

Unidad 3. ACCESO UTILIZANDO MAPEO RELACIONAL DE OBJETOS (ORM)

Parte 1. Acceso usando Hibernate Classic

Acceso a Datos (ADA) (a distancia en inglés)

CFGs Desarrollo de Aplicaciones Multiplataforma (DAM)

Abelardo Martinez

Año 2023-2024



Créditos

- Apuntes realizados por Abelardo

Martínez. •Basado y modificado de Sergio Badal

(www.sergiobadal.com). •Las imágenes e iconos utilizados están protegidos por la [LGPL](#) . licencia y haber sido de:

- https://commons.wikimedia.org/wiki/Crystal_Clear

- <https://www.openclipart.org>

Progreso de la unidad



Contenido

1. ¿QUÉ ES ORM?

1. Introducción

2.POJO en Java

3.ORM. Hibernar

2. ELEMENTOS DE UN PROYECTO DE HIBERNACIÓN

3. CONFIGURACIÓN DEL PROYECTO Y LA BASE DE DATOS

4. CONFIGURAR LA HIBERNACIÓN

5. HIBERNAR: SESIONES

6. HIBERNAR: MAPEO DE UNA ÚNICA TABLA

1.archivo DAO

2.Util Hibernar

3.Prueba de hibernación

7. HIBERNAR: MAPEO AVANZADO

8. ACTIVIDADES PARA LA PRÓXIMA SEMANA

9. BIBLIOGRAFÍA

1. ¿QUÉ ES ORM?

1.1 Introducción

ORM (mapeo objeto-relacional)

1) DAO. Un objeto de acceso a datos es un patrón que a menudo se sigue cuando una aplicación necesita interactuar con algún almacén de datos persistente (a menudo una base de datos). El DAO proporciona una serie de operaciones al resto de la aplicación sin que la aplicación necesite conocer los detalles del almacén de datos.

2) ORM. Un ORM generalmente describe una biblioteca/API más robusta que se utiliza para realizar interacciones con una base de datos.

En resumen, un DAO es un objeto que abstrae la implementación de un almacén de datos persistente fuera de la aplicación y permite una interacción simple con ella.

Un ORM es una biblioteca/API robusta que proporciona un montón de herramientas para guardar/recuperar un objeto directamente desde/hacia la base de datos sin tener que escribir sus propias declaraciones SQL.

El patrón de objeto de acceso a datos (DAO) es un patrón estructural que nos permite aislar la capa de aplicación/negocio de la capa de persistencia utilizando una API abstracta.

El mapeo relacional de objetos (ORM) es una técnica para convertir datos entre sistemas de tipos incompatibles utilizando lenguajes de programación orientados a objetos.

DAO y ORM

DAO. Hay varias formas de encapsular entidades de bases de datos en clases (DAO). Los dos proyectos DAO sobre Java más utilizados son:

- **POJO** 
- JavaBeans

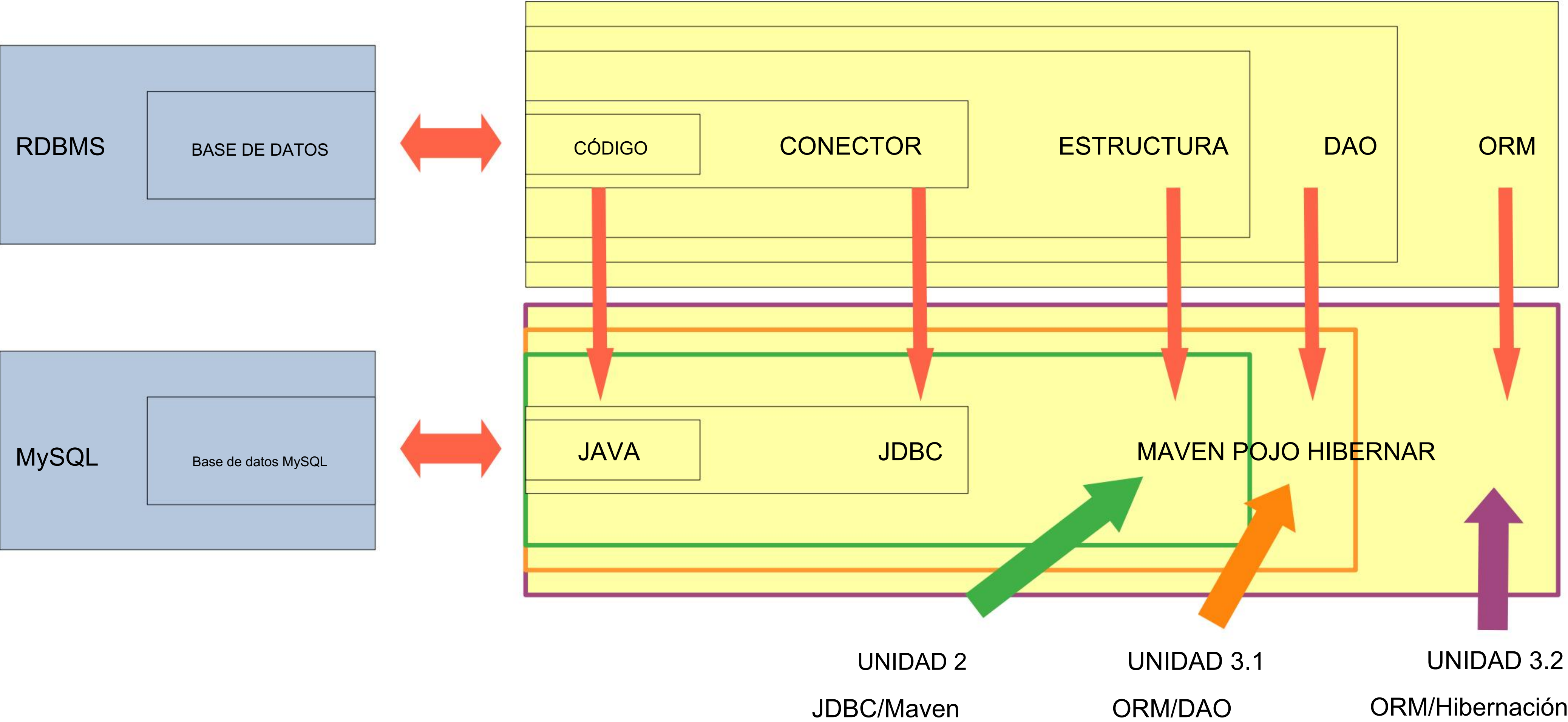
ORM. Hay varias formas de construir el "puente" entre la base de datos y el código (ORM). Los proyectos ORM sobre Java más utilizados son:

- **Hibernate** 
- Primavera

¡Usaremos esos!

Capas de encapsulación

Lo que estamos haciendo es conectar el código con la base de datos mediante el uso de capas, herramientas y marcos para hacerlo más fácil. Usaremos estos:



1.2 POJO en Java

POJO en Java

POJO en Java significa Objeto Java antiguo simple . Es un objeto ordinario, que no está sujeto a ninguna restricción especial. El archivo POJO no requiere ninguna ruta de clase especial. Aumenta la legibilidad y reutilización de un programa Java.

Los POJO ahora son ampliamente aceptados debido a su fácil mantenimiento. Son fáciles de leer y escribir.

Una clase POJO no tiene ninguna convención de nomenclatura para propiedades y métodos. No está vinculado a ningún marco de Java; cualquier programa Java puede usarlo.

Se introdujo el término POJO.

por Martin Fowler (un desarrollador de software estadounidense) en 2000.



Presenta encapsulación POJO

Propiedades de la clase POJO:

- La clase POJO debe ser pública.
- Debe tener un constructor público predeterminado.
- Puede tener el constructor de argumentos.
- Todos los objetos deben tener algunos Getters y Setters públicos para acceder al valores de objeto por otros programas Java.
- El objeto en la Clase POJO puede tener cualquier acceso, como privado, público o protegido. Sin embargo, todas las variables de instancia deben ser privadas para mejorar la seguridad del proyecto.
- Una clase POJO no debe extender clases predefinidas.
- No debe implementar interfaces preespecificadas.
- No debe tener ninguna anotación preespecificada.

Para más información: <https://www.javatpoint.com/pojo-in-java>

1.3 ORM. Hibernar

ORM. Hibernar

Hibernate es una solución de mapeo relacional de objetos (ORM) para JAVA. Es un marco persistente de código abierto creado por Gavin King en 2001. Es un servicio de consulta y persistencia relacional de objetos potente y de alto rendimiento para cualquier aplicación Java.

Hibernate asigna clases de Java a tablas de bases de datos y de tipos de datos de Java a tipos de datos de SQL y libera al desarrollador del 95% de las tareas de programación comunes relacionadas con la persistencia de datos.

Hibernate se ubica entre los objetos Java tradicionales y el servidor de base de datos para manejar todos los trabajos de persistencia de esos objetos en función de los mecanismos y patrones O/R apropiados.

Características de la encapsulación de Hibernate

Ventajas •

Hibernate se encarga de asignar clases Java a tablas de bases de datos

utilizando archivos XML y sin escribir ninguna línea de código. • Proporciona

API simples para almacenar y recuperar Java

objetos directamente hacia y desde la base de datos.

• Si hay un cambio en la base de datos o en cualquier tabla, entonces sólo

necesita cambiar las propiedades del archivo XML. • Elimina los

tipos de SQL desconocidos y proporciona una

forma de evitar objetos Java familiares.

• Hibernate no requiere un servidor de aplicaciones para

funcionar.

• Manipula asociaciones complejas de objetos de tu

base de datos.

• Minimiza el acceso a la base de datos con estrategias de búsqueda inteligentes.

• Proporciona una consulta sencilla de datos.

Para mayor información:

https://www.tutorialspoint.com/hibernate/hibernate_overview.htm

Bases de datos compatibles

Hibernate soporta casi todos los RDBMS principales: •

Oracle

• MySQL •

Base de datos de Microsoft SQL Server

• PostgreSQL •

Motor de base de datos HSQL •

DB2/NT

• Base frontal

• Servidor Sybase SQL •

Servidor dinámico Informix

Tecnologías soportadas

Hibernate soporta una variedad de otras tecnologías, incluyendo: •

XDoclet Spring •

J2EE

• Complementos de

Eclipse • Maven

Línea de tiempo de hibernación

Última versión estable (nueva interfaz HQL):
6.4.0 (nov. 2023)

Última versión compatible (interfaz antigua HQL
con métodos obsoletos): 5.6.14
(noviembre de 2022)

2. ELEMENTOS DE UN PROYECTO DE HIBERNACIÓN

Elementos de un proyecto de Hibernate

Ahora estamos construyendo un proyecto desde cero para crear una aplicación CRUD para trabajar con algunas tablas usando:

