EXERCISE

BASIC OPERATIONS WITH HIBERNATE AND JPA

EXERCISE OBJECTIVE

Create an exercise to perform basic operations using Hibernate and JPA. The final result is as follows :

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

Projects | Files | Services

- HibernateExample2
  - Source Packages
    - dao
    - domain
    - test
      - HelloWorldHibernate.java
      - OperationsHibernateJPA.java
  - Test Packages
  - Other Sources
  - Dependencies
  - Java Dependencies
  - Project Files

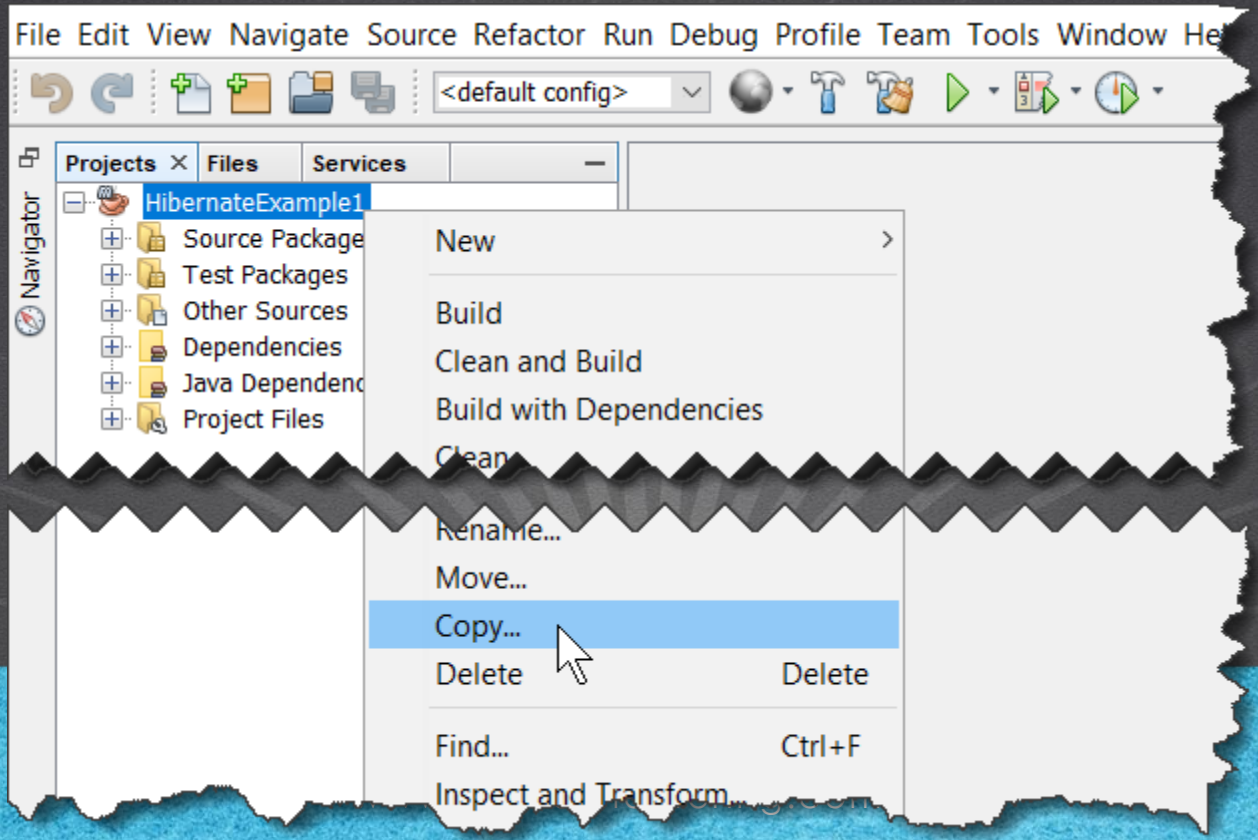
OperationsHibernateJPA.java

```
6
7   public class OperationsHibernateJPA {
8
9       public static void main(String args[]) {
10          // Variables generales
11          PersonDAO personDao = new PersonDAO();
12          Person person = null;
13          int option = -1;
14          Scanner scanner = new Scanner(System.in);
15          String idStr, name;
16
```

```
Output
Retriever Output | Run (OperationsHibernateJPA)
21:39:08 [main] WARN  org.hibernate.orm.connections.pooling - HHH10001002: Using Hibernate built-in connection pool (not
21:39:08 [main] INFO  org.hibernate.orm.connections.pooling - HHH10001005: using driver [com.mysql.jdbc.Driver] at URL [
21:39:08 [main] INFO  org.hibernate.orm.connections.pooling - HHH10001001: Connection properties: {user=root, password=
21:39:08 [main] INFO  org.hibernate.orm.connections.pooling - HHH10001003: Autocommit mode: false
21:39:08 [main] INFO  org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl - HHH000115: H
21:39:08 [main] INFO  org.hibernate.dialect.Dialect - HHH000400: Using dialect: org.hibernate.dialect.MySQL57Dialect
Choose an option:
1.- List People
2.- Find person by id
3.- Add a person
4.- Modify a person
5.- Delete a person
0.- Exit
```

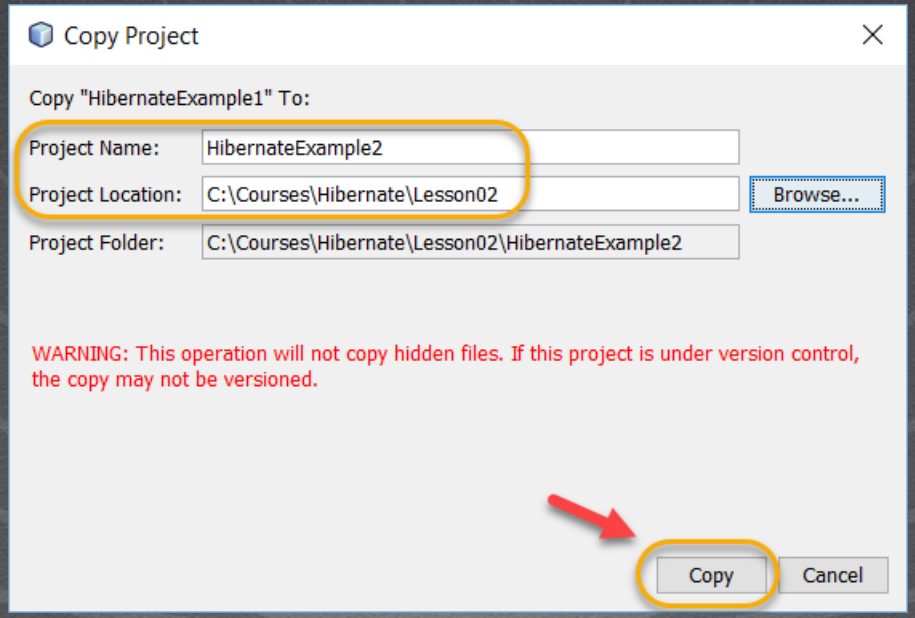
1. CREATE THE PROJECT

We copy and paste the previous project:



