GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

Unit 3. ACCESS USING OBJECT-RELATIONAL MAPPING (ORM)

Part 2. Hibernate using annotations

Acceso a Datos (ADA) (a distancia en inglés)
CFGS Desarrollo de Aplicaciones Multiplataforma (DAM)

Abelardo Martínez
Year 2024-2025

Credits

- Notes made by Abelardo Martínez.
- Based and modified from Sergio Badal (www.sergiobadal.com).
- The images and icons used are protected by the LGPL licence and have been obtained from:
  - https://commons.wikimedia.org/wiki/Crystal_Clear
  - https://www.openclipart.org

Contents

1. USING ANNOTATIONS

What is Hibernate annotations?

So far you have seen how Hibernate uses XML mapping file for the transformation of data from POJO to database tables and vice versa.

- Hibernate annotations are the **newest way to define mappings without the use of XML file**. You can use annotations in addition to or as a replacement of XML mapping metadata.
- Hibernate annotations is the powerful way to provide the metadata for the Object and Relational Table mapping. **All the metadata is clubbed into the POJO java file along with the code**, this helps the user to understand the table structure and POJO simultaneously during the development.

If you are going to make your application portable to other ORM applications, **you must use annotations to represent the mapping information**, but still if you want greater flexibility, then you should go with XML-based mappings.

For further information: https://www.tutorialspoint.com/hibernate/hibernate_annotations.htm

How to include annotations

1)First of all you would have to make sure that you are using **JDK 5.0 (or higher)** to take advantage of the native support for annotations.

2)Second, you will need to **install the Hibernate annotations distribution package**, available from the sourceforge and copy hibernate-annotations.jar, lib/hibernate-comons-annotations.jar and lib/ejb3-persistence.jar from the Hibernate Annotations distribution to your CLASSPATH.

Or… let Maven do the work for us.

Using Maven, you just need to add this new dependency to your pom.xml file:

```
<!--https://central.sonatype.com/artifact/org.hibernate/hibernate-annotations-->
   <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-annotations</artifactId>
      <version>3.5.6-Final</version>
   </dependency>
```

Maven repository: https://mvnrepository.com/artifact/org.hibernate/hibernate-annotations

Maven central: https://central.sonatype.com/artifact/org.hibernate/hibernate-annotations

2. UPGRADING EXAMPLE

Upgrading to annotations

Last week we created an application using XML files to map the tables. With annotations, we won’t need those files anymore.

Instead we’ll have to:

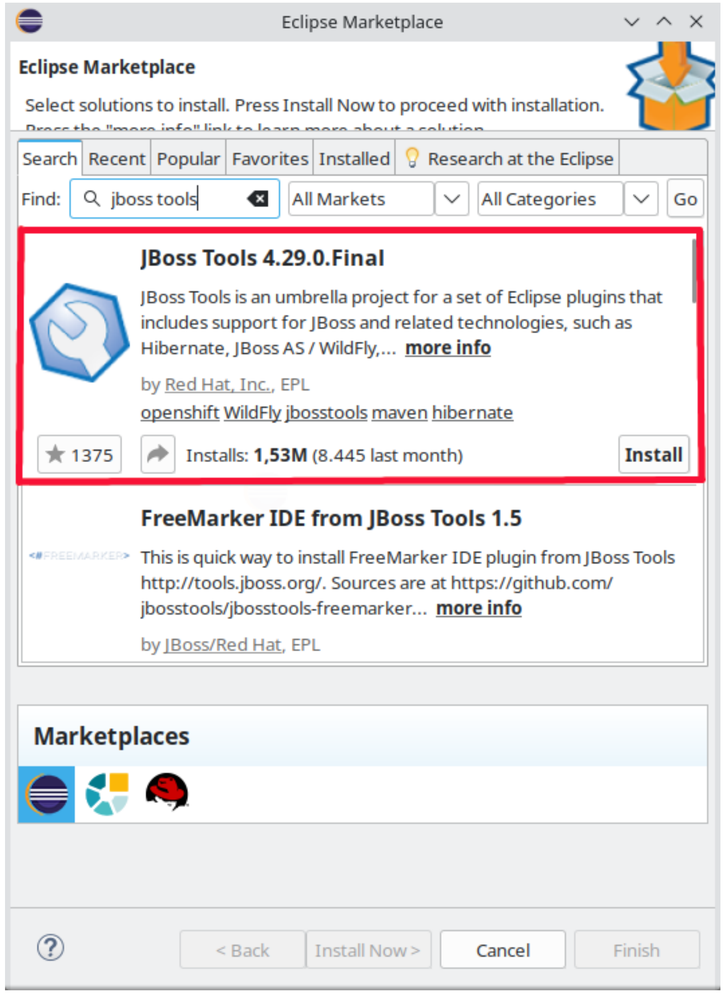
1) Set the database columns directly within our POJO files.
2) Set the relationships directly within our POJO files.
3) Change the hibernate cfg file to set this new mapping type.