- MySQL como nuestro RDBMS •

JDBC como nuestro conector

- Java como nuestro lenguaje •

MAVEN como nuestro framework

- POJO como nuestro patrón DAO •

Hibernación como nuestra técnica ORM

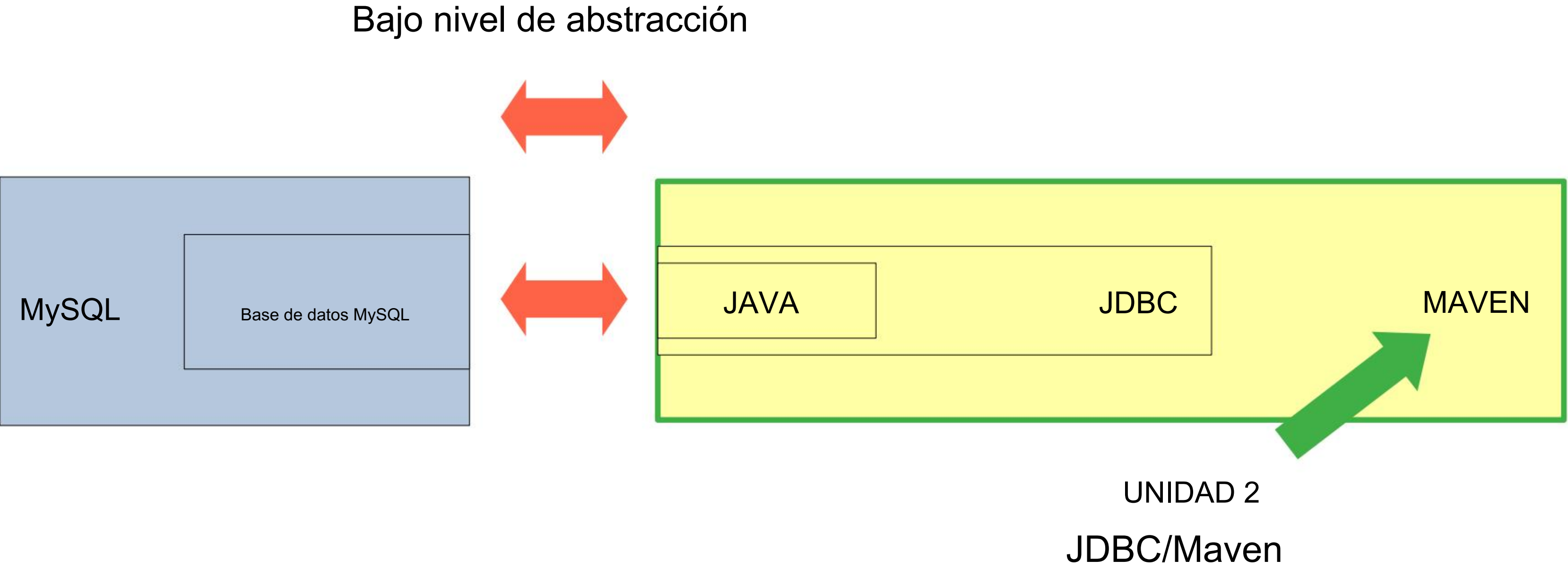


En primer lugar, revisemos lo que estamos tratando de lograr.

Encapsulación. Nivel 1

Lo primero que hicimos (UNIDAD 2) fue acceder a la base de datos mediante **el conector JDBC** y **el marco Maven** sobre **el lenguaje Java**.

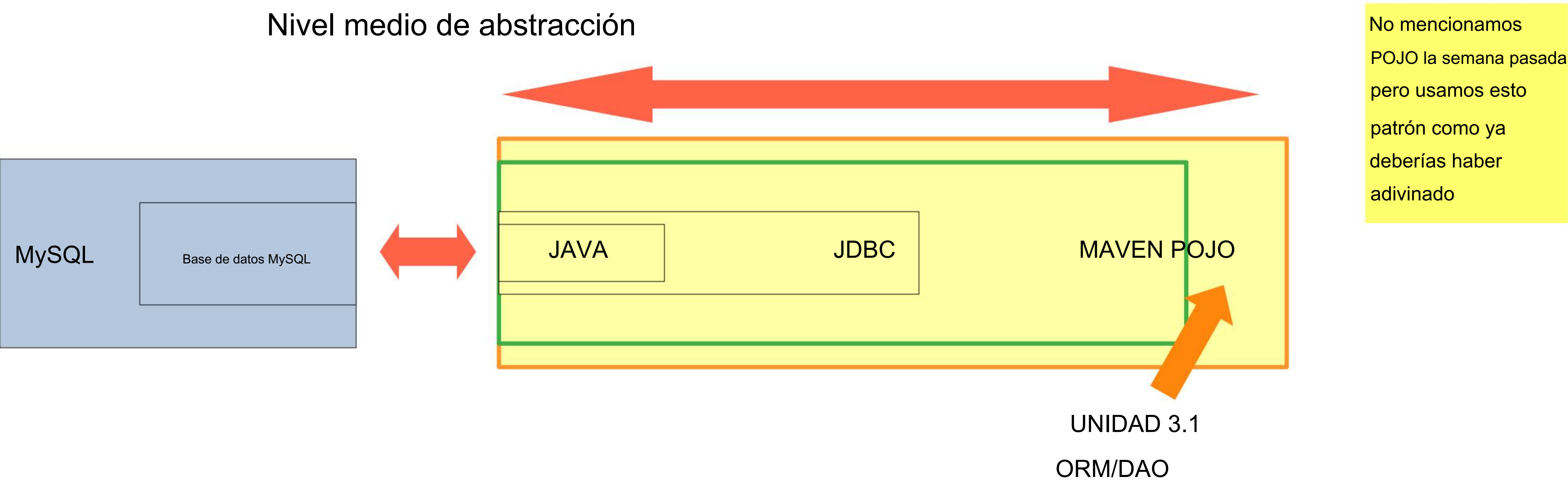
En ese caso, ¡tuvimos que usar código SQL en la clase principal! El nivel de abstracción (la distancia entre la base de datos y el código) era irrelevante.



Encapsulación. Nivel 2

En segundo lugar, (UNIDAD 3 SEMANA 1) nos conectamos a la base de datos mediante el conector JDBC, el marco Maven y objetos DAO usando patrones POJO en lenguaje Java.

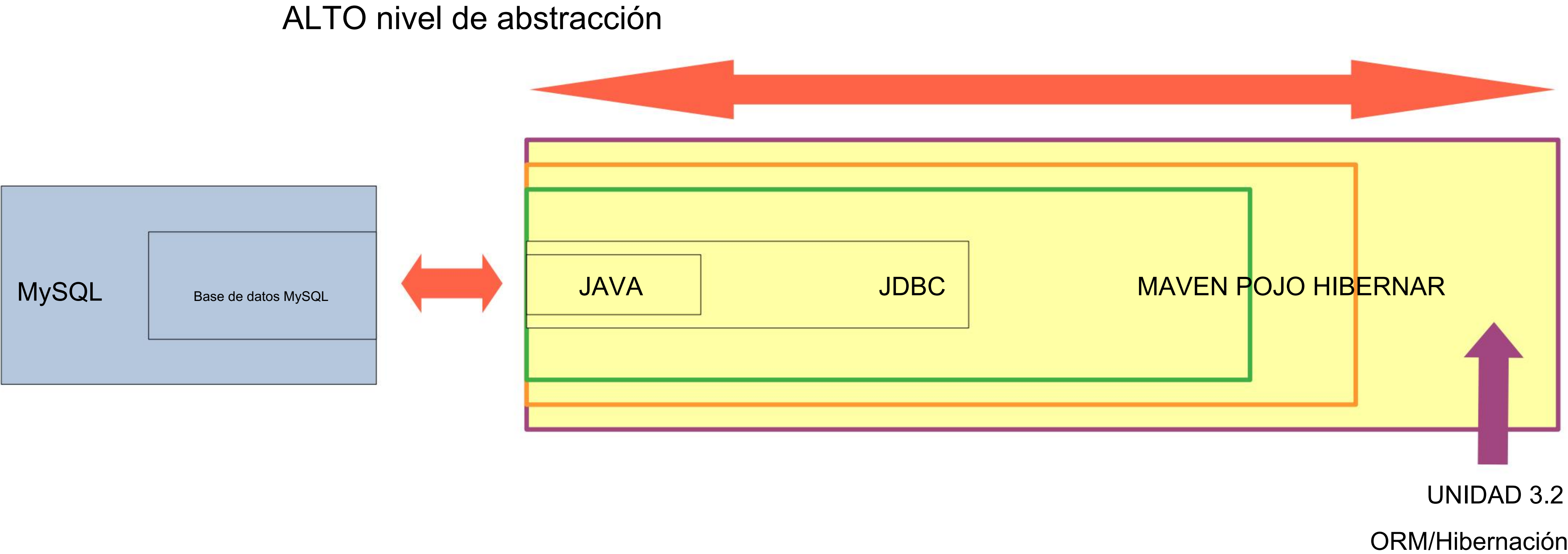
En ese caso, movimos el código SQL a una clase DAO, introduciendo una capa de separación entre la base de datos y el código. El nivel de abstracción fue medio y aún cercano al diseño de la base de datos.



Encapsulación. Nivel 3

Ahora (UNIDAD 3 SEMANA 2 y siguientes) conectaremos la base de datos utilizando **el conector JDBC**, el marco Maven , objetos DAO usando **patrones POJO** y **la técnica** ORM sobre el lenguaje Java.

El código SQL DESAPARECERÁ de nuestro código Java y el nivel de abstracción será máximo.



3. CONFIGURACIÓN DEL PROYECTO Y LA BASE DE DATOS

Recursos de ejemplo

Presentaremos una solución elaborada a partir de estos fantásticos tutoriales:

- <https://www.tutorialspoint.com/hibernate> • https://www.javawebtutor.com/articles/maven/maven_hibernate_example.php • <https://www.journaldev.com/2934/hibernate-many-to-many-mapping-join-tables>

La base de datos (MySQL)

Por ahora, simplemente cree una base de datos simple con estas tablas para representar una relación N:M entre EMPLEADO y CERTIFICADO.

```
CREAR BASE DE DATOS DBCertificados;  
CREAR mavenuser@localhost IDENTIFICADO CON  
mysql_native_password POR 'ada0486';  
  
OTORGAR TODOS LOS PRIVILEGIOS EN DBCertificates.* a  
usuariomaven@localhost;
```

```
USE Certificados DB;  
  
CREAR TABLA Empleado (  
    empID          INTEGER NO NULO AUTO_INCREMENT,  
    nombre VARCHAR(20),  
    apellido VARCHAR(20),  
    salario        DOBLE,  
    RESTRICCIÓN emp_id_pk CLAVE PRIMARIA (id)  
);  
  
CREAR TABLA Certificado (  
    certID INTEGER NO NULL AUTO_INCREMENT,  
    nombre de certificado VARCHAR(30),  
    RESTRICCIÓN cer_id_pk CLAVE PRIMARIA (id)  
);  
  
CREAR TABLA EmpCert (  
    empleadoID INTEGER,  
    certificadoID INTEGER,  
    RESTRICCIÓN empcer_pk CLAVE PRIMARIA (ID de empleado, ID de certificado),  
    RESTRICCIÓN emp_id_fk REFERENCIAS DE CLAVE EXTRANJERA (ID de empleado  
Empleado(empID),  
    RESTRICCIÓN cer_id_fk REFERENCIAS DE CLAVE EXTRANJERA (ID de certificado)  
Certificado (ID de certificado)  
);
```


El proyecto (Java-Maven)

- En ECLIPSE, cree un proyecto Java Maven vacío con estos parámetros como vimos en la UNIDAD 2:

La conexión (base de datos)

Agregue las dependencias al Proyecto Maven (pom.xml) para MySQL. Consulta cómo obtener tu versión aquí: <https://phoenixnap.com/kb/how-to-check-mysql-version>

```
<dependencies>
  <dependencia>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId> <versión>4.11</
    versión> <scope>test</scope> </
    dependencia> <!-- https://
mvnrepository.com/
artifact/org.xerial/sqlite-jdbc --> <dependencia> <groupId>com.mysql</groupId> <artifactId>mysql-
connector-j</
    artifactId> <versión>8.0.33</versión> </
    dependencia> </dependencies>
```

4. CONFIGURAR LA HIBERNACIÓN

Descargar Hibernate a través de Maven

Siga estos sencillos pasos para descargar Hibernate: •

Vaya a <https://hibernate.org/orm/releases/> •

Haga clic en “Más información” en la última versión estable •

Opción 1: • Haga clic en el enlace hibernate-core (xxxFinal) •

Opción 2: • Haga clic en el botón “Artefactos Maven”

• Haga clic en el enlace hibernate-core (xxxFinal) •

Copie y pegue la dependencia en su POM • Guarde y deje que Maven haga el trabajo.

```
<dependencies> [...]  
  
  <!--  
  https://central.sonatype.com/artifact/org.hibernate.orm/hibernate-core/  
  6.4.0.Final--> <dependency>  
  
    <groupId>org.hibernate.orm</groupId> <artifactId>hibernate-  
    core</artifactId> <version>6.4.0.Final</version> </  
    dependency> </dependencies>
```




Configurar la hibernación manualmente

Ahora necesitamos crear:

- Un archivo de configuración



genérico • Un archivo de configuración + un archivo de clase POJO para cada tabla

 Por ahora, estamos trabajando sólo con esta tabla.