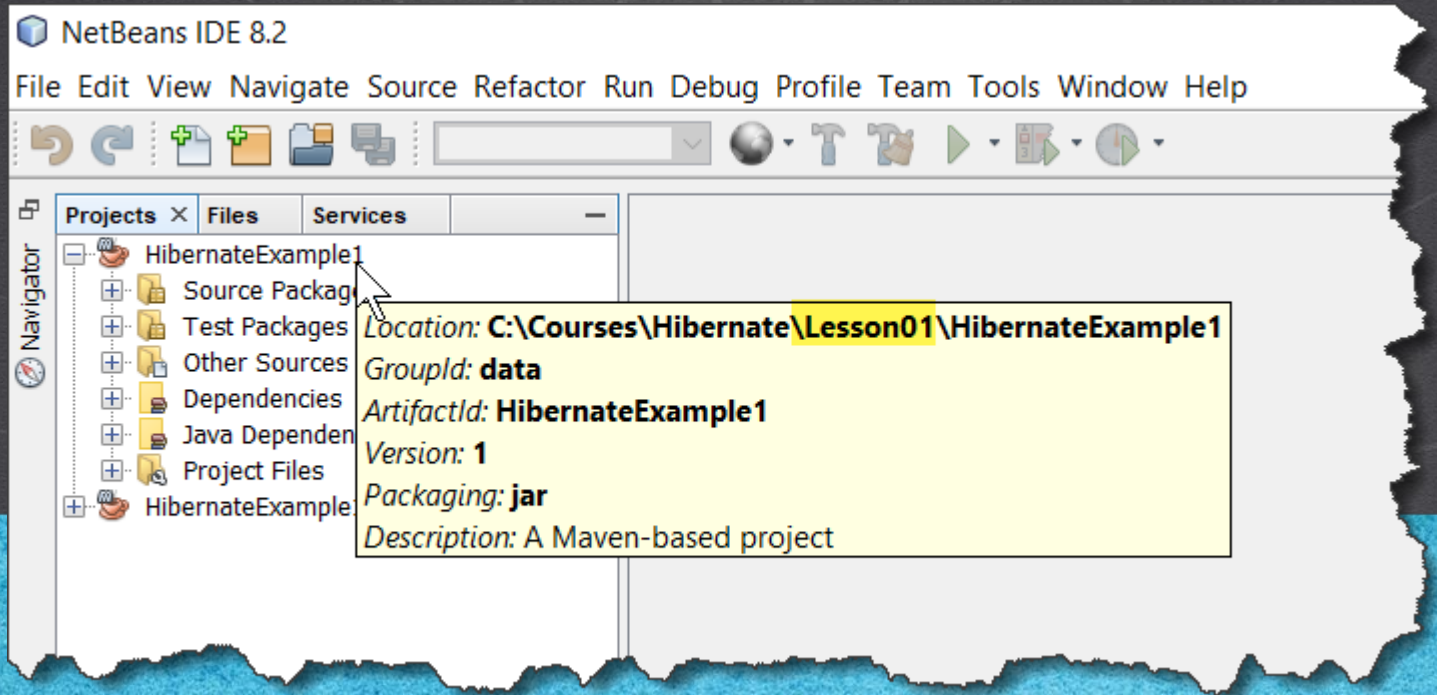
1. CREATE THE PROJECT

We copy and paste the previous project:



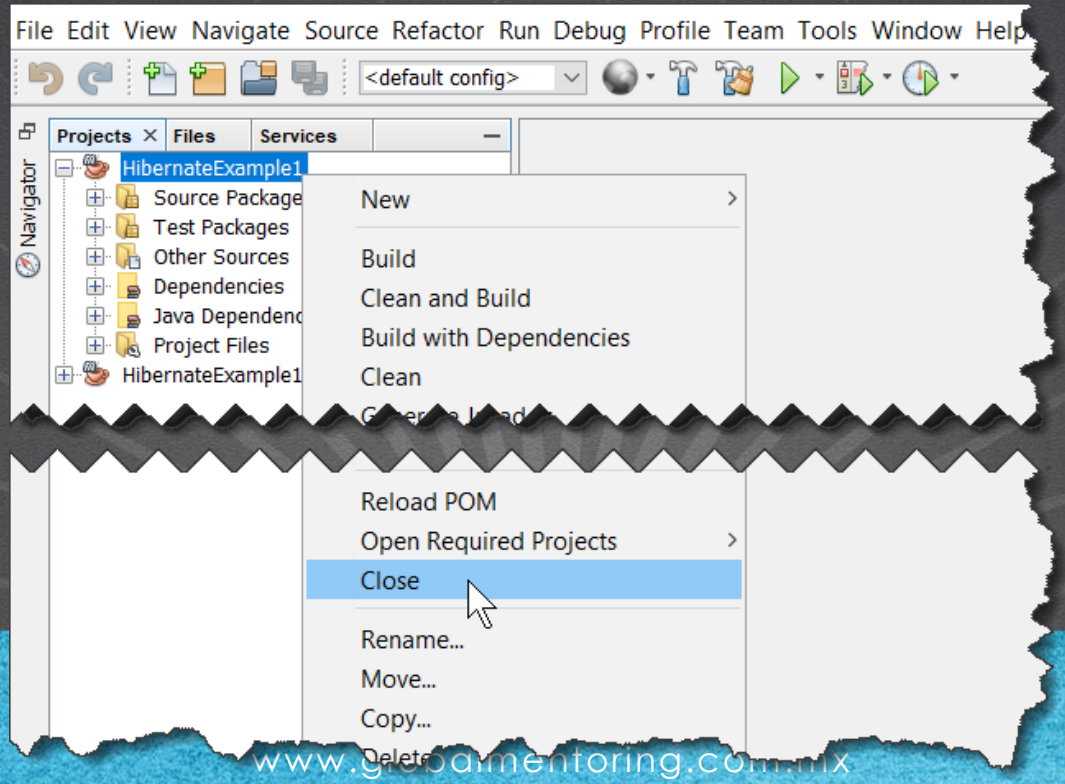
1. CREATE THE PROJECT

We position ourselves on the project to know which one we should close and which one we should rename. We closed the project of Lesson01 :