Set-up Hibernate automatically

Now we could set-up Hibernate in an automatic or manual way. If we choose to **set-up automatically**, we must install the **Jboss tools plugin** and follow the instructions explained in the extended materials.

Although it is most common to configure Hibernate automatically, the following slides will explain the manual procedure, as it helps to better understand the ORM mapping concepts.

The extended materials detail how to perform the same process automatically using the “JBoss tools” plugin.



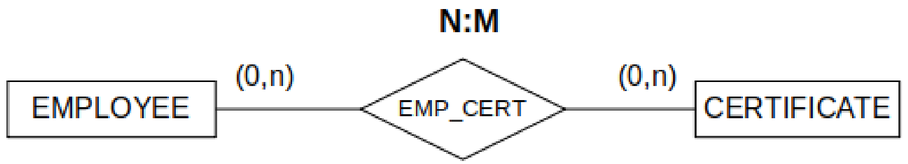
The (MySQL) database

We will use the same DB seen in part 1 of the Hibernate classic topic.

```
CREATE DATABASE ADAU3DBExample CHARACTER
SET utf8mb4 COLLATE utf8mb4_es_0900_ai_ci;

CREATE USER mavenuser@localhost IDENTIFIED
BY 'ada0486';
GRANT ALL PRIVILEGES ON ADAU3DBExample.* to
mavenuser@localhost;

USE ADAU3DBExample;
```

