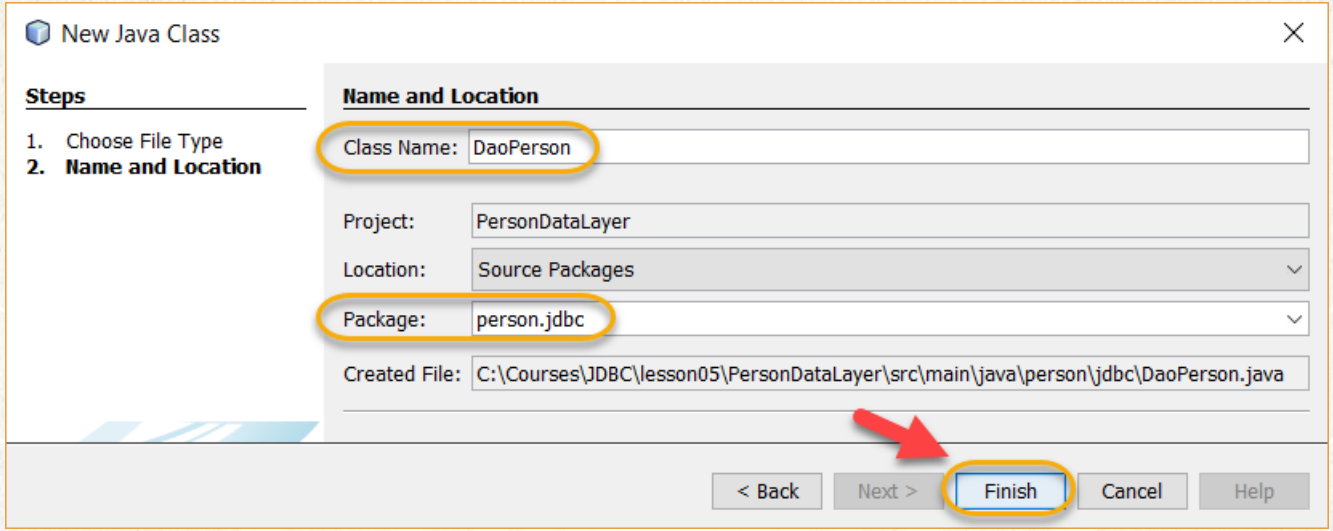
7. MODIFY THE CODE

PersonDTO.java:

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

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

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

8. CREATE A NEW CLASS

Create a new class:



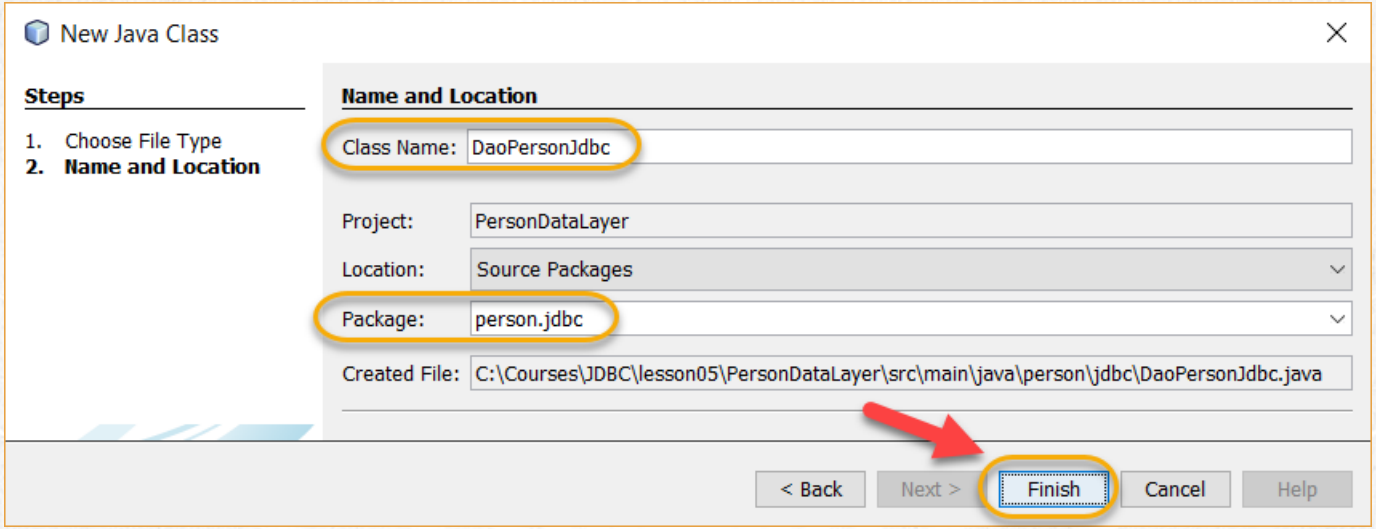
9. MODIFY THE CODE

DaoPerson.java:

```
package person.jdbc;

import java.sql.SQLException;
import java.util.List;
import person.dto.PersonDTO;

public interface DaoPerson {

    public int insert(PersonDTO persona) throws SQLException;

    public int update(PersonDTO persona) throws SQLException;

    public int delete(PersonDTO persona) throws SQLException;

    public List<PersonDTO> select() throws SQLException;
}
```