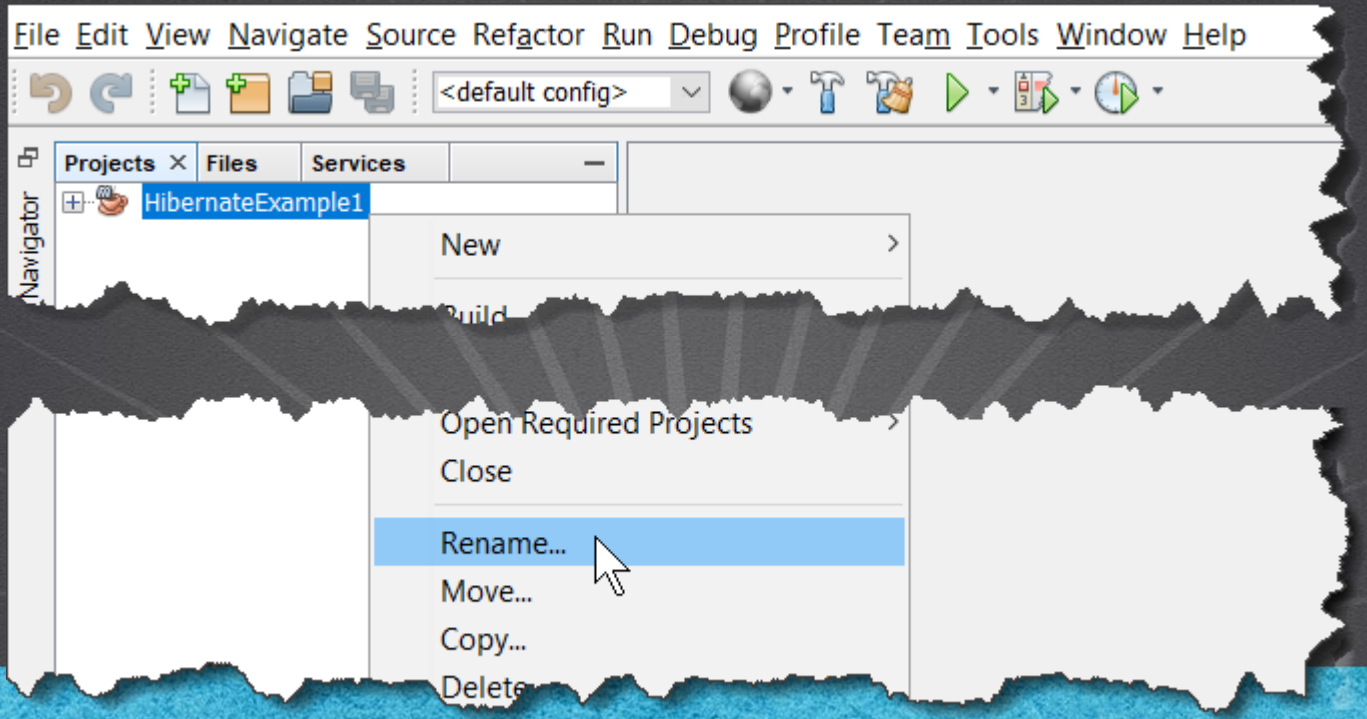
1. CREATE THE PROJECT

We closed the project of Lesson01:



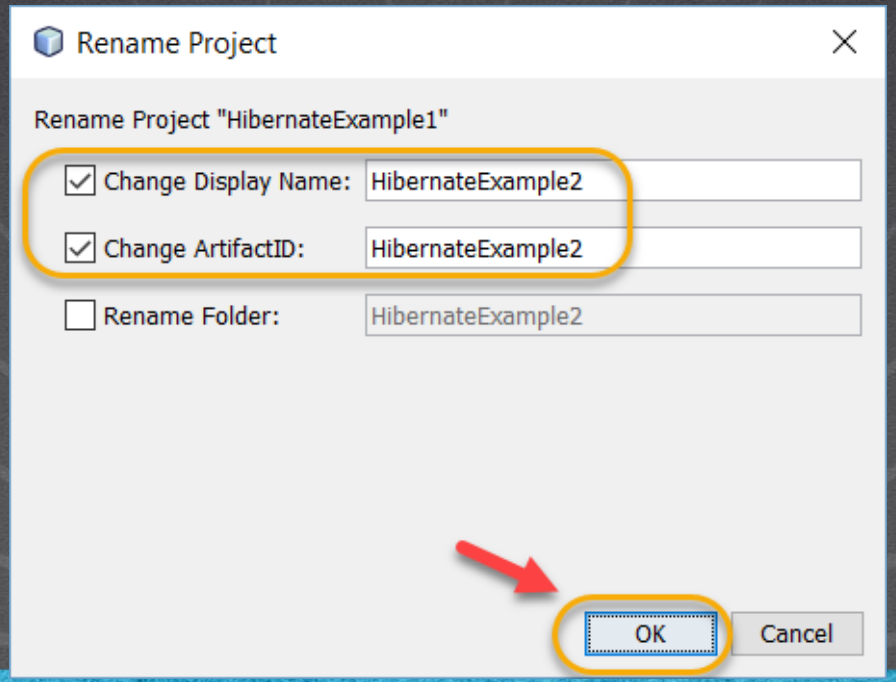
1. CREATE THE PROJECT

We rename the project of Lesson02:



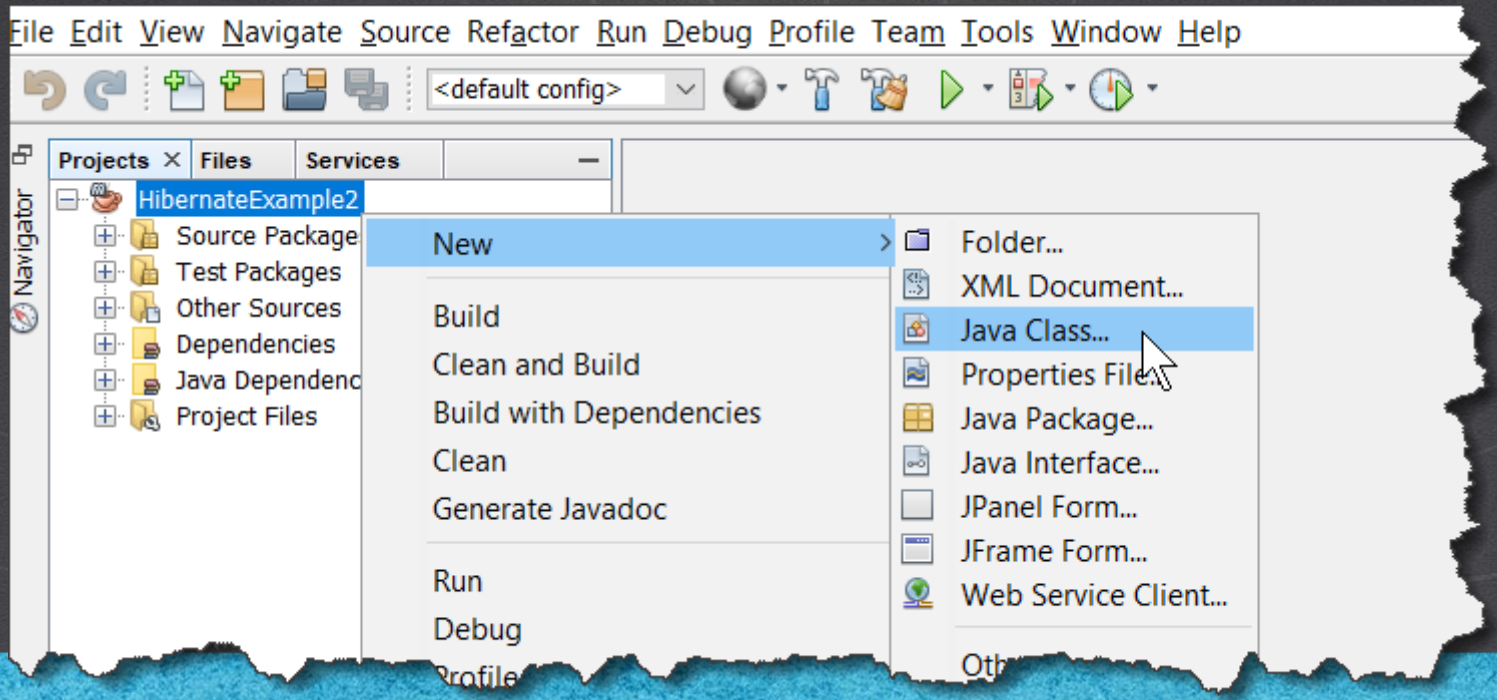
1. CREATE THE PROJECT

We rename the project of Lesson02 as follows:



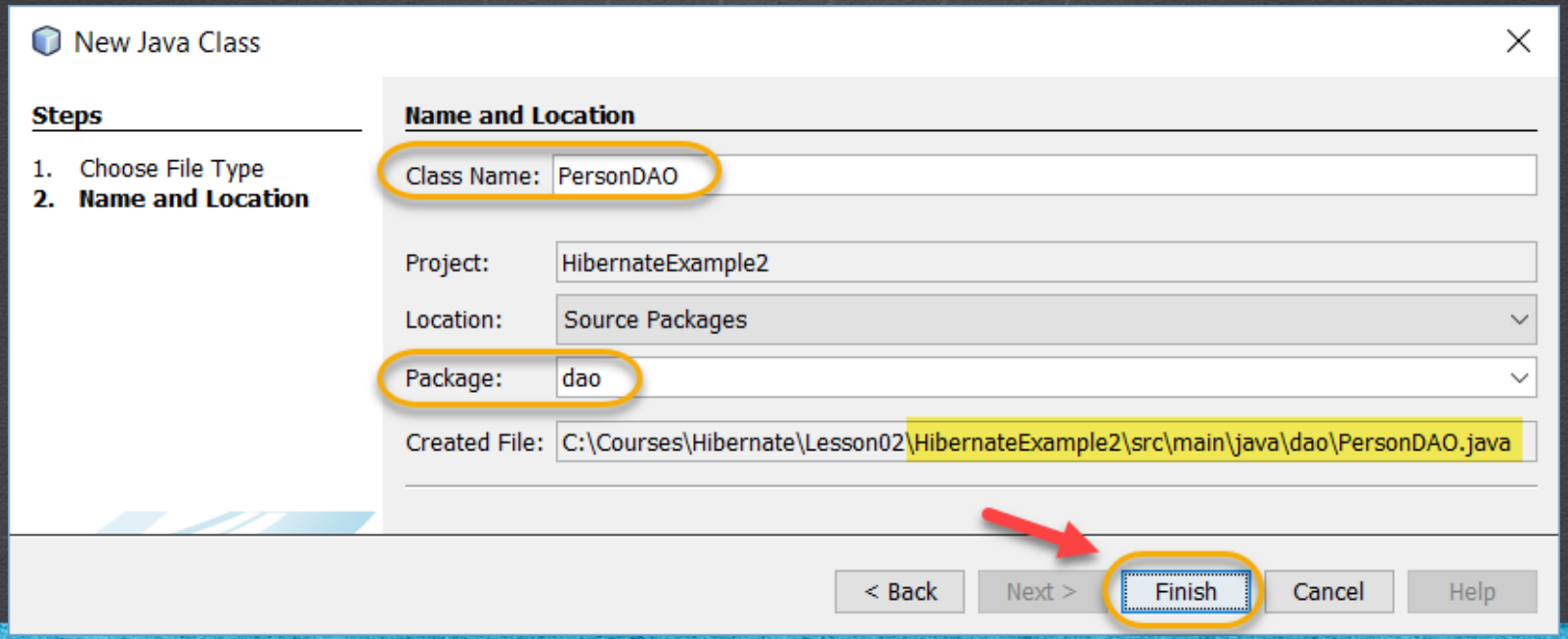
2. CREATE A NEW JAVA CLASS

We create the class PersonDAO.java:



2. CREATE A NEW JAVA CLASS

We create the class PersonDAO.java:



3. MODIFY THE CODE

PersonDAO.java:

```
package dao;

import java.util.List;
import javax.persistence.*;
import domain.Person;
import org.apache.logging.log4j.*;

public class PersonDAO {

    Logger log = LogManager.getLogger(PersonDAO.class);

    protected EntityManager em;
    private EntityManagerFactory emf = null;

    public PersonDAO() {
        // use the persistence unit defined in the persistence.xml file
        emf = Persistence.createEntityManagerFactory("HibernatePU");
    }

    private EntityManager getEntityManager() {
        return emf.createEntityManager();
    }
```

3. MODIFY THE CODE

PersonDAO.java:

```
public void list() {
    // Query to be executed
    // We do not need to create a new transaction
    String hql = "SELECT p FROM Person p";
    em = getEntityManager();
    Query query = em.createQuery(hql);
    List<Person> list = query.getResultList();
    for (Person p : list) {
        log.info(p);
    }
}
```

3. MODIFY THE CODE

PersonDAO.java:

```
public void insert(Person person) {
    try {
        log.info("person to insert: " + person);
        em = getEntityManager();
        // We start a transaction
        em.getTransaction().begin();
        // We insert the new person
        em.persist(person);
        // We finish the transaction
        em.getTransaction().commit();
    } catch (Exception ex) {
        log.error("Error inserting the object:" + ex.getMessage());
    } finally {
        if (em != null) {
            em.close();
        }
    }
}
```