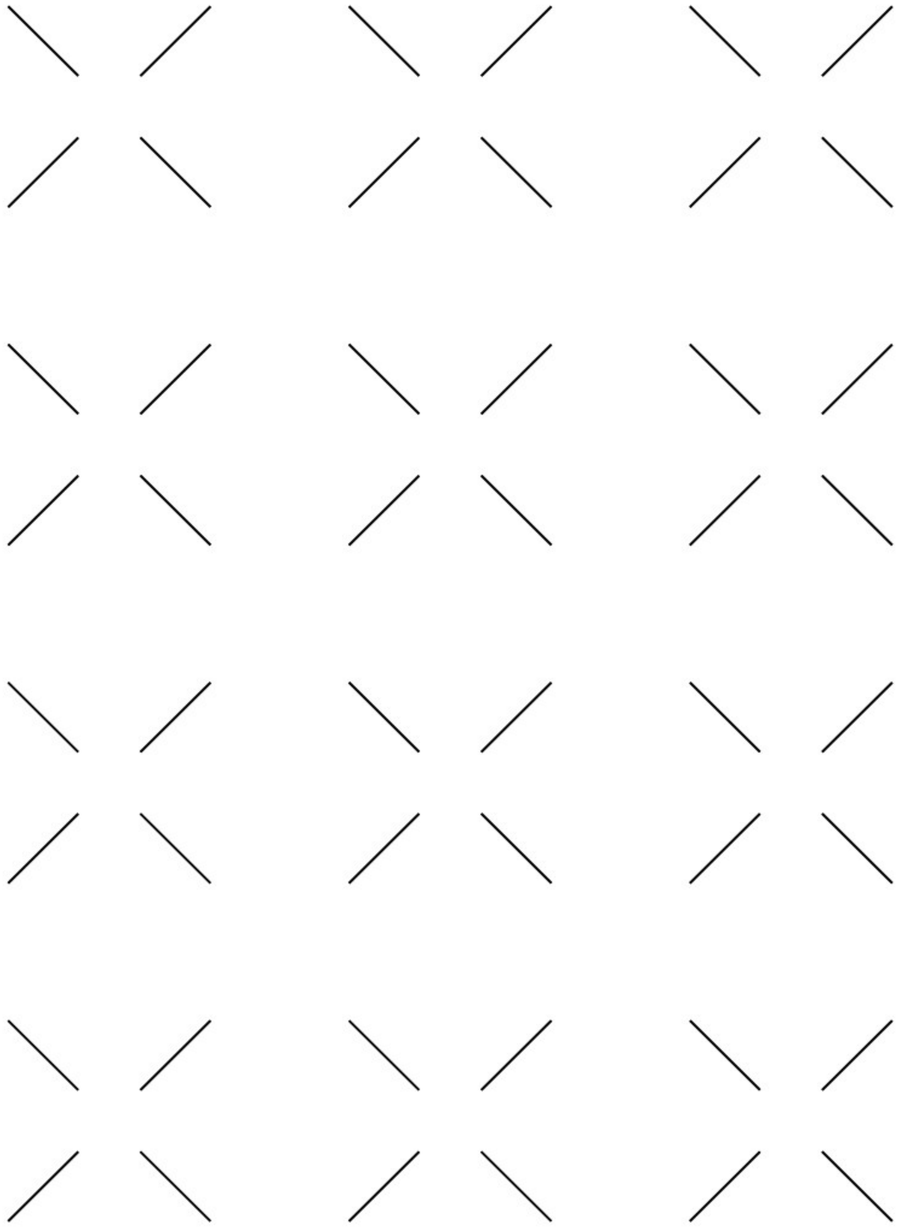



```
CREATE TABLE Employee (
empID        INTEGER PRIMARY KEY AUTO_INCREMENT,
firstname    VARCHAR(20),
lastname     VARCHAR(20),
salary       FLOAT
);

CREATE TABLE Certificate (
certID       INTEGER PRIMARY KEY AUTO_INCREMENT,
certname     VARCHAR(30)
);

CREATE TABLE EmpCert (
employeeID     INTEGER,
certificateID  INTEGER,
PRIMARY KEY (employeeID, certificateID),
CONSTRAINT emp_id_fk FOREIGN KEY (employeeID)
REFERENCES Employee(empID),
CONSTRAINT cer_id_fk FOREIGN KEY (certificateID)
REFERENCES Certificate(certID)
);
```

2.1 Upgrading. Step 1



Upgrading. Step 1

1) Set the database columns directly within our POJO files. Now we should write the annotations into the several POJO files.

Remember to be careful about the name of the variables. The getters and setters must follow the same criteria to allow Hibernate to find the appropriate methods:
https://stackoverflow.com/questions/921239/hibernate-propertynotfoundexception-could-not-find-a-getter-for

Hibernate 6 moves from Java Persistence as defined by the Java EE specs to Jakarta Persistence as defined by the Jakarta EE spec. The most immediate impact of this change is that applications would need to be updated to **use the Jakarta Persistence classes (jakarta.persistence.*)** instead of the Java Persistence ones (javax.persistence.*).
https://docs.jboss.org/hibernate/orm/6.0/migration-guide/migration-guide.html#_jakarta_persistence

POJO Employee


```java
public class Employee {

    // ATTRIBUTES

   private int iEmpID;
   private String stFirstName;
   private String stLastName;
   private double dSalary;
```

We should remove @GeneratedValue when the ID field (PRIMARY KEY) will be set manually

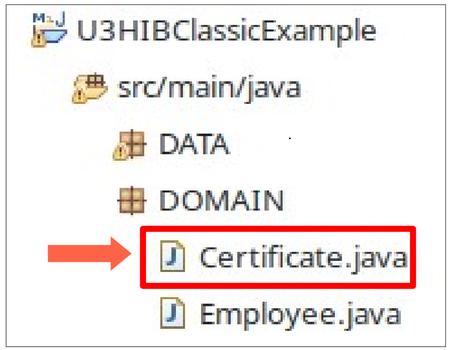
```java
import jakarta.persistence.*;

@Entity
@Table(name = "Employee")
public class Employee {

   /*
    * -------------
    * ATTRIBUTES
    * -------------
    */

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   @Column(name = "empID")
   private int iEmpID;
   @Column(name = "firstname")
   private String stFirstName;
   @Column(name = "lastname")
   private String stLastName;
   @Column(name = "salary")
   private double dSalary;
```