Archivo de configuración genérico

Siga estos sencillos pasos para configurar este primer archivo:

- Cree una carpeta **de recursos** en src/main

- Crear un archivo en src/main/resources

- Guárdelo como hibernate.cfg.xml •

Copie y pegue el código de la siguiente diapositiva, cambiando la configuración de

conexión a la base de datos: •

Nombre de la base de datos: DBCertificates

- Usuario: <su usuario> o mavenuser •

Contraseña: <su contraseña> o ada0486

hibernación.cfg.xml

Archivo de configuración genérico

Por ahora estamos trabajando sólo con esta tabla, pero si queremos trabajar con más tablas, agregaremos más líneas y más archivos: un archivo para cada tabla



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

<hibernate-configuration> <session-
  factory> <property
    name="hibernate.connection.url">jdbc:mysql://localhost:3306/DBCertificates</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">actualizar</property>
    <mapping resources="employee.hbm.xml" /> </sesión- fábrica> </
hibernación-configuración>
```



hibernación.cfg.xml

Archivo de configuración de tabla(s)

Siga estos sencillos pasos para configurar este archivo:

- Crear un archivo en src/main/resources
- Guárdelo como empleado.hbm.xml.
- Copie y pegue el código de la siguiente diapositiva, donde configurará cómo se representa su tabla en su clase usando DAO (mediante notación POJO).



Por ahora, estamos trabajando sólo con esta tabla.

empleado.hbm.xml

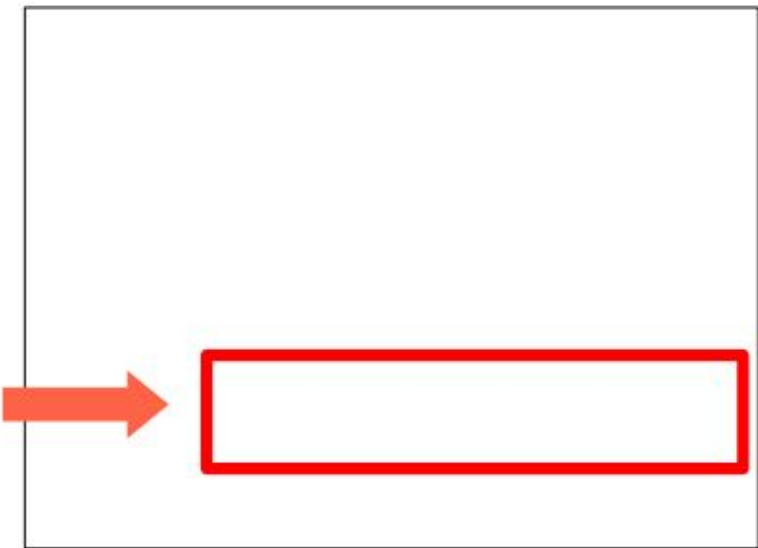
Archivo de configuración de tabla(s)

Como puede ver aquí, estamos configurando este archivo de mapeo para que funcione solo con la tabla "Empleado" . El siguiente paso será crear una clase Java (DOMAIN.Employee.java) para mapear esta tabla de acuerdo con estas especificaciones.

```
<?xml versión = "1.0" codificación = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//
Hibernate/Hibernate Mapping DTD//ES"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<mapeo-hibernación>
  < nombre de clase = "DOMINIO.Empleado" tabla = "Empleado"> < atributo
    meta = "descripción-clase">
      Esta clase contiene los detalles del empleado. </meta> <id
        nombre =
          "iEmpID" tipo = "int" columna = "empID">
            <generador clase="nativo"/>
          </id> <
            nombre de propiedad = "stFirstName" columna = "firstname" tipo = "cadena"/>
            < nombre de propiedad = "apellido" columna = "apellido" tipo = "cadena"/>
            < nombre de propiedad = "dSalario" columna = "salario" tipo = "doble"/>
          </clase>
        </hibernate-mapping>
```

empleado.hbm.xml

El elemento <generator> dentro del elemento id se utiliza para generar los valores de clave principal automáticamente. El atributo de clase del elemento generador se establece en nativo para permitir que la hibernación seleccione un algoritmo de identidad, secuencia o subprocesso para crear una clave principal dependiendo de las capacidades de la base de datos subyacente.



Archivo POJO (para cada tabla)

Ahora deberíamos crear un java clase (DOMAIN.Employee.java) para asignar esta tabla de acuerdo con las especificaciones de la base de datos y los estándares POJO.

Tenga cuidado con el nombre de las variables. El

Los captadores y definidores deben seguir los mismos criterios para permitir que Hibernate encuentre los métodos apropiados: <https://stackoverflow.com/questions/921239/hibernate-propertynotfoundexception-could-not-find-a-getter-for>



```
paquete DOMINIO;

importar java.util.Set;

Empleado de clase pública {

    // ATRIBUTOS
    privado int iEmpID; cadena
    privada stFirstName; cadena privada
    apellido; doble salario privado ;

    // MÉTODOS
    //Constructor vacío
    empleado público () { }

    // Constructor sin ID. Todos los campos, excepto la clave principal.
    Empleado público (String stFirstName, String stLastName, doble dSalario) {
        this.stFirstName = stFirstName; this.stLastName
        = stLastName; this.dSalario = dSalario;

    }

    // GETTERS
    public int getiEmpID() { return iEmpID;

    }

    public String getstFirstName() { return stFirstName;

    }

    public String getstLastName() { return stLastName;

    }

    public double getdSalary() { return dSalario;

    }

    // CONFIGURADORES
    public void setiEmpID(int iemplID)
    { this.iEmpID = iemplID;
    }

    public void setstFirstName(String stFirstName) { this.stFirstName = stFirstName;

    }

    public void setstLastName(String stLastName) { this.stLastName =
    stLastName;

    }

    public void setdSalary(doble dSalario) { this.dSalario = dSalario;

    }

}
```

5. HIBERNAR: SESIONES

Ejecute Hibernate a través de sesiones

Ahora que hemos configurado la Hibernación, vamos a activarla y dar el paso final. Trabajaremos con un objeto llamado FÁBRICA y varias SESIONES como esta:

```
// CONFIGURACIÓN
Configuración = cfg = nueva Configuración().configure();

// FÁBRICA
SessionFactory sessionFactory = cfg.buildSessionFactory();

// SESIONES
Sesión sesión = sessionFactory.openSession(); sesión.beginTransaction();

    Empleado emp = nuevo Empleado(); emp.setEmpId(1);
    emp.setEmpName("Sergio");

    ...

sesión.guardar(emp);
sesión.getTransaction().commit(); sesión.cerrar();
```

Vamos a dividir esta idea en tres archivos más:

- TestHibernateMySQL.java (con método principal)
- HibernateUtil.java (para activar Hibernate)
- EmployeeDAO.java (con métodos CRUD)

Faltan tres archivos más

- 1) Archivo DAO con todos los métodos para interactuar con la base de datos vía Hibernate.
- 2) Un nuevo archivo de clase de “prueba” con el método principal para ejecutar nuestro proyecto.
- 3) Un archivo de clase para activar el proceso de Hibernación y ponerlo en suspensión.

Estos son los archivos que necesitamos:

- Archivos de configuración de Maven: pom.xml •

Archivos de configuración de Hibernate: hibernate.cfg.xml • Un archivo de

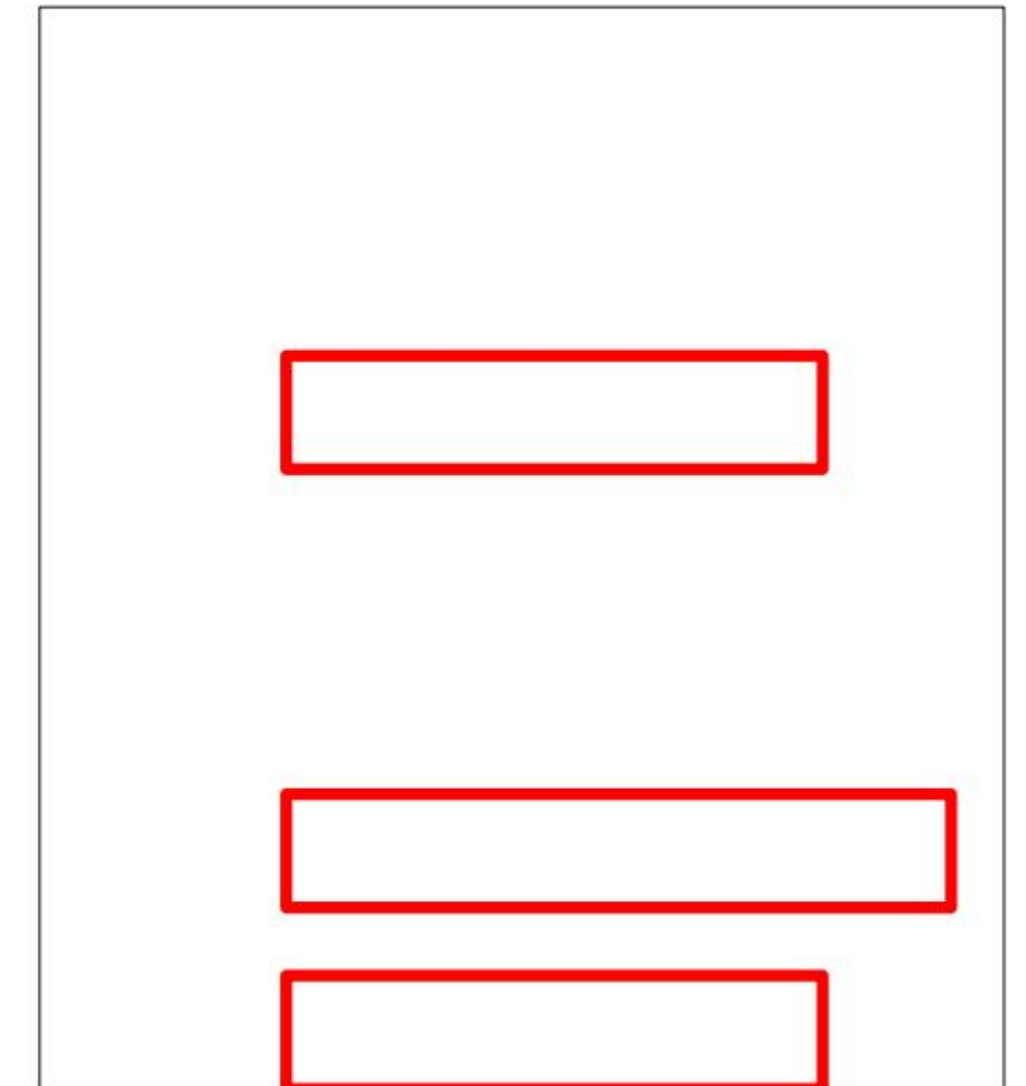
clase para activar/desactivar Hibernate: HibernateUtil.java • Un archivo de clase de

interfaz “ficticia” con el método principal: TestHibernateMySQL.java • Para cada tabla/recurso: • Archivo de capa de datos:

EmployeeDAO.java • Archivo

relacionado con el dominio: Employee.java • Archivo de

tabla de configuración: empleado.hbm.xml



6. HIBERNAR: MAPEO DE UNA ÚNICA TABLA

6.1 archivo DAO

Archivo DAO. Operaciones CRUD. Crear