10. CREATE A NEW CLASS

Create a new class:



11. MODIFY THE CODE

DaoPersonJdbc.java:

```
package person.jdbc;

import java.sql.*;
import java.util.*;
import person.dto.PersonDTO;

public class DaoPersonJdbc implements DaoPerson{

    private java.sql.Connection userConn;
    private final String SQL_INSERT = "INSERT INTO person(name) VALUES(?)";
    private final String SQL_UPDATE = "UPDATE person SET name=? WHERE id_person=?";
    private final String SQL_DELETE = "DELETE FROM person WHERE id_person = ?";
    private final String SQL_SELECT = "SELECT id_person, name FROM person ORDER BY id_person";

    /*
     * Add the empty constructor
     */
    public DaoPersonJdbc() {
    }
```

11. MODIFY THE CODE

DaoPersonJdbc.java:

```
/**
 * Constructor that assigns an existing connection to be used in the queries
 * of this class
 *
 * @param conn Connection to the DB previously created
 */
public DaoPersonJdbc(Connection conn) {
    this.userConn = conn;
}
```

11. MODIFY THE CODE

DaoPersonJdbc.java:

```
@Override
public int insert(PersonDTO personDTO) throws SQLException {
    Connection conn = null;
    PreparedStatement stmt = null;
    int rows = 0; //affected rows
    try {
        //If the connection to reuse is different from null, it is used, if not
        //create a new connection
        conn = (this.userConn != null) ? this.userConn : JConnection.getConnection();
        stmt = conn.prepareStatement(SQL_INSERT);
        stmt.setString(1, personDTO.getName());//param 1 => ? name
        System.out.println("Executing query:" + SQL_INSERT);
        rows = stmt.executeUpdate();
        System.out.println("Affected records:" + rows);
    } finally {
        if (this.userConn == null) {
            JConnection.close(conn);
        }
    }
    return rows;
}
```

11. MODIFY THE CODE

DaoPersonJdbc.java:

```
@Override
public int update(PersonDTO personDTO) throws SQLException {
    Connection conn = null;
    PreparedStatement stmt = null;
    int rows = 0;
    try {
        conn = (this.userConn != null) ? this.userConn : JConnection.getConnection();
        System.out.println("Executing query:" + SQL_UPDATE);
        stmt = conn.prepareStatement(SQL_UPDATE);
        stmt.setString(1, personDTO.getName());//param 1 => ?  name
        stmt.setInt(2, personDTO.getIdPerson());//param 2 => ? id_person
        rows = stmt.executeUpdate();
        System.out.println("Updated records:" + rows);
    } finally {
       if (this.userConn == null) {
            JConnection.close(conn);
        }
    }
    return rows;
}
```

11. MODIFY THE CODE

DaoPersonJdbc.java:

```
@Override
public int delete(PersonDTO personDTO) throws SQLException {
    Connection conn = null;
    PreparedStatement stmt = null;
    int rows = 0;
    try {
        conn = (this.userConn != null) ? this.userConn : JConnection.getConnection();
        System.out.println("Executing query:" + SQL_DELETE);
        stmt = conn.prepareStatement(SQL_DELETE);
        stmt.setInt(1, personDTO.getIdPerson());//param 1 => ? id_person
        rows = stmt.executeUpdate();
        System.out.println("Deleted records:" + rows);
    } finally {
        if (this.userConn == null) {
            JConnection.close(conn);
        }
    }
    return rows;
}
```