3. MODIFY THE CODE

PersonDAO.java:

```
public void update(Person person) {
    try {
        log.info("person to update: " + person);
        em = getEntityManager();
        // We start a transaction
        em.getTransaction().begin();
        // We update to the person object
        em.merge(person);
        // We finish the transaction
        em.getTransaction().commit();
    } catch (Exception ex) {
        log.error("Error updating the object:" + ex.getMessage());
    } finally {
        if (em != null) {
            em.close();
        }
    }
}
```

3. MODIFY THE CODE

PersonDAO.java:

```
public void delete(Person person) {
    try {
        log.info("person to delete: " + person);
        em = getEntityManager();
        // We start a transaction
        em.getTransaction().begin();
        // We synchronize and eliminate the person
        em.remove(em.merge(person));
        // We finish the transaction
        em.getTransaction().commit();
    } catch (Exception ex) {
        log.error("Error deleting the object:" + ex.getMessage());
    } finally {
        if (em != null) {
            em.close();
        }
    }
}
```

3. MODIFY THE CODE

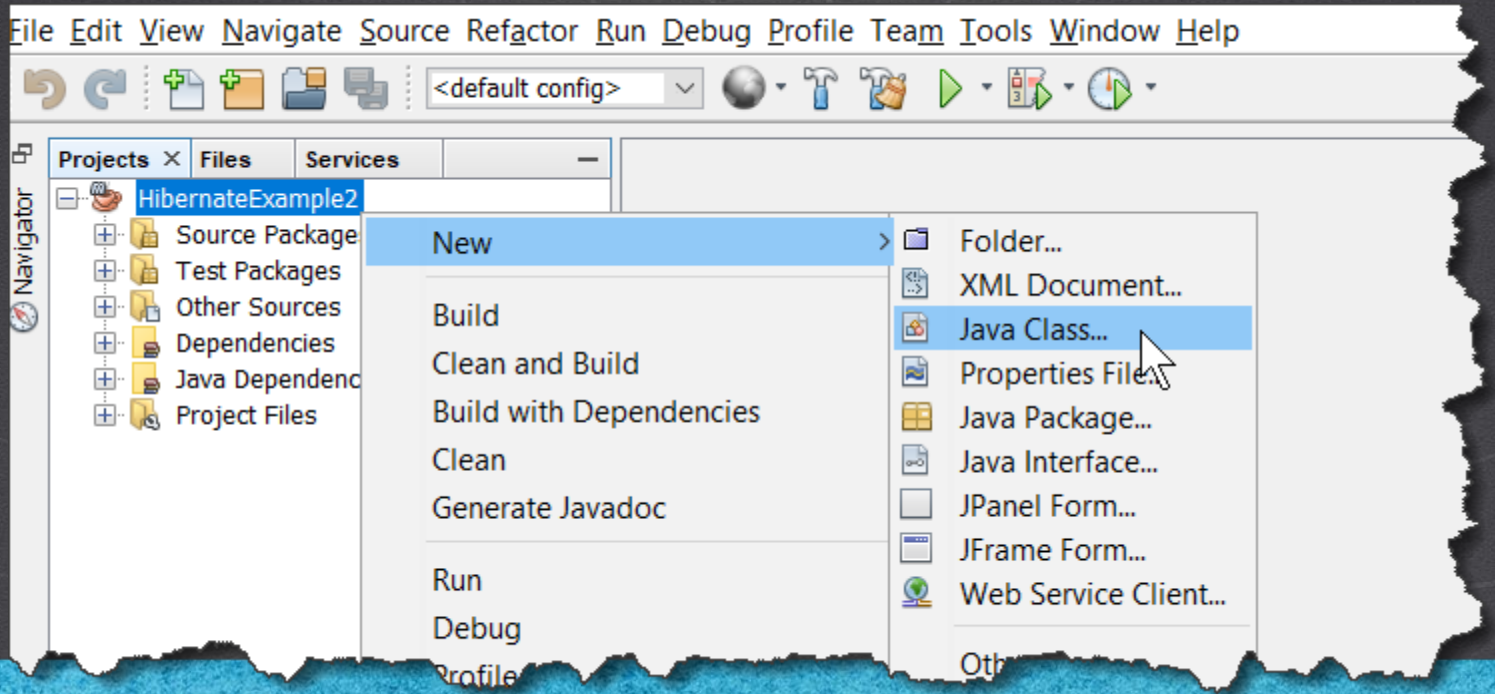
PersonDAO.java:

```
    public Person findById(Person p) {
        log.info("person to find: " + p);
        em = getEntityManager();
        return em.find(Person.class, p.getIdPerson());
    }

}
```

