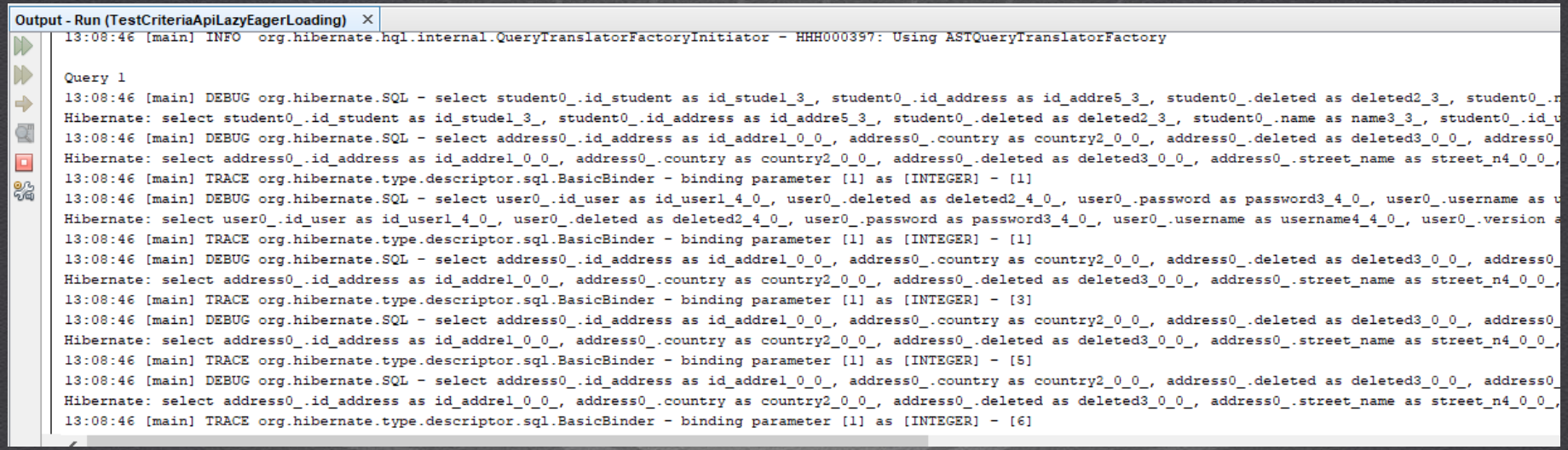
HIBERNATE & JPA COURSE

EXERCISE

CRITERIA API
LAZY & EAGER LOADING

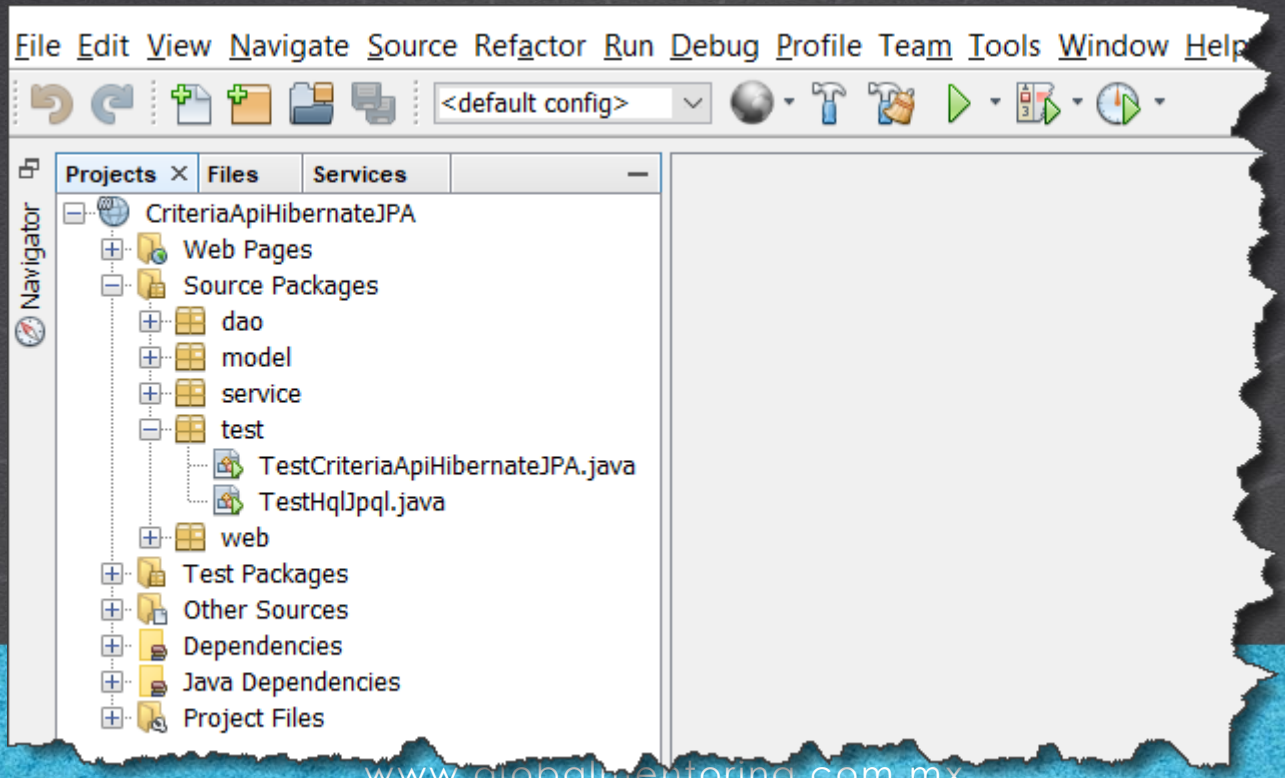
EXERCISE OBJECTIVE

Use the Criteria API to implement the concept of Lazy and Eager Loading. At the end we should observe the following:



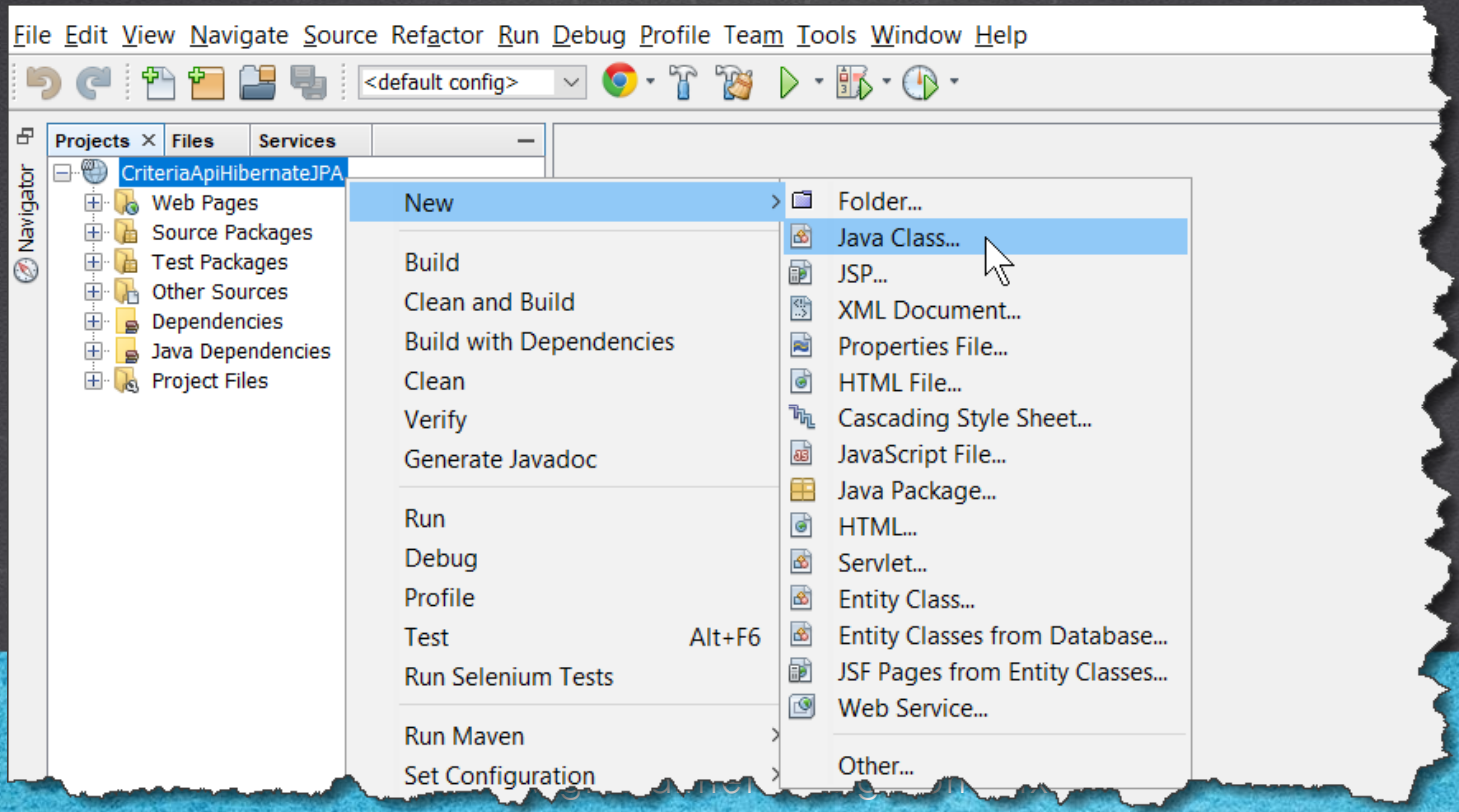
1. USE THE PROJECT

We open the CriteriaApiHibernateJPA project:



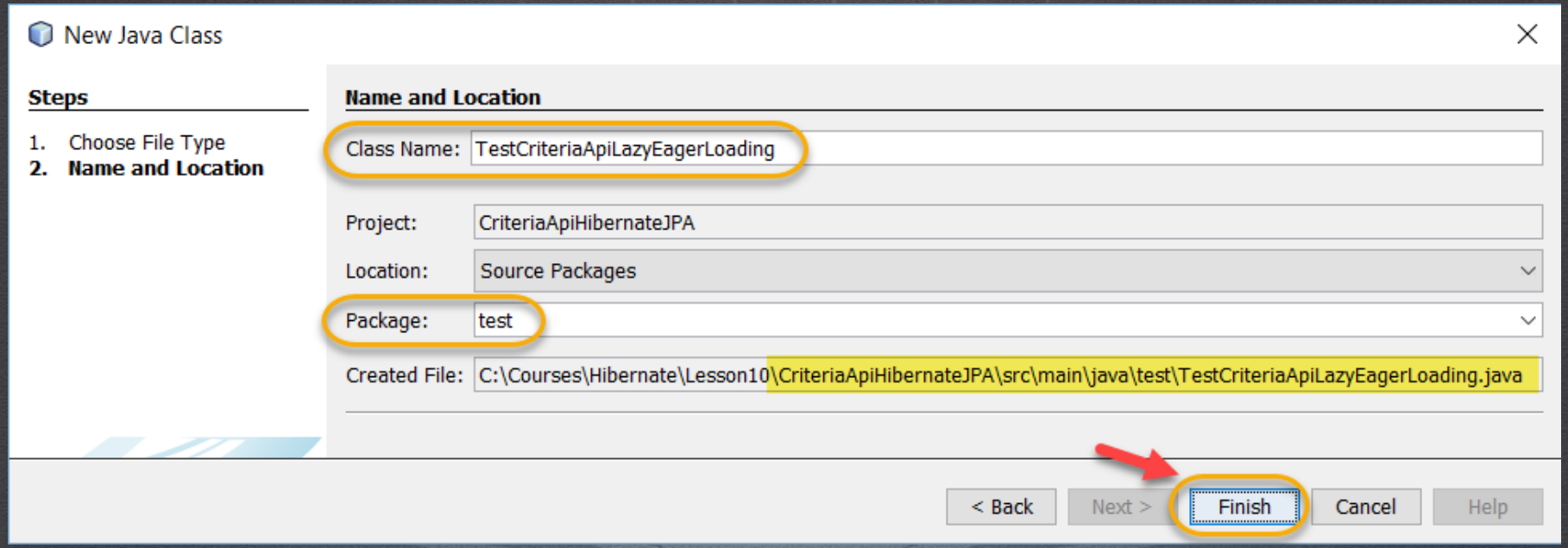
2. CREATE A CLASS

We create the class TestCriteriaApiLazyEagerLoading.java:



2. CREATE A CLASS

We create the class TestCriteriaApiLazyEagerLoading.java:



3. MODIFY THE CODE

TestCriteriaApiLazyEagerLoading.java:

Click to download

```
package test;

import java.util.*;
import javax.persistence.*;
import javax.persistence.criteria.*;
import model.*;

public class TestCriteriaApiLazyEagerLoading {

    public static void main(String[] args) {
        EntityManagerFactory fabrica = Persistence.createEntityManagerFactory("HibernateJpaPU");
        EntityManager em = fabrica.createEntityManager();

        //Help variables
        CriteriaBuilder cb = em.getCriteriaBuilder();
        List<Student> students = null;

        //By default the queries are lazy type, that is, they do not
        //recover the Left Join relations with the Criteria API
        //Query 1
        System.out.println("\nQuery 1");
        CriteriaQuery<Student> qb1 = cb.createQuery(Student.class);
        Root<Student> c1 = qb1.from(Student.class);
        c1.join("address", JoinType.LEFT);

        students = em.createQuery(qb1).getResultList();
        printStudents(students);
```

3. MODIFY THE CODE

TestCriteriaApiLazyEagerLoading.java:

Click to download

```
//Query 2
System.out.println("\nQuery 2");
//We define the query
CriteriaQuery<Student> qb2 = cb.createQuery(Student.class);
//We define the query root
Root<Student> c2 = qb2.from(Student.class);
//We specify the join
Join<Student, Address> add = c2.join("address");
//Optionally add the restriction using the join
qb2.where(cb.equal(add.<Integer>get("idAddress"), 1));
//We define an Entity Graph to specify the Fetch join
EntityGraph<Student> fetchGraph = em.createEntityGraph(Student.class);
//We specify the relationship to be raised in an eager way (anticipated)
fetchGraph.addSubgraph("address");
//loadgraph adds the definition plus what is already specified in the
//entity class. fetchgraph ignores what is defined in the entity class
//and adds only the new
em.createQuery(qb2).setHint("javax.persistence.loadgraph", fetchGraph);

TypedQuery<Student> q2 = em.createQuery(qb2);
students = q2.getResultList();
printStudents(students);
```

3. MODIFY THE CODE

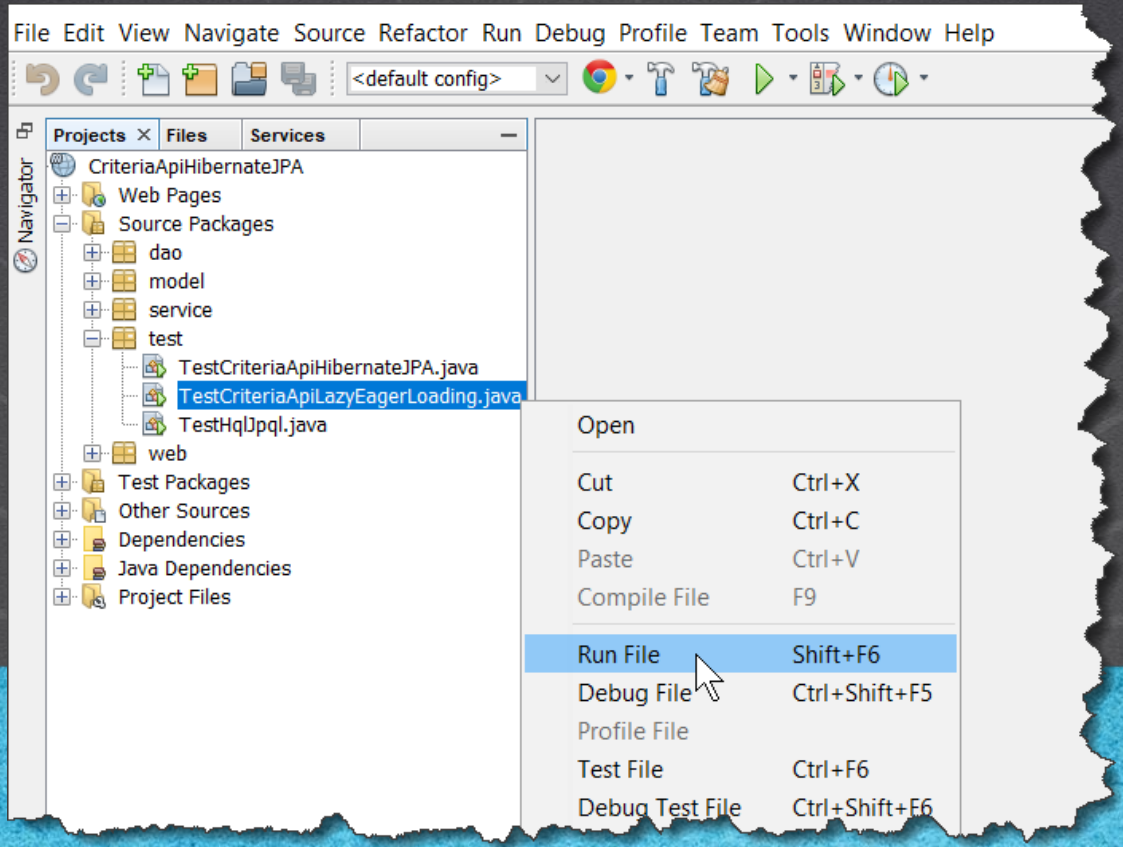
TestCriteriaApiLazyEagerLoading.java:

Click to download

```
        // Query 3
        System.out.println("\nQuery 3");
        CriteriaQuery<Student> qb3 = cb.createQuery(Student.class);
        Root<Student> c3 = qb3.from(Student.class);
        Join<Student, Address> address = c3.join("address");
        List<Predicate> conditions = new ArrayList();
        Integer idStudent = 1;
        conditions.add(cb.equal(c3.get("idStudent"), idStudent));
        conditions.add(cb.isNotNull(address.get("streetNumber")));

        TypedQuery<Student> q3 = em.createQuery(
                qb3
                .select(c3)
                .where(conditions.toArray(new Predicate[]{}))
                .orderBy(cb.asc(c3.get("idStudent")))
                .distinct(true)
        );

        printStudents(q3.getResultList());
    }

    private static void printStudents(List<Student> students) {
        for (Student a : students) {
            System.out.println(a);
        }
    }
}
```

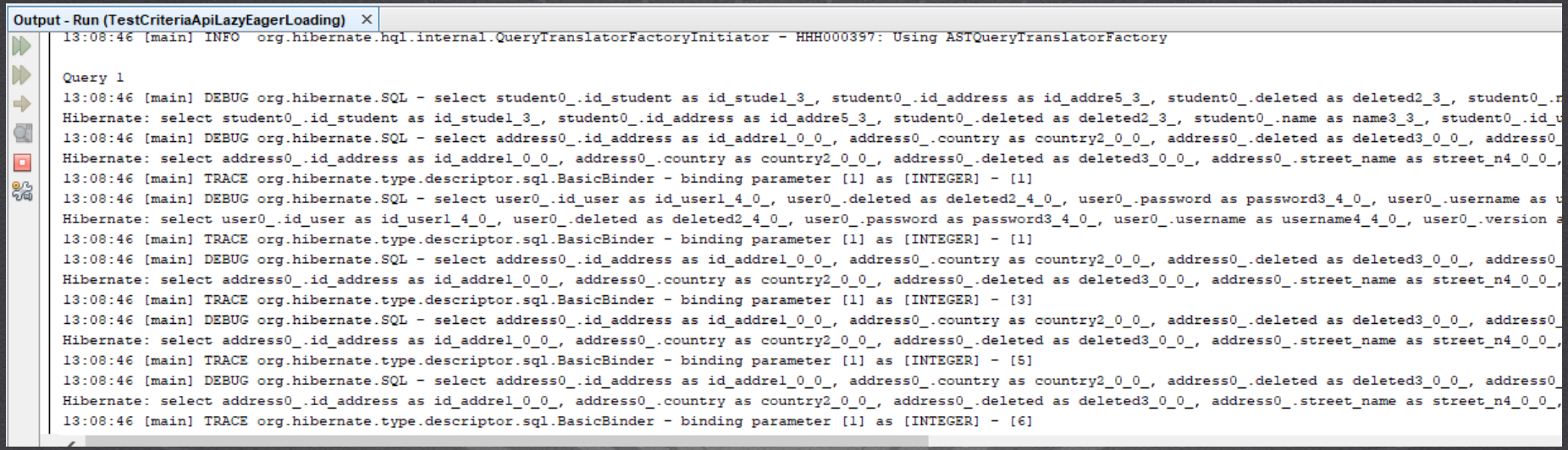
4.EXECUTE THE PROJECT

We execute each of the queries of the project:



4. EXECUTE THE PROJECT

We execute each of the queries of the project:



EXERCISE CONCLUSION

- With this exercise we have executed several of the queries with the Criteria API of Hibernate / JPA applying the concept of Fetch or Eager Loading.
- By default the queries are Lazy type in Hibernate / JPA, however we have seen how to load the relationships in advance (eager) by applying the concept of Fetch to the queries.
- With this we can already compare and decide if we use the HQL / JPQL language or the Criteria API.
- Each one has its advantages and disadvantages, but everything will depend on what we need in our application to know if we use another solution.

ONLINE COURSE

HIBERNATE & JPA

By: Eng. Ubaldo Acosta