```
DATOS del paquete ;

importar java.util.List; importar
java.util.Set; importar
java.util.Iterator;

importar org.hibernate.HibernateException; importar
org.hibernate.Session; importar
org.hibernate.Transaction;

importar DOMINIO.*;
importar UTIL.*;

clase pública EmpleadoDAO {
    /*
     * -----
     * INSERTAR
     * -----
     */
    /* Método para CREAR un empleado en la base de datos */ public Employee
addEmployee(String stFirstName, String stLastName, double dSalary) {

    Sesión hibSession = HibernateUtil.SFACTORY.openSession(); //abre la fábrica de sesiones de hibernación Transacción txDB = null; //transacción
de base de datos Empleado objEmpleado = new Empleado(stFirstName,
stLastName, dSalary);

    intente
    { txDB = hibSession.beginTransaction(); //inicia la transacción //el método de guardar está
en desuso, pero aún funciona en la última versión //https://stackoverflow.com/questions/71211904/alternative-
to-using-deprecated-save-method-in-hibernate hibSession.persist (objEmpleado); txDB.commit(); //finaliza la transacción System.out.println("***** Elemento agregado.
\n"); } captura (HibernateException hibe) {

    si (txDB != nulo)
        txDB.rollback(); //algo salió mal, así que revertir hibe.printStackTrace(); } finalmente
    { hibSession.close(); //cerrar sesión
de hibernación

    } return objEmpleado;
}
```



Archivo DAO. Operaciones CRUD. Leer

```
/*
 * _____
 * SELECCIONAR
 * _____
 */
/* Método para LEER todos los empleados */ public void
listEmployees() {

    Sesión hibSession = HibernateUtil.SFACTORY.openSession(); //abre la fábrica de sesiones de hibernación Transacción txDB = null; //transacción
    de base de datos

    intente
    { txDB = hibSession.beginTransaction(); //inicia la transacción //el antiguo método
      createQuery está obsoleto, pero sigue funcionando en la última versión //https://www.roseindia.net/hibernate/hibernate5/
      hibernate-5-query-deprecated.shtml List<Empleado> listEmployees = hibSession.createQuery("DESDE Empleado",
      Empleado.class).list(); si (listaEmpleados.isEmpty())

        System.out.println("***** No se encontraron elementos"); else

        System.out.println("\n***** Iniciar listado...\n");

        for (Iterador<Empleado> itEmployee = listEmployees.iterator(); itEmployee.hasNext();) {
            Empleado objEmpleado = (Empleado) itEmployee.next(); System.out.print("
            Nombre: System.out.print("Apellido : " + objEmployee.getstFirstName() + " | "; " | ");
            System.out.println("Salario: " + objEmpleado.getstApellido() +
            + objEmpleado.getdSalario());

        } txDB.commit(); //finaliza la transacción
    } captura (HibernateException hibe) {
        si (txDB != nulo)
            txDB.rollback(); //algo salió mal, así que revertir hibe.printStackTrace(); } finalmente
        { hibSession.close(); //cerrar sesión
        de hibernación

    }
}
```



Archivo DAO. Operaciones CRUD. **Actualizar**

```
/*
 * _____
 * ACTUALIZAR
 * _____
 */

/* Método para ACTUALIZAR el salario de un empleado */ public void
updateEmployee(int iEmpID, double dSalary) {

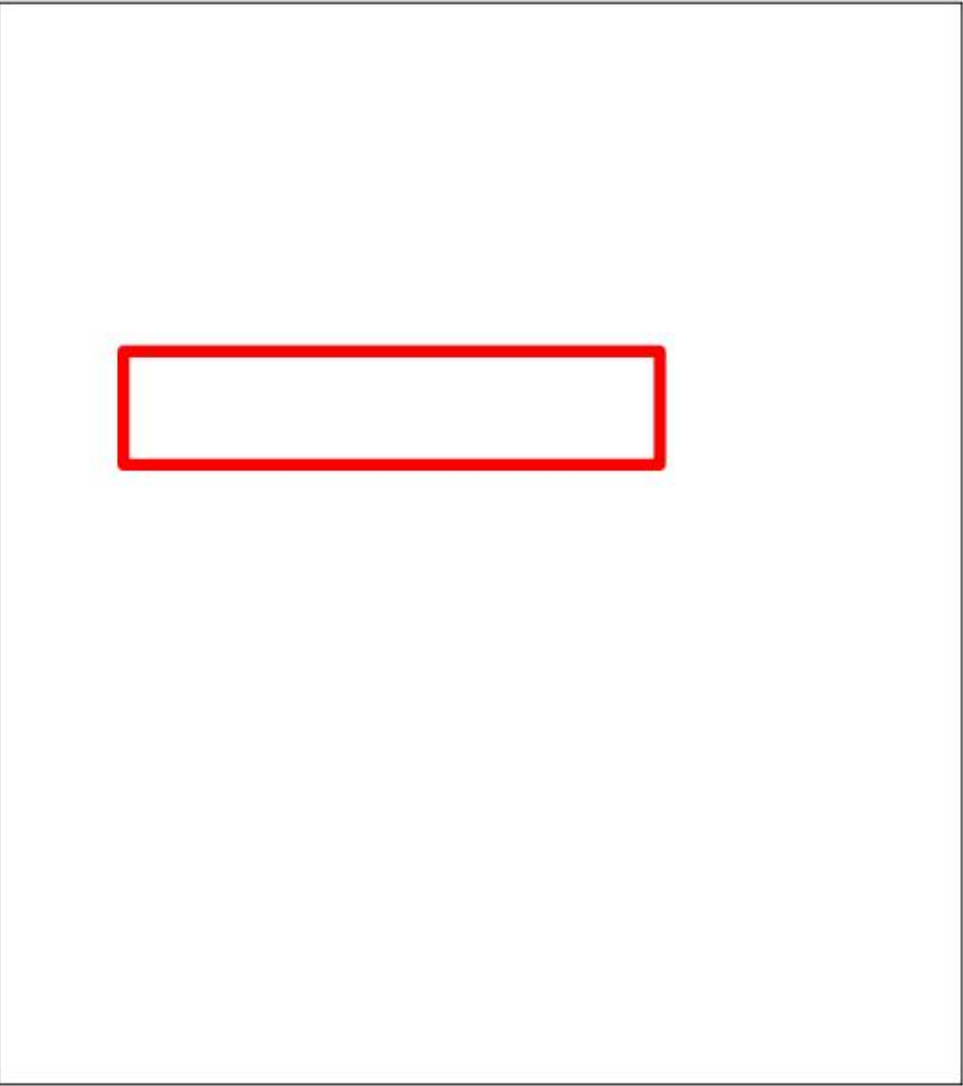
    Sesión hibSession = HibernateUtil.SFACTORY.openSession(); //abre la fábrica de sesiones de hibernación Transacción txDB = null; //transacción
    de base de datos

    intente
    { txDB = hibSession.beginTransaction(); // inicia la transacción Empleado objEmployee =
      (Empleado) hibSession.get(Employee.class, iEmpID); objEmpleado.setdSalario(dSalario); //el método de actualización
      está obsoleto, pero sigue funcionando en la última
      versión //https://stackoverflow.com/questions/71211904/alternative-to-using-deprecated-save-
método-en-hibernación

      hibSession.merge(objEmpleado); txDB.commit(); //
      finaliza la transacción System.out.println("*****
      Elemento actualizado.\n"); } captura (HibernateException hibe) {

      si (txDB != nulo)
        txDB.rollback(); //algo salió mal, así que revertir
      hibe.printStackTrace(); } finalmente

    { hibSession.close(); //cerrar sesión de hibernación
    }
}
```



Archivo DAO. Operaciones CRUD. **Borrar**

```
/*
 * _____
 * BORRAR
 * _____
 */

/* Método para ELIMINAR un empleado de los registros */ public void
deleteEmployee(int iEmpID) {

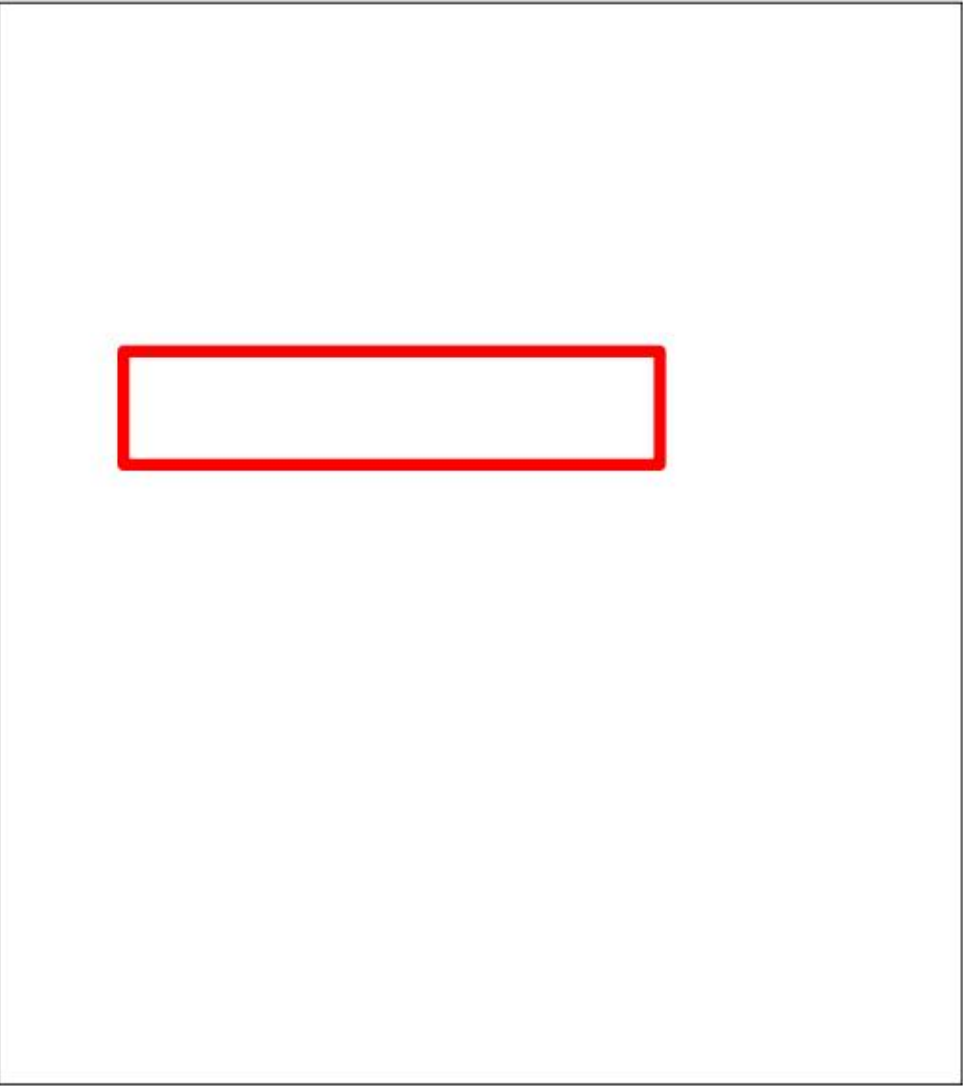
    Sesión hibSession = HibernateUtil.SFACTORY.openSession(); //abre la fábrica de sesiones de hibernación Transacción txDB = null; //transacción
    de base de datos

    intente
    { txDB = hibSession.beginTransaction(); // inicia la transacción Empleado objEmployee =
      (Empleado) hibSession.get(Employee.class, iEmpID); //el método de eliminación está en desuso, pero aún funciona
      en la última versión //https://stackoverflow.com/questions/71211904/alternative-to-using-deprecated-save-
método-en-hibernación

      hibSession.remove(objEmpleado); txDB.commit(); //
      finaliza la transacción System.out.println("*****
      Elemento eliminado.\n"); } captura (HibernateException hibe) {

      si (txDB != nulo)
        txDB.rollback(); //algo salió mal, así que revertir
        hibe.printStackTrace(); } finalmente