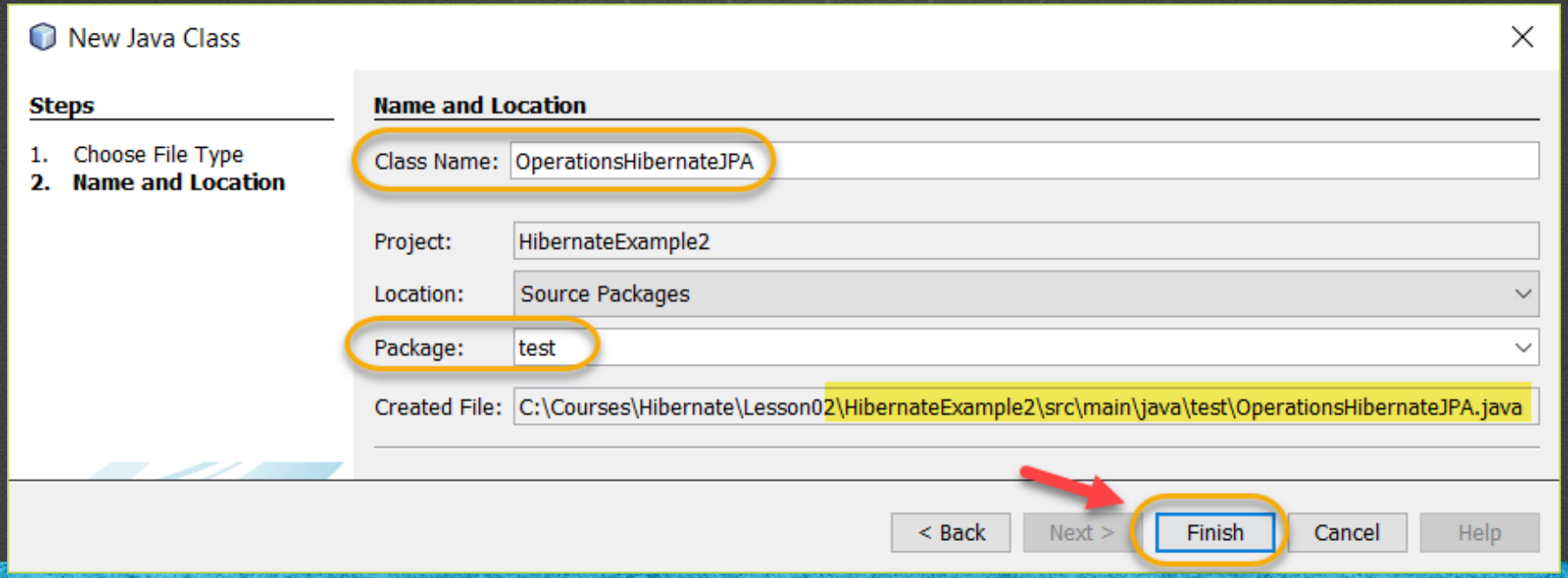
4. CREATE A JAVA CLASS

We create the OperationsHibernateJPA.java class:



4. CREATE A JAVA CLASS

We create the OperationsHibernateJPA.java class:



5. MODIFY THE CODE

OperationsHibernateJPA.java:

```
package test;

import java.util.Scanner;
import dao.PersonDAO;
import domain.Person;

public class OperationsHibernateJPA {

    public static void main(String args[]) {
        PersonDAO personDao = new PersonDAO();
        Person person = null;
        int option = -1;
        Scanner scanner = new Scanner(System.in);
        String idStr, name;
```

5. MODIFY THE CODE

OperationsHibernateJPA.java:

```
// Press 0 to exit
while (option != 0) {
    try {
        System.out.println(
                "Choose an option:\n1.- List People"
                + "\n2.- Find person by id "
                + "\n3.- Add a person"
                + "\n4.- Modify a person\n"
                + "5.- Delete a person\n"
                + "0.- Exit");

        option = Integer.parseInt(scanner.nextLine());

        switch (option) {
            case 1:
                System.out.println("\n1.List People***********");
                personDao.list();;
                break;
```

5. MODIFY THE CODE

OperationsHibernateJPA.java:

```
case 2:
    System.out.println("\n2.Find by id***********");
    System.out.println("Enter the id of the person to search:");
    idStr = scanner.nextLine();
    person = new Person();
    person.setIdPerson(new Integer(idStr));
    person = personDao.findById(person);
    System.out.println("Object found:" + person);
    break;
case 3:
    System.out.println("\n3.Insert**********");
    System.out.println("Enter the name of the person to add:");
    name = scanner.nextLine();
    person = new Person();
    person.setName(name);
    // We save the new object
    personDao.insert(person);
    break;
```

5. MODIFY THE CODE

OperationsHibernateJPA.java:

```
case 4:
    System.out.println("\n4.Modify***********");
    // First we look for the person to modify
    System.out.println("Enter the id of the person to search:");
    idStr = scanner.nextLine();
    person = new Person();
    person.setIdPerson(new Integer(idStr));
    person = personDao.findById(person);
    System.out.println("Enter the name of the person to be modified:");
    name = scanner.nextLine();
    // Modificamos algun valor
    person.setName(name);
    personDao.update(person);
    break;
case 5:
    System.out.println("\n5. Delete***********");
    // First we look for the person to eliminate
    System.out.println("Enter the id of the person to be deleted:");
    idStr = scanner.nextLine();
    person = new Person();
    person.setIdPerson(new Integer(idStr));
    person = personDao.findById(person);
    // Eliminamos el objeto encontrado
    personDao.delete(person);
    break;
```

5. MODIFY THE CODE

OperationsHibernateJPA.java:

```
                    case 0:
                        System.out.println("See you soon!");
                        break;
                    default:
                        System.out.println("Option not recognized");
                        break;
                }
                System.out.println("\n");

            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
    }
}
```