POJO Certificate



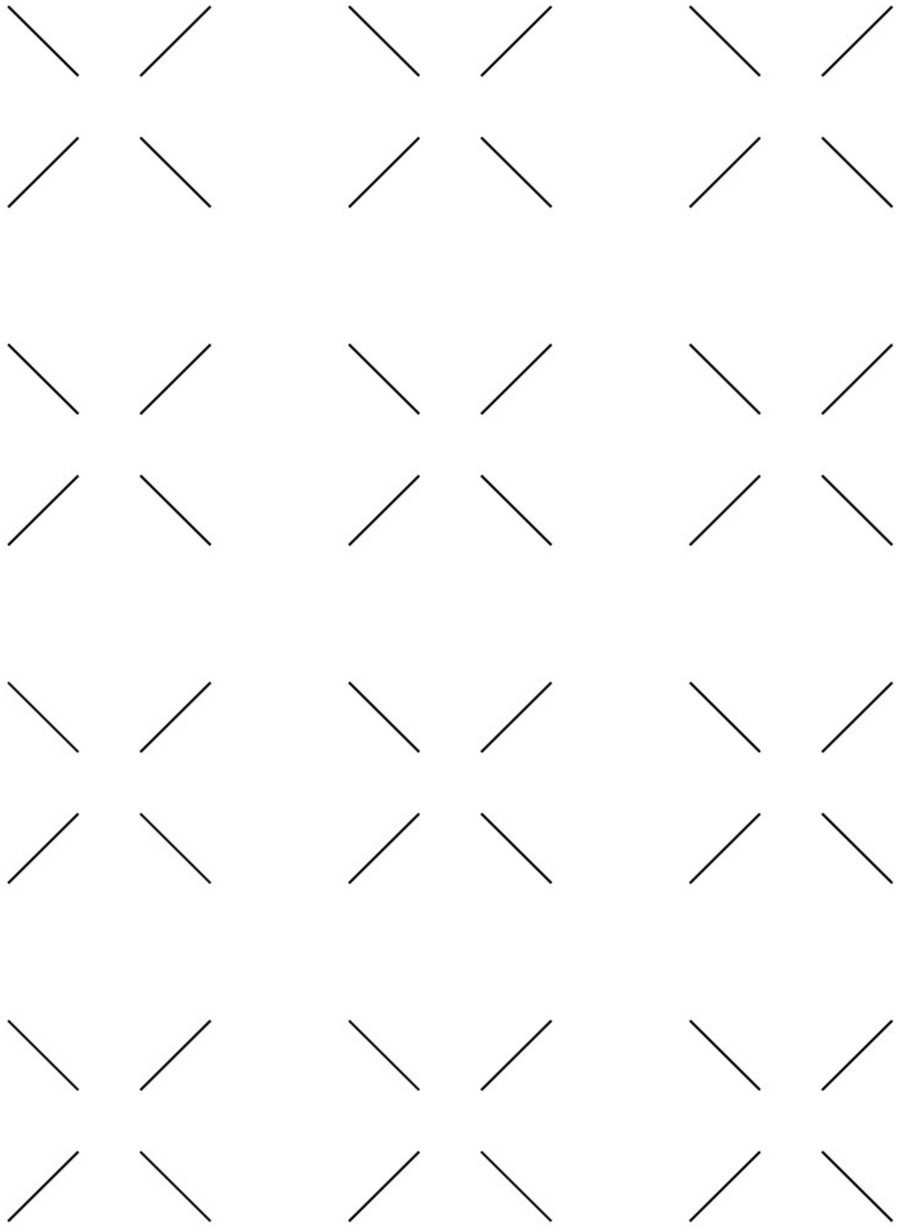

```java
import jakarta.persistence.*;

@Entity
@Table(name = "Certificate")
public class Certificate {

    // ATTRIBUTES

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   @Column(name = "certID")
   private int iCertID;
   @Column(name = "certname")
   private String stCertName;
```