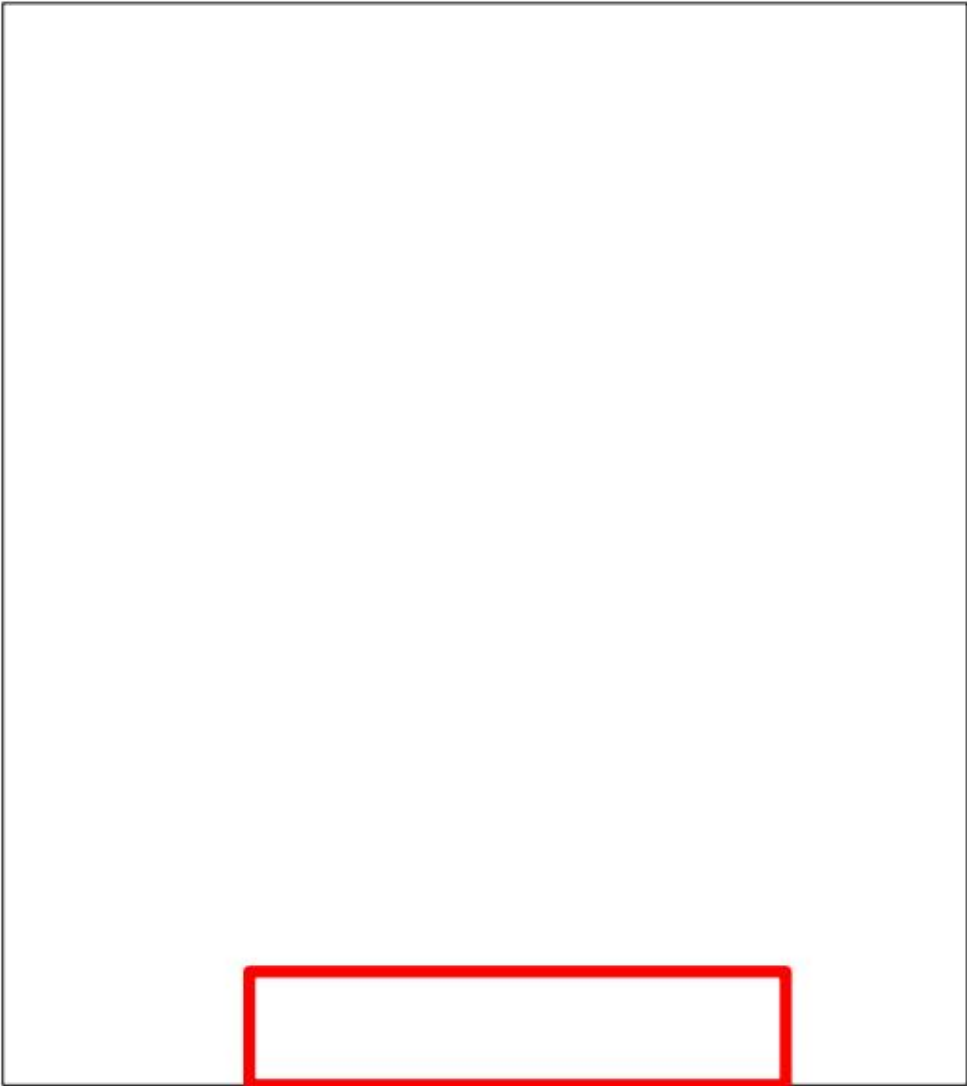
    { hibSession.close(); //cerrar sesión de hibernación
    }
}
```



6.2 Hibernación de utilidades

Utilidad de hibernación

Finalmente, necesitamos una clase para activar el proceso de Hibernación y ponerlo en suspensión.



```
paquete UTIL;

importar java.util.logging.Level;

importar org.hibernate.SessionFactory; importar
org.hibernate.cfg.Configuration;

clase pública HibernateUtil {

    // Sesión persistente public static
    final SessionFactory SFACTORY = buildSessionFactory();

    // GESTIÓN DE SESIONES /*

    * Crear nueva sesión de hibernación */

SessionFactory estática privada buildSessionFactory()
{ java.util.logging.Logger.getLogger("org.hibernate").setLevel(Level.OFF); try { // Crea SessionFactory desde

    hibernate.cfg.xml return new Configuration().configure().buildSessionFactory(); } catch
    (Throwable sfe) { // Asegúrese de registrar la excepción, ya que podría ser tragada System.err.println("
    Falló la creación de SessionFactory." +
        sfe); lanzar un nuevo ExceptionInInitializerError(sfe);

    }
}

/*
    * Cerrar sesión de hibernación */

public static void ShutdownSessionFactory() { // Cerrar cachés y grupos
    de conexiones getSessionFactory().close();

}

/*
    *Get método para obtener la sesión*/

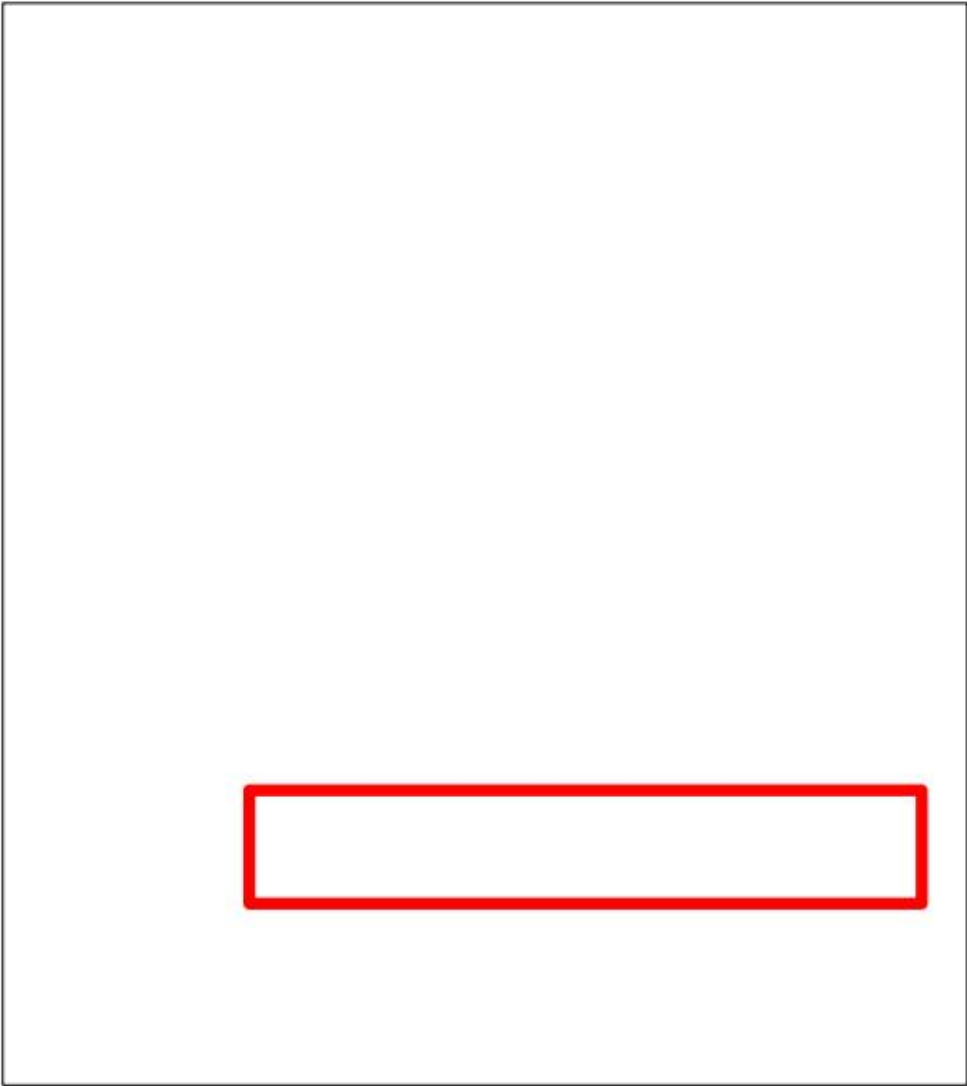
public static SessionFactory getSessionFactory() {
    devolver FÁBRICA;
}

}
```

6.3 Prueba de hibernación

PruebaHibernar. Capa de interfaz

Y el programa principal:



```
datos de
importacion .*; importar
DOMINIO.*; importar UTIL.*;

clase pública TestHibernateMySQL {

    /*
     * -----
     * PROGRAMA PRINCIPAL
     * -----
     */

    principal vacío estático público (cadena [] stArgs) {

        //Crear nuevos objetos DAO para operaciones CRUD
        EmpleadoDAO objEmpleadoDAO = nuevo EmpleadoDAO();

        //TRUNCAR TABLAS. Eliminar todos los registros de las tablas
        objEmployeeDAO.deleteAllItems();

        /* Agregar registros en la base de datos */ Empleado
        objEmp1 = objEmployeeDAO.addEmployee("Alfred", "Vincent", 4000); Empleado objEmp2 =
        objEmployeeDAO.addEmployee("John", "Gordon", 3000);

        /* Actualizar el campo de salario del empleado */
        objEmployeeDAO.updateEmployee(objEmp1.getiEmpID(), 5000); /* Enumera todos los
        empleados */ objEmployeeDAO.listEmployees(); /*
        Eliminar un empleado de la base de datos */
        objEmployeeDAO.deleteEmployee(objEmp2.getiEmpID()); /*
        Enumera todos los empleados */ objEmployeeDAO.listEmployees();

        //Cerrar la fábrica de sesiones de hibernación global
        HibernateUtil.shutdownSessionFactory();