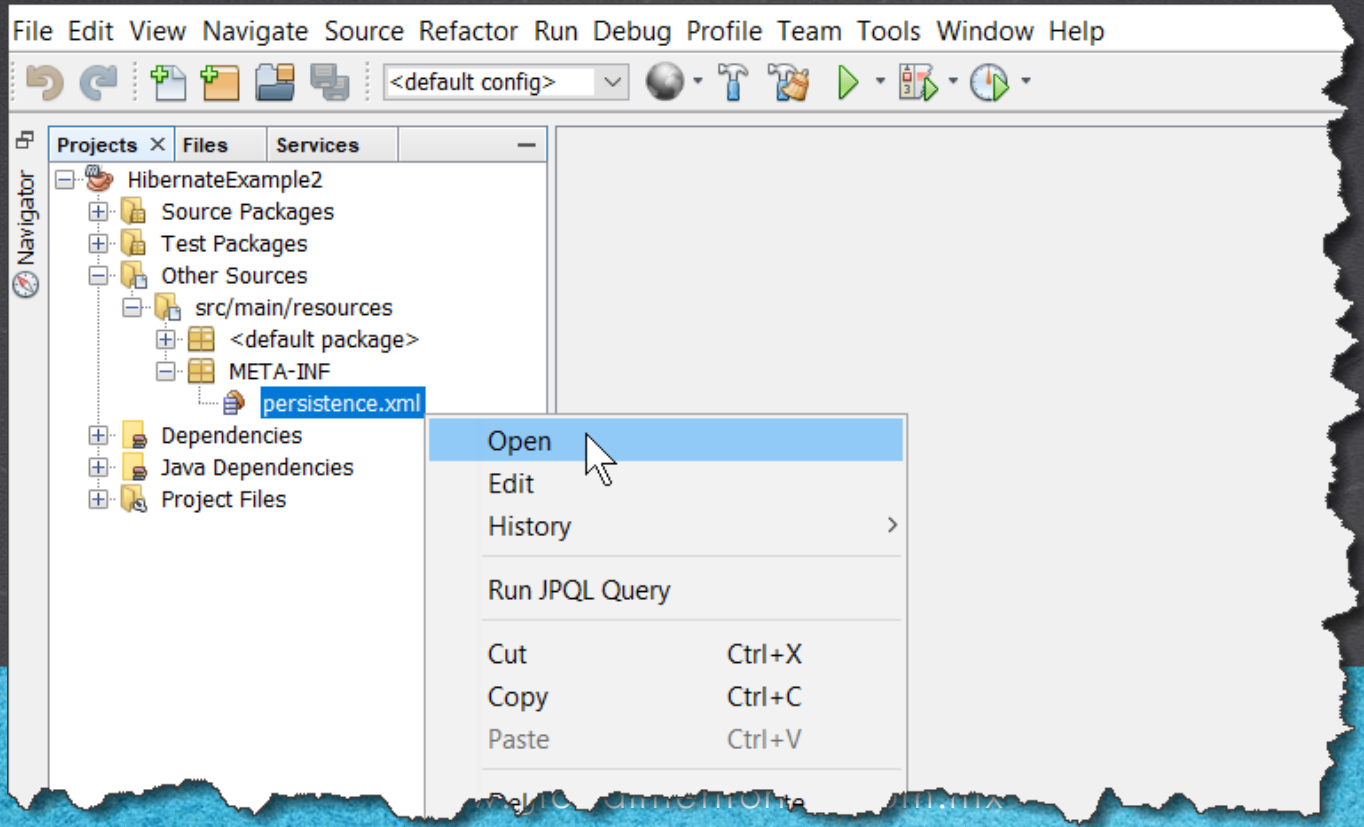
6. MODIFY THE XML FILE

We modified the persistence.xml file so that now the persistence unit is called HibernatePU.

Let's see how our file is:

6. MODIFY THE XML FILE

Open the persistence.xml file to edit it:



6. MODIFY THE XML FILE

Open the persistence.xml file to edit it:

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

<default config>

Navigator

Projects × Files Services

- HibernateExample2
  - Source Packages
  - Test Packages
  - Other Sources
    - src/main/resources
      - <default package>
      - META-INF
        - persistence.xml
  - Dependencies
  - Java Dependencies
  - Project Files

persistence.xml ×

Design Source History General:

Source View

HibernatePU

General:

Persistence Unit Name: HibernatePU

Persistence Library: EclipseLink (JPA 2.1)