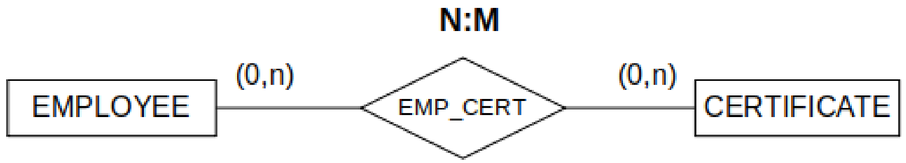
2.2 Upgrading. Step 2



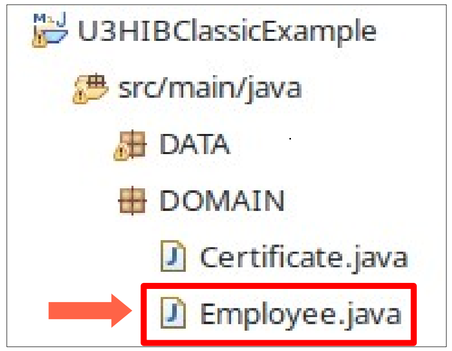
Upgrading. Step 2

2) Set the relationships directly within our POJO files

There are several approaches depending on the type of the relationship (1:1, 1:N, N:M). You can check them all here:
https://www.javatpoint.com/hibernate-many-to-many-example-using-annotation



POJO Employee



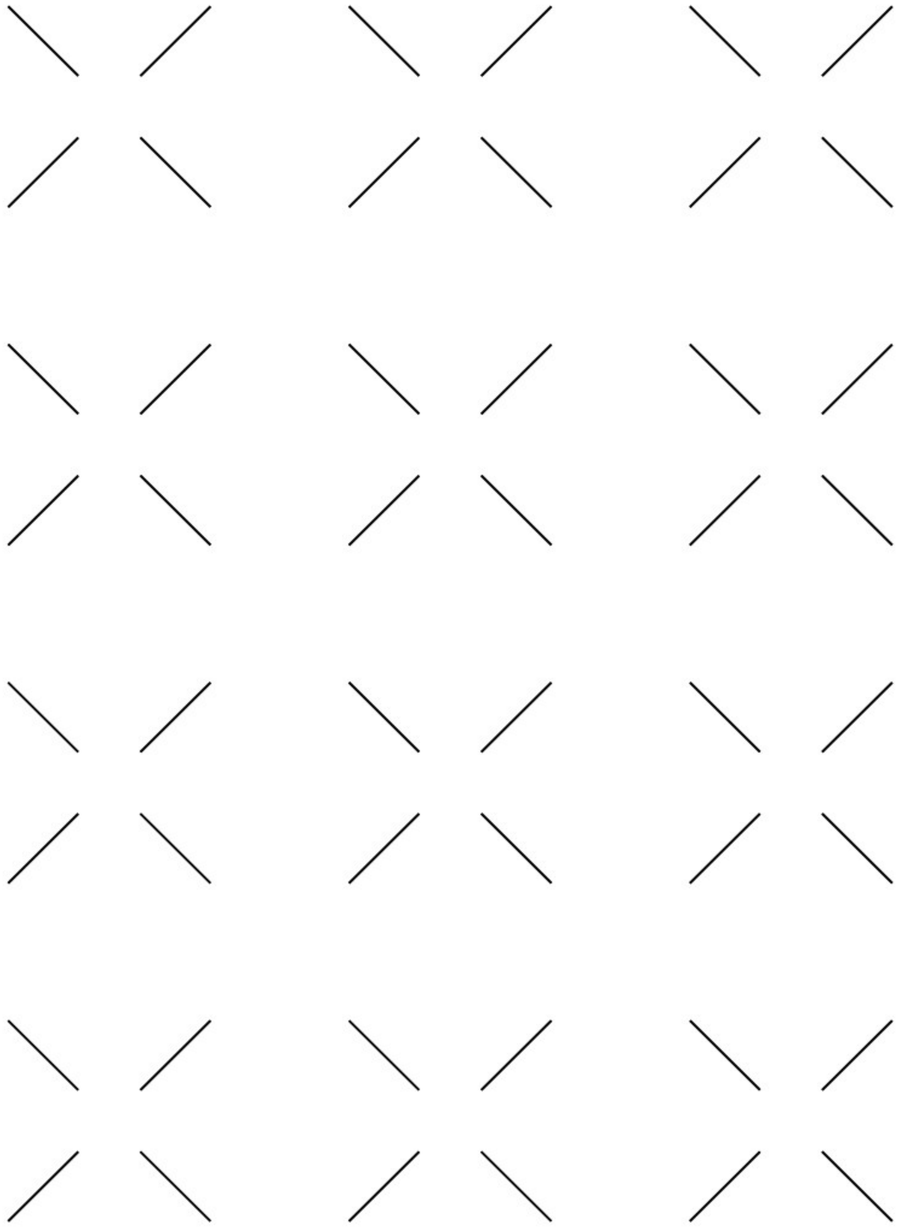

```java
import java.util.Set;
import jakarta.persistence.*;
import org.hibernate.annotations.OnDelete;
import org.hibernate.annotations.OnDeleteAction;

@Entity
@Table(name = "Employee")
public class Employee {

    // ATTRIBUTES

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   @Column(name = "empID")
   private int iEmpID;
   @Column(name = "firstname")
   private String stFirstName;
   @Column(name = "lastname")
   private String stLastName;
   @Column(name = "salary")
   private double dSalary;
   @ManyToMany(targetEntity = Certificate.class)
   @JoinTable(name = "EmpCert", joinColumns = { @JoinColumn(name = "employeeID") },
inverseJoinColumns = { @JoinColumn(name = "certificateID") })
   @OnDelete(action = OnDeleteAction.CASCADE)
   //Set class in Java
   private Set<Certificate> relCertificates; //relationship Employee-Certificate (N:M)
```