    }

}
```

7. HIBERNAR: MAPEO AVANZADO

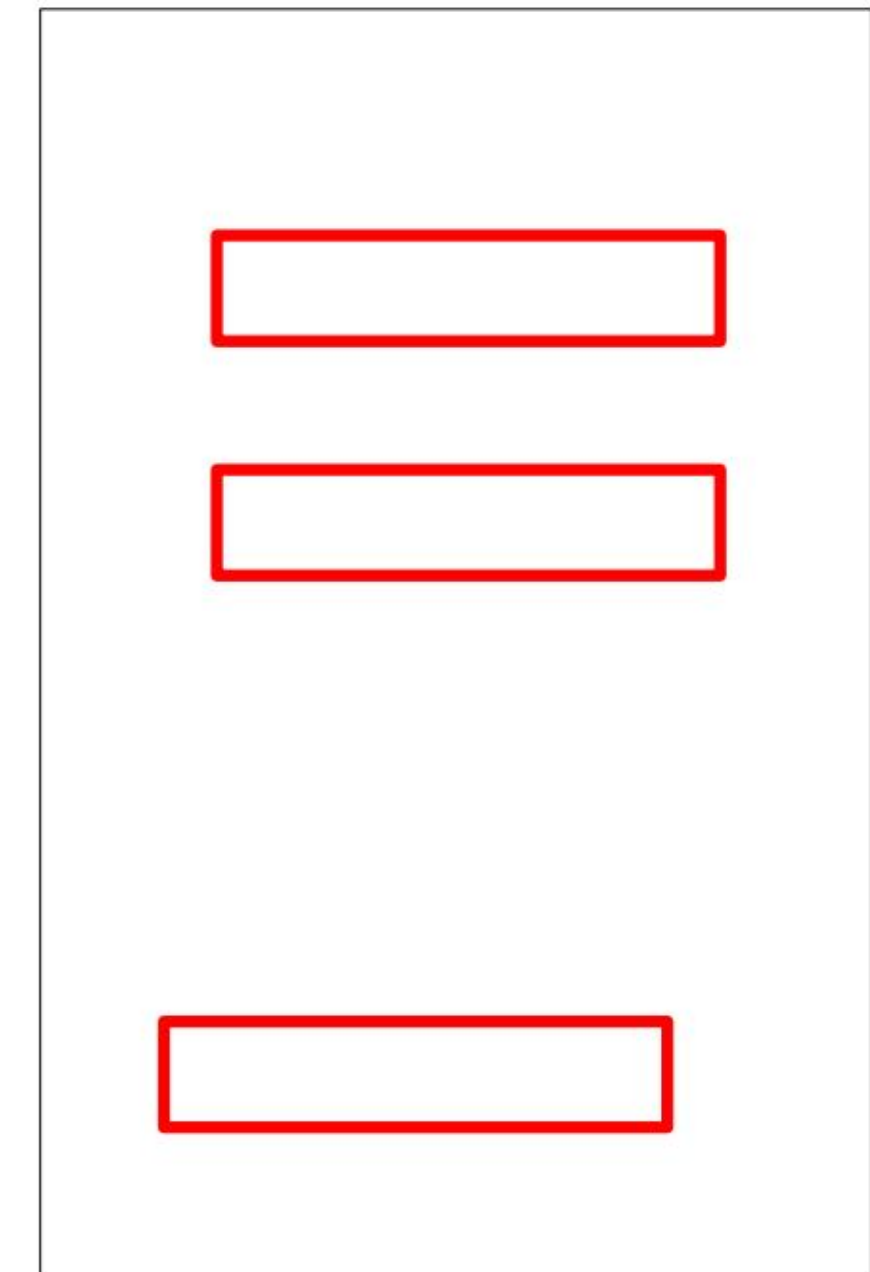
Agregar más recursos (tablas)

Una vez que hayamos despertado el proceso de Hibernación para trabajar con una sola tabla podemos ir más allá y agregar más recursos siguiendo estos sencillos pasos:

- 1) Cree su archivo DAO (DATA/TableDAO.java)
- 2) Cree su archivo POJO (DOMINIO/Table.java)
- 3) Modifique el archivo de configuración (mapeo) de la tabla relacionada con (table.hbm.xml)
- 4) Agregue el recurso al archivo genérico de configuración de Hibernate (hibernate.cfg.xml)

Tenga en cuenta que colocar archivos en paquetes dentro de src/main/java es **TOTALMENTE OPCIONAL** pero nos ayuda a facilitar su comprensión.

Eche un vistazo a los nuevos archivos agregados para manejar estas dos entidades (Empleado y Certificado). Tenga en cuenta que Hibernate manejará las relaciones de uno a muchos, de muchos a muchos, etc., por lo que solo tendrá que configurar las ENTIDADES, nunca las tablas intermedias como EmpCert.



Configuración de muchos a muchos (N:M)

Para configurar este tipo de relación necesitamos realizar esta solución:

- 1) Cree su archivo DAO (DATA/CertificateDAO.java)
- 2) Cree su archivo POJO (DOMINIO/Certificado.java)
- 3) Cree métodos en el otro lado de la relación igual a () y hashCode () para que Java pueda determinar si dos elementos/objetos son idénticos.

Por ejemplo, podemos hacerlo en el lado derecho: DOMINIO/Certificado.java

- 4) Cree su archivo de configuración (mapeo) (certificate.hbm.xml)
- 5) Agregue el recurso al archivo genérico de configuración de Hibernate (hibernate.cfg.xml)
- 6) Cree un nuevo campo en un lado de la relación (lado izquierdo) y coloque un campo SET allí para almacenar "tantos elementos del otro lado" que necesitemos, con sus definidores y captadores.

Por ejemplo, podemos hacerlo en el lado izquierdo: DOMINIO/Empleado.java

- 7) Modifique el archivo DAO para insertar un conjunto de elementos de un lado al agregar elementos del otro y enumerar todos los elementos relacionados.

Por ejemplo, podemos hacerlo en el lado izquierdo: DATA/EmployeeDAO.java

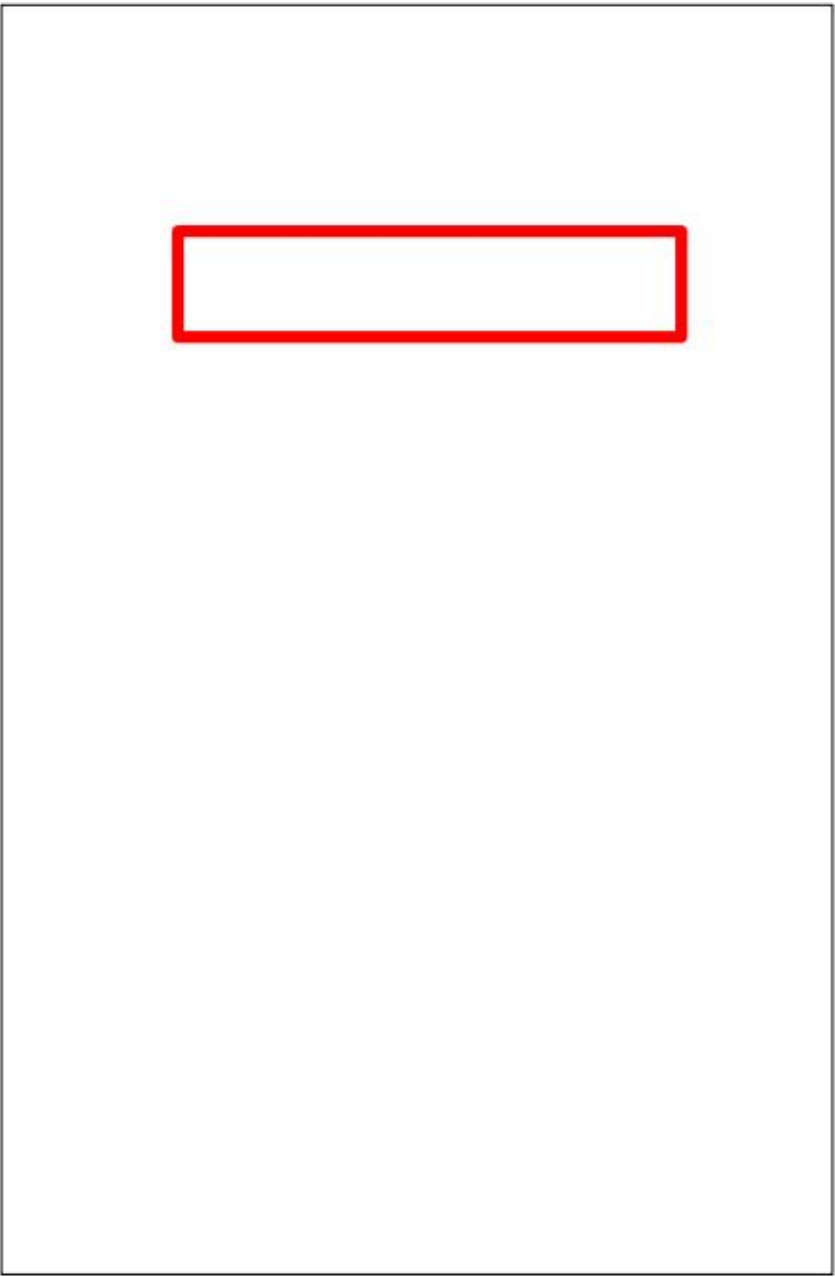
- 8) Agregue una nueva asignación de campo SET en el lado izquierdo (src/main/resources/Employee.hbm.xml)

Configuración de muchos a muchos. Paso 1

1) Cree su archivo DAO (DATA/CertificateDAO.java)

Todos los métodos para son casi idéntico

datos.EmployeeDAO.java



```
importar java.util.List; importar
java.util.lterator;

importar org.hibernate.HibernateException; importar
org.hibernate.Session; importar
org.hibernate.Transaction;

importar DOMINIO.*;
importar UTIL.*;

Certificado de clase públicaDAO {

    // INSERTAR
    /* Método para CREAR un certificado en la base de datos */ public Certificate
addCertificate(String stCertName) { }

    // SELECCIONAR
    /* Método para LEER todos los certificados */ public void
listCertificates() { }

    // ACTUALIZAR
    /* Método para ACTUALIZAR el nombre de un certificado */ public
void updateCertificate(int iCertID, String stCertName) { }

    // BORRAR
    /* Método para ELIMINAR un certificado de los registros */ public void
deleteCertificate(int iCertID) { }

    /* Método para ELIMINAR todos los registros */
    public void deleteAllItems() { }