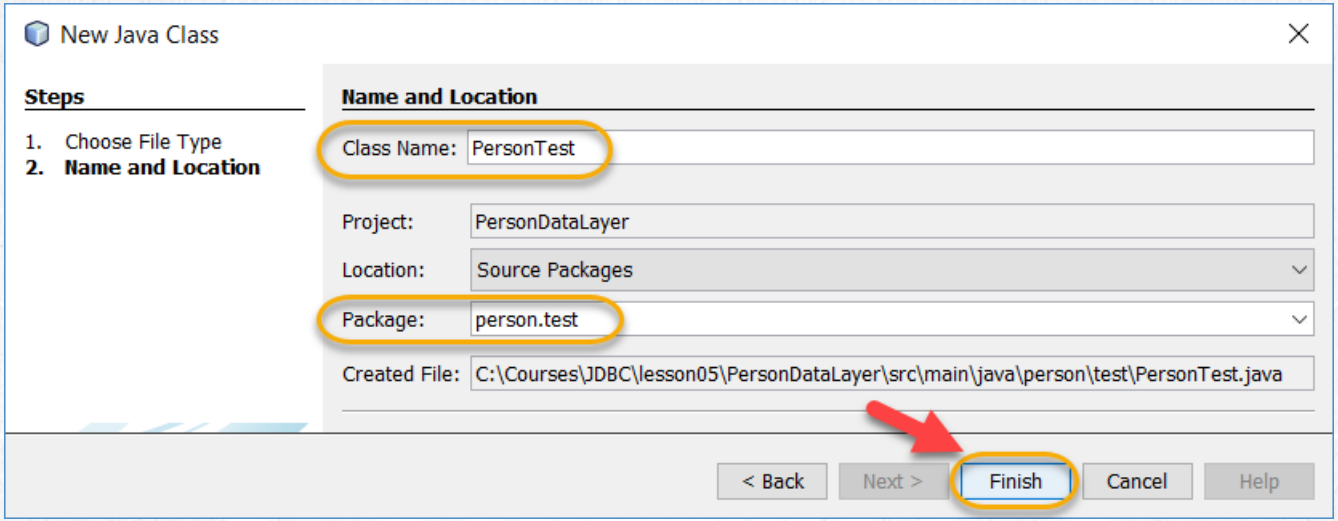
11. MODIFY THE CODE

DaoPersonJdbc.java:

```
    @Override
    public List<PersonDTO> select() throws SQLException {
        Connection conn = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        PersonDTO persona = null;
        List<PersonDTO> personas = new ArrayList<>();
        try {
            conn = (this.userConn != null) ? this.userConn : JConnection.getConnection();
            stmt = conn.prepareStatement(SQL_SELECT);
            rs = stmt.executeQuery();
            while (rs.next()) {
                int id_persona = rs.getInt(1);
                String nombre = rs.getString(2);
                persona = new PersonDTO();
                persona.setIdPerson(id_persona);
                persona.setName(nombre);
                personas.add(persona);
            }
        } finally {
            if (this.userConn == null)
                JConnection.close(conn);
        }
        return personas;
    }
}
```

12. CREATE A NEW CLASS

Create a new class:



13. MODIFY THE CODE

PersonTest.java:

```
package person.test;

import java.sql.SQLException;
import java.util.List;
import person.dto.PersonDTO;
import person.jdbc.*;

public class PersonTest {
        public static void main(String[] args) {
        //We use the interface type as reference to a specific class
        DaoPerson personaDao = new DaoPersonJdbc();

        //We create a new record
        //We use the DTO person class which is used
        //to transfer the information between the layers
        //it is not necessary to specify the primary key
        //since in this case it is of the autonumeric type
        //and the DB is in charge to assign a new value
        PersonDTO personDto = new PersonDTO();
        personDto.setName("Charly");
        //Utilizamos la capa DAO para persistir el objeto DTO
        try {
            //personaDao.insert(personDto);

            //we remove a record, the id 3
//            personaDao.delete( new PersonDTO(3));

            //update a record
//             PersonDTO personDto2= new PersonDTO();
//             personDto2.setIdPerson(2);//updated the record 2
//             personDto2.setName("Katty2");
//             personaDao.update(personDto2);
```

13. MODIFY THE CODE

PersonTest.java:

```
            //Seleccionamos todos los registros
            List<PersonDTO> personas = personaDao.select();
            for (PersonDTO personaDTO : personas) {
                System.out.print( personaDTO );
                System.out.println();
            }

        } catch (SQLException e) {
            System.out.println("Data Layer Exception");
            e.printStackTrace(System.out);
        }
    }

}
```

14. EXECUTE THE PROJECT

The result is as follows:

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

<default config>

Output - Run (PersonTest)

```
Scanning for projects...

------------------------------------------------------------------------
Building PersonDataLayer 1
------------------------------------------------------------------------

--- exec-maven-plugin:1.2.1:exec (default-cli) @ PersonDataLayer ---
PersonDTO{idPerson=1, name=John}
PersonDTO{idPerson=2, name=Katty}
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

EXERCISE CONCLUSION

As this exercise we have seen how to create a data layer using JDBC.

We also apply some design patterns such as: DAO and DTO, which we will be using when we create our data layers.

ONLINE COURSE

JAVA WITH JDBC

By: Eng. Ubaldo Acosta

Experiencia y Conocimiento para tu vida