2.3 Upgrading. Step 3



Upgrading. Step 3

3) Change the hibernate cfg file to set this new mapping type

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
      <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ADAU3DBExample</property>
         <property name="hibernate.connection.username">mavenuser</property>
         <property name="hibernate.connection.password">ada0486</property>
      <property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>
      <property name="show_sql">false</property>
      <property name="format_sql">true</property>
      <property name="hbm2ddl.auto">update </property>
      <!-- mapping resource="employee.hbm.xml" / -->
      <!-- mapping resource="certificate.hbm.xml" / -->
      <!-- https://www.javatpoint.com/hibernate-many-to-many-example-using-annotation -->
      <mapping class="DOMAIN.Employee" />
      <mapping class="DOMAIN.Certificate" />
   </session-factory>
</hibernate-configuration>
```

3. ACTIVITIES FOR NEXT WEEK

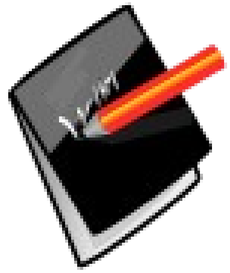
Proposed activities

Check the suggested exercises you will find at the “Aula Virtual”. **These activities are optional and non-assessable but** understanding these non-assessable activities is essential to solve the assessable task ahead.

Shortly you will find the proposed solutions.

4. BIBLIOGRAPHY

Resources



- Tutorialspoint. Hibernate tutorial. https://www.tutorialspoint.com/hibernate/index.htm
- An Introduction to Hibernate 6.

https://docs.jboss.org/hibernate/orm/6.3/introduction/html_single/Hibernate_Introduction.html#queries

- Hibernate Annotations.
  https://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/
- Tutorialspoint. Hibernate – Annotations.
  https://www.tutorialspoint.com/hibernate/hibernate_annotations.htm
- Josep Cañellas Bornas, Isidre Guixà Miranda. Accés a dades. Desenvolupament d'aplicacions multiplataforma. Creative Commons. Departament d'Ensenyament, Institut Obert de Catalunya. Dipòsit legal: B. 29430-2013. https://ioc.xtec.cat/educacio/recursos
- Alberto Oliva Molina. Acceso a datos. UD 3. Herramientas de mapeo objeto relacional (ORM). IES Tubalcaín. Tarazona (Zaragoza, España).