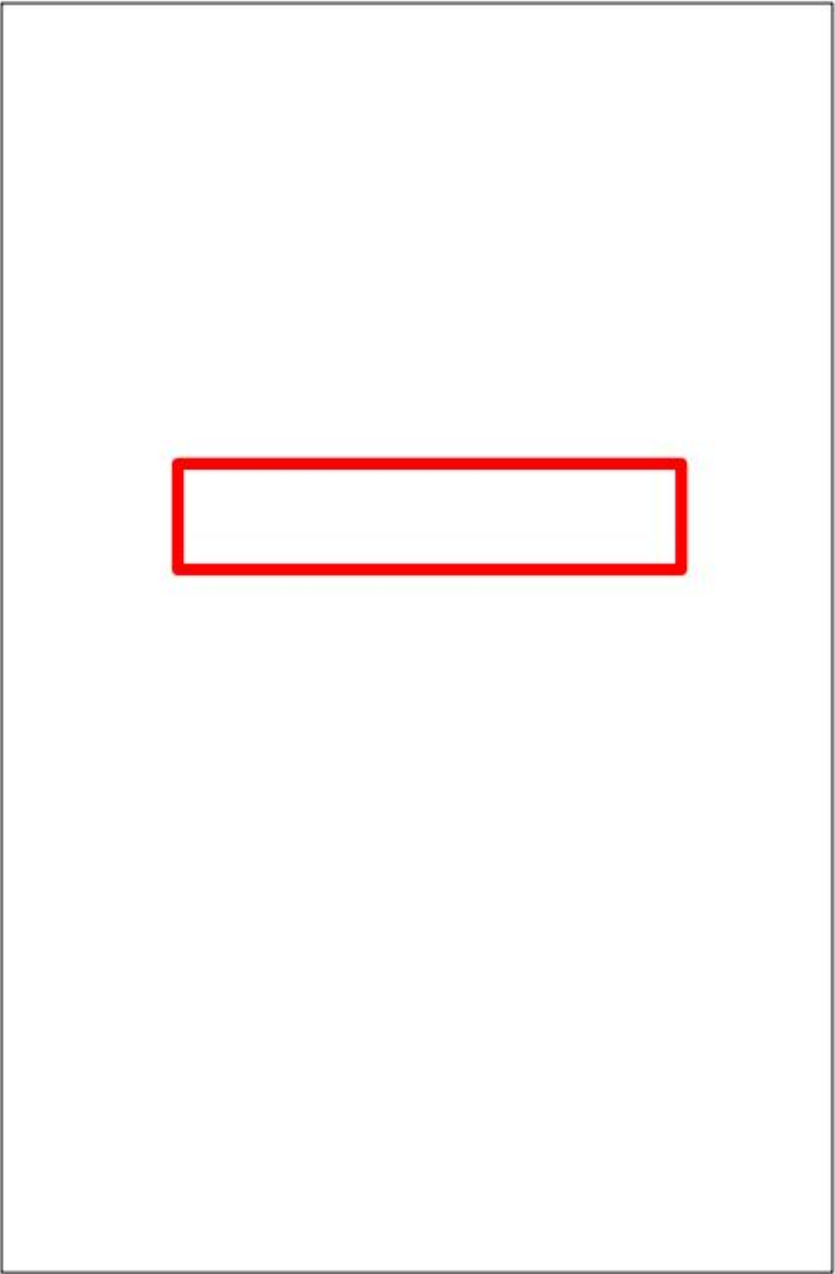
}
```

Configuración de muchos a muchos. Paso 2 y 3

2) Cree su archivo POJO (DOMINIO/Certificado.java)

3) Cree métodos en el otro lado de la relación igual a() y hashCode() para que Java pueda determinar si dos elementos/objetos son idénticos.

Por ejemplo, podemos hacerlo en el lado derecho: DOMINIO/Certificado.java



```
paquete DOMINIO;

Certificado de clase pública {

    // ATRIBUTOS privado
    int iCertID; cadena privada
    stCertName;

    // MÉTODOS

    //Constructor vacío certificado
    público () {}

    //Constructor sin ID. Todos los campos, excepto la clave principal público Certificate(String
    stCertName) { this.stCertName = stCertName;

    }

    // GETTERS
    público int getiCertID() { return iCertID;

    }

    cadena pública getstCertName() {
        devolver stCertName;
    }

    //
    CONFIGURADORES público void setiCertID(int iCertID)
        { this.iCertID = iCertID;

    }

    público void setstCertName(String stCertName) { this.stCertName = stCertName;

    }

}/*
 * Establecer la relación MUCHOS A MUCHOS entre Empleado y Certificado
 * Elegimos el lado derecho (mesa Certificado)
 * Cree métodos iguales () y hashCode () para que Java pueda determinar si hay dos
 elementos/objetos son idénticos */

    público booleano es igual (Objeto obj) { si (obj == nulo)
        devuelve falso; si (!
            this.getClass().equals(obj.getClass())) devuelve falso;

            Certificado objCert = (Certificado) obj; si ((this.iCertID ==
            objCert.getiCertID()) &&
            (this.iCertName.equals(objCert.getstCertName())) { devolver verdadero;

        } falso retorno;
    }

    público int hashCode() { int iHash = 0;
        iHash = (iCertID +
            stCertName).hashCode(); devolver iHash;

    }

}
```

Configuración de muchos a muchos. Etapa 4

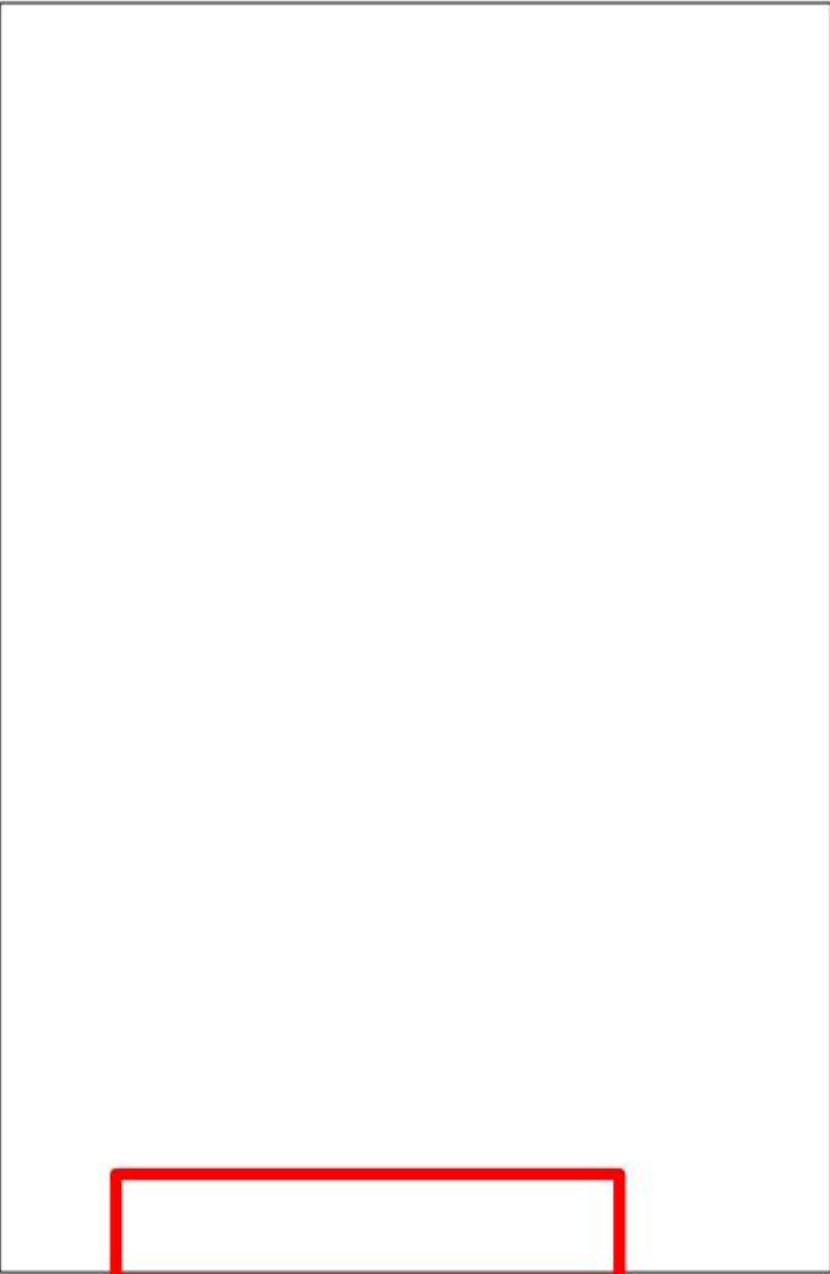
4) Cree su archivo de configuración (mapeo) (certificate.hbm.xml)



```
<?xml versión = "1.0" codificación = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/
Hibernate Mapping DTD//ES"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<mapeo-hibernación>
  < nombre de clase = "DOMINIO.Certificado" tabla = "Certificado">
    <meta atributo = "descripción de clase">
      Esta clase contiene el detalle del certificado. </meta> <id nombre
      = "iCertID"
      tipo = "int" columna = "certID">
        <generador clase="nativo"/>
      </id> <
      nombre de propiedad = "stCertName" columna = "certname" tipo = "cadena"/>
    </clase>
  </hibernate-mapping>
```

Configuración de muchos a muchos. Paso 5

5) Agregue el recurso al archivo genérico de configuración de Hibernate (hibernate.cfg.xml)



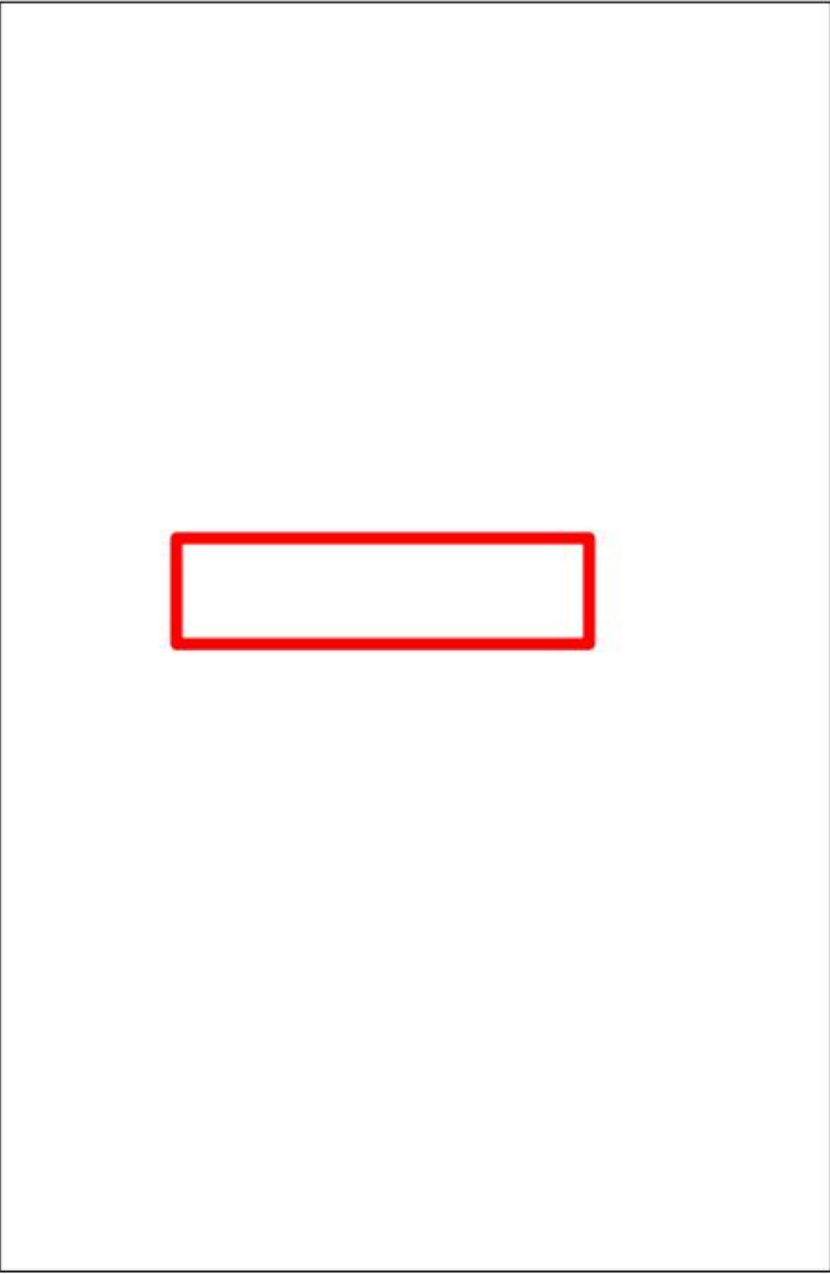
```
<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE
hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//ES" "http://www.hibernate.org/dtd /hibernate-
configuration-3.0.dtd">

<hibernate-configuration> <session-
factory> <property
    name="hibernate.connection.url">jdbc:mysql://localhost:3306/DBCertificates</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"> falso</
property> <property name="format_sql">true</property>
<property name="hbm2ddl.auto">actualizar</property>
<mapping recurso="employee.hbm.xml" /> <mapping recurso=
"certificado.hbm.xml" /> </session-factory> </hibernate-
configuration>
```


Configuración de muchos a muchos. Paso 6

6) Cree un nuevo campo en un lado de la relación (lado izquierdo) y coloque un campo SET allí para almacenar "tantos elementos del otro lado" que necesitemos, con sus definidores y captadores.

Por ejemplo, podemos hacerlo en el lado izquierdo: DOMINIO/Empleado.java



```
paquete DOMINIO;

importar java.util.Set;

Empleado de clase pública {

    // ATRIBUTOS
    privado int iEmplID; cadena
    privada stFirstName; cadena privada
    apellido; doble salario privado ; //Establecer
    clase en Java https://
    www.geeksforgeeks.org/set-in-java/ private Set<Certificate> relCertificates; //relación Empleado-
    Certificado (N:M)

    // MÉTODOS

    //Constructor vacío
    empleado público () { }

    //Constructor sin ID. Todos los campos, excepto la clave principal.
    Empleado público (String stFirstName, String stLastName, doble dSalary) { this.stFirstName = stFirstName;
        this.stLastName = stLastName; this.dSalario =
        dSalario;

    }

    [...]