JDBC Connection: jdbc:mysql://localhost:3306/test?useSSL=true

Use Java Transaction APIs

Table Generation Strategy: Create Drop and Create None

Validation Strategy: Auto Callback None

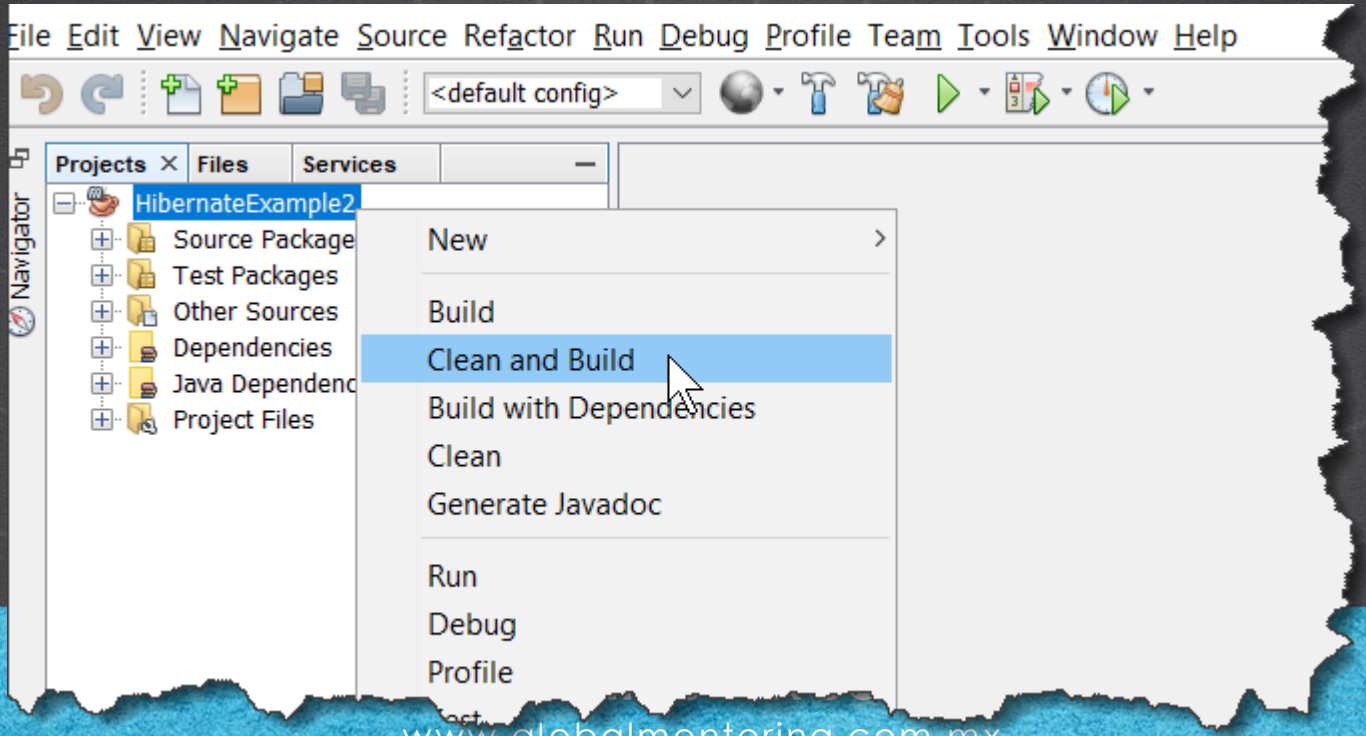
7. MODIFY THE CODE

persistence.xml:

```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="HibernatePU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <class>domain.Person</class>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/test?useSSL=true"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value="admin"/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="hibernate.show_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

8. EXECUTE CLEAN & BUILD

We do a Clean & Build of the project so that we have the latest versions of our newly compiled files:



8. EXECUTE CLEAN & BUILD

We do a Clean & Build of the project so that we have the latest versions of our newly compiled files:

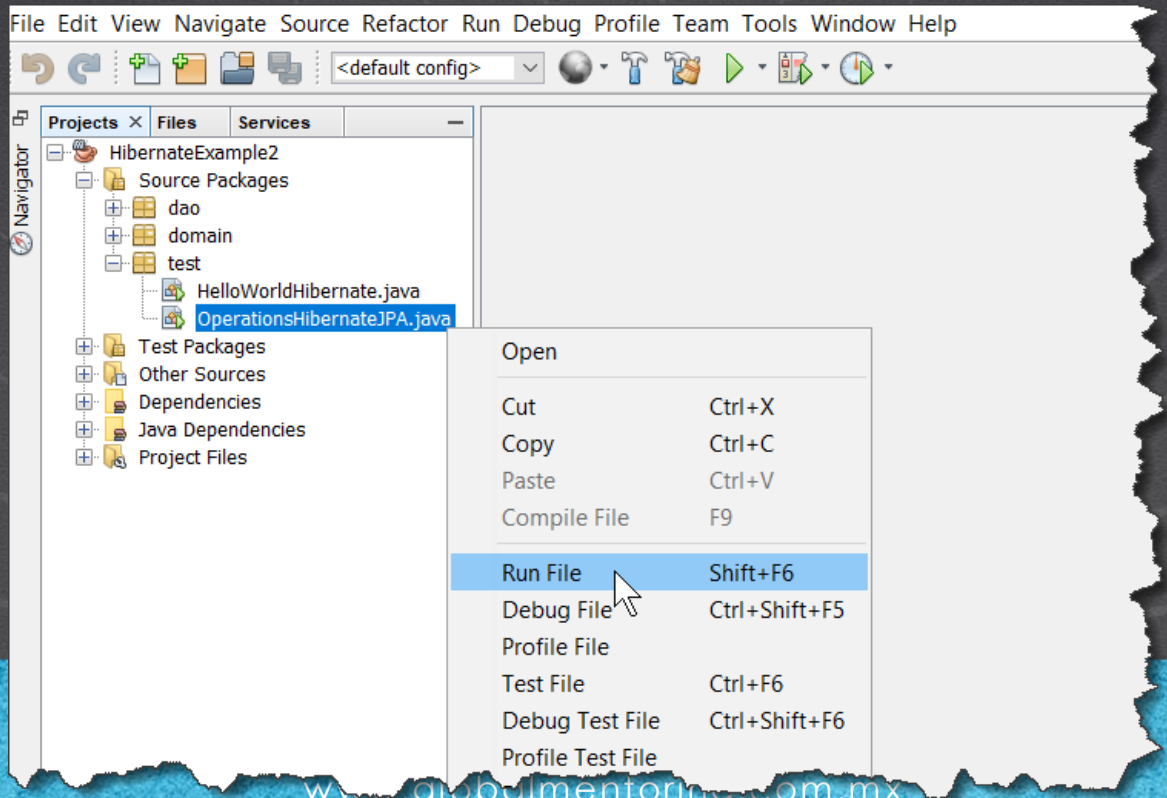
```
Output
Retriever Output   Build (HibernateExample2)

--- maven-jar-plugin:2.3.2:jar (default-jar) @ HibernateExample2 ---
Building jar: C:\Courses\Hibernate\Lesson02\HibernateExample2\target\HibernateExample2-1.jar

--- maven-install-plugin:2.3.1:install (default-install) @ HibernateExample2 ---
Installing C:\Courses\Hibernate\Lesson02\HibernateExample2\target\HibernateExample2-1.jar to C:\Users\user\.m2\repository\data\HibernateExample2\1
Installing C:\Courses\Hibernate\Lesson02\HibernateExample2\pom.xml to C:\Users\user\.m2\repository\data\HibernateExample2\1\HibernateExample2-1.po
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 2.646s
Finished at: Thu Sep 06 23:28:30
Final Memory: 16M/381M
```

9. EXECUTE THE PROJECT

Execute the Project as follows:



9. EXECUTE THE PROJECT

We execute each of the actions and verify the execution in the database used:

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

Projects Files Services

- HibernateExample2
  - Source Packages
    - dao
    - domain
    - test
      - HelloWorldHibernate.java
      - OperationsHibernateJPA.java
  - Test Packages
  - Other Sources
  - Dependencies
  - Java Dependencies
  - Project Files

OperationsHibernateJPA.java

```
public class OperationsHibernateJPA {

    public static void main(String args[]) {
        // Variables generales
        PersonDAO personDao = new PersonDAO();
        Person person = null;
        int option = -1;
        Scanner scanner = new Scanner(System.in);
        String idStr, name;
```

Output

Retriever Output | Run (OperationsHibernateJPA)

```
21:39:08 [main] WARN  org.hibernate.orm.connections.pooling - HHH10001002: Using Hibernate built-in connection pool (not
21:39:08 [main] INFO  org.hibernate.orm.connections.pooling - HHH10001005: using driver [com.mysql.jdbc.Driver] at URL [
21:39:08 [main] INFO  org.hibernate.orm.connections.pooling - HHH10001001: Connection properties: {user=root, password=
21:39:08 [main] INFO  org.hibernate.orm.connections.pooling - HHH10001003: Autocommit mode: false
21:39:08 [main] INFO  org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl - HHH000115: H
21:39:08 [main] INFO  org.hibernate.dialect.Dialect - HHH000400: Using dialect: org.hibernate.dialect.MySQL57Dialect
Choose an option:
1.- List People
2.- Find person by id
3.- Add a person
4.- Modify a person
5.- Delete a person
0.- Exit
```

10. STOP THE PROGRAM

If necessary, we stop the execution of the program :

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

Projects | Files | Services

- HibernateExample2
  - Source Packages
    - dao
    - domain
    - test
      - HelloWorldHibernate.java
      - OperationsHibernateJPA.java
  - Test Packages
  - Other Sources
  - Dependencies
  - Java Dependencies
  - Project Files

OperationsHibernateJPA.java

Source | History

```
6
7   public class OperationsHibernateJPA {
8
9       public static void main(String args[]) {
10          // Variables generales
11          PersonDAO personDao = new PersonDAO();
12          Person person = null;
13          int option = -1;
14          Scanner scanner = new Scanner(System.in);
15          String idStr, name;
16
17
36                  System.out.println("\nE.Find by id**********");
```

Output

Retriever Output | Run (OperationsHibernateJPA)

```
21:39:08 [main] WARN  org.hibernate.orm.connections.pooling - HHH10001002: Using Hibernate built-in connection pool (not
21:39:08 [main] INFO  org.hibernate.orm.connections.pooling - HHH10001005: using driver [com.mysql.jdbc.Driver] at URL [
21:39:08 [main] INFO  org.hibernate.orm.connections.pooling - HHH10001001: Connection properties: {user=root, password=
21:39:08 [main] INFO  org.hibernate.orm.connections.pooling - HHH10001003: Autocommit mode: false
21:39:08 [main] INFO  org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl - HHH000115: Hi
21:39:08 [main] INFO  org.hibernate.dialect.Dialect - HHH000400: Using dialect: org.hibernate.dialect.MySQL57Dialect
Choose an option:
1.- List People
2.- Find person by id
3.- Add a person
4.- Modify a person
5.- Delete a person
0.- Exit
```

EXERCISE CONCLUSION

- With this exercise we have put into practice the basic operations with Hibernate and JPA.
- Operations such as list, search by id, insert, modify and delete.
- As we move forward we will review each of these operations in more detail, as well as the code used, but with this exercise we already have a very good idea of the great help that Hibernate and JPA provides us for both persisting as well as reading information in the database. of data.

ONLINE COURSE

HIBERNATE & JPA

By: Eng. Ubaldo Acosta