    // RECOGEDORES

    [...]

    conjunto público <Certificado> getrelCertificados() {
        devolver certificados rel;
    }

    // CONFIGURADORES

    [...]

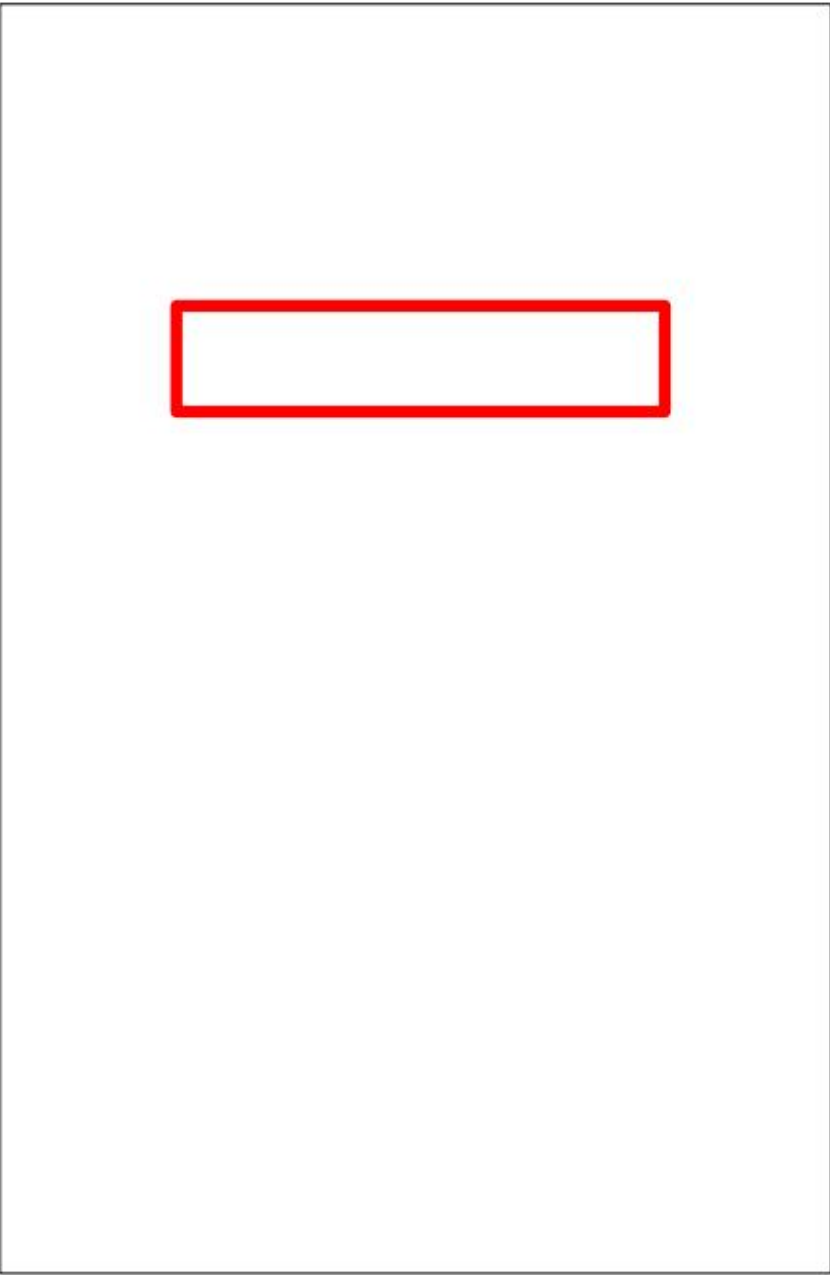
    public void setrelCertificates(Set<Certificado> relCertificates) { this.relCertificates = relCertificates;
    }

}
```

Configuración de muchos a muchos. Paso 7

7) Modifique el archivo DAO para insertar un conjunto de elementos de un lado al agregar elementos del otro y enumerar todos los elementos relacionados.

Por ejemplo, podemos hacerlo en el lado izquierdo: DATA/EmployeeDAO.java



```
clase pública EmpleadoDAO {

    // INSERTAR
    /* Método para CREAR un empleado en la base de datos */ public Employee
    addEmployee(String stFirstName, String stLastName, double dSalary, Set<Certificate> setCertificates) {

        Sesión hibSession = HibernateUtil.SFACTORY.openSession(); //abre la fábrica de sesiones de hibernación Transacción txDB = null; //transacción
        de base de datos Empleado objEmployee = new Empleado(stFirstName,
        stLastName, dSalary);

        intente
        { txDB = hibSession.beginTransaction(); //inicia la transacción
        objEmployee.setrelCertificates(setCertificates); hibSession.persist(objEmpleado);
        txDB.commit(); //finaliza la transacción
        System.out.println("***** Elemento agregado.\n"); }
        catch (HibernateException hibe) { if (txDB != null) txDB.rollback(); //
        algo salió mal, así que revertir hibe.printStackTrace(); }
        finalmente
        { hibSession.close(); //cerrar sesión de hibernación

        } return objEmpleado;
    }

    // SELECCIONAR
    /* Método para LEER todos los empleados */ public void
    listEmployees() {

        Sesión hibSession = HibernateUtil.SFACTORY.openSession(); //abre la fábrica de sesiones de hibernación Transacción txDB = null; //transacción
        de base de datos

        intente
        { txDB = hibSession.beginTransaction(); //inicia la transacción Lista<Empleado> listEmployees
        = hibSession.createQuery("FROM Employee", Employee.class).list(); si (listaEmpleados.isEmpty())

        System.out.println("***** No se encontraron elementos");
        demás
        System.out.println("\n***** Iniciar listado...\n");

        for (Iterator<Empleado> itEmployee = listEmployees.iterator(); itEmployee.hasNext();) { Empleado objEmployee = (Empleado)
        itEmployee.next(); System.out.print(" Nombre: System.out.print("Apellido :
        System.out.println("Salario: " +
        " + objEmployee.getstFirstName() + " | "; " | ");
        objEmployee.getdSalary()); Set<Certificado> + objEmpleado.getstApellido() +
        setCertificates = objEmployee.getrelCertificates()pr (Iterator<Certificado> itCertificate =
        setCertificates.iterator(); itCertificate.hasNext();) {

        Certificado objCertificate = (Certificado) itCertificate.next(); System.out.println("Certificado: "
        + objCertificate.getstCertName());

        }

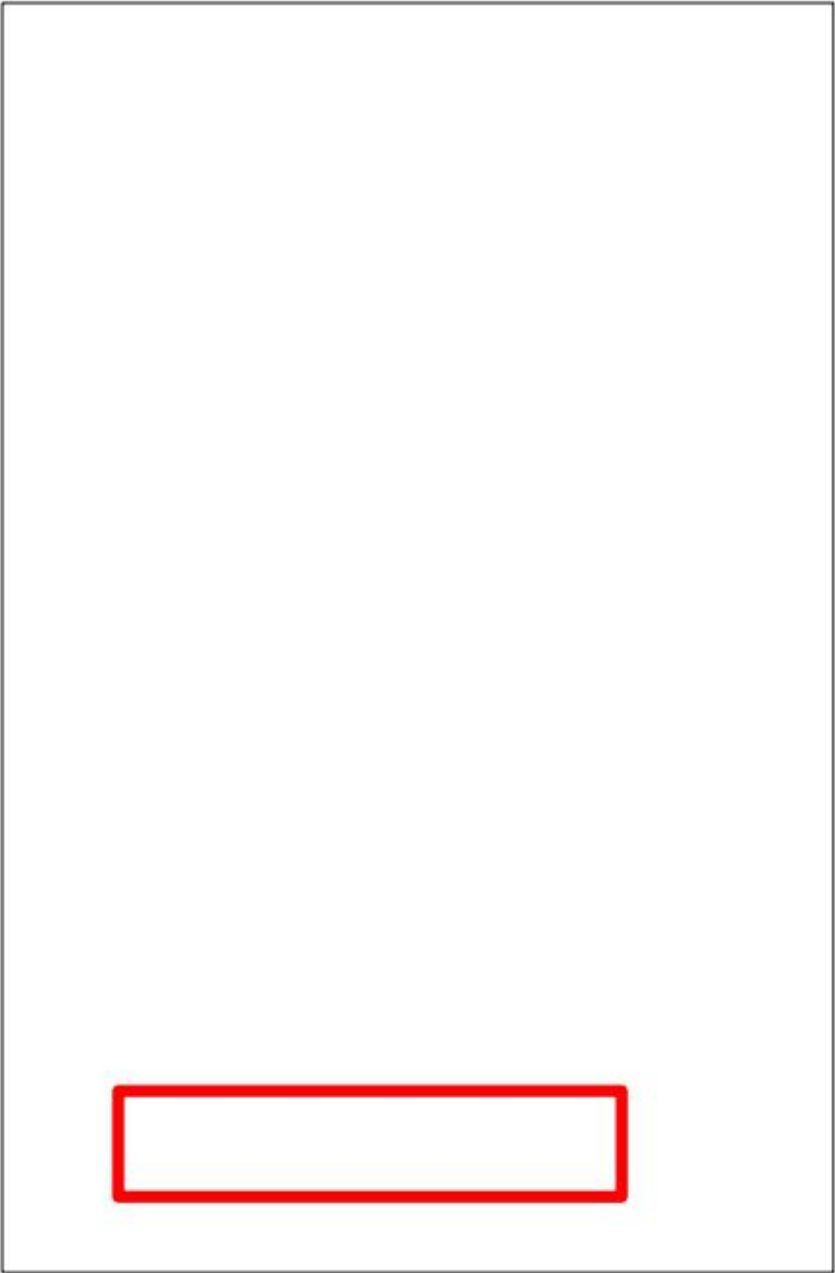
        } txDB.commit(); //finaliza la transacción
    } catch (HibernateException hibe) { if (txDB != null)
        txDB.rollback(); //algo
        salió mal, así que revertir hibe.printStackTrace(); } finalmente { hibSession.close(); //
        cerrar sesión de hibernación

    }

}
```


Configuración de muchos a muchos. Paso 8

8) Agregue una nueva asignación de campo SET en el lado izquierdo (src/main/resources/Employee.hbm.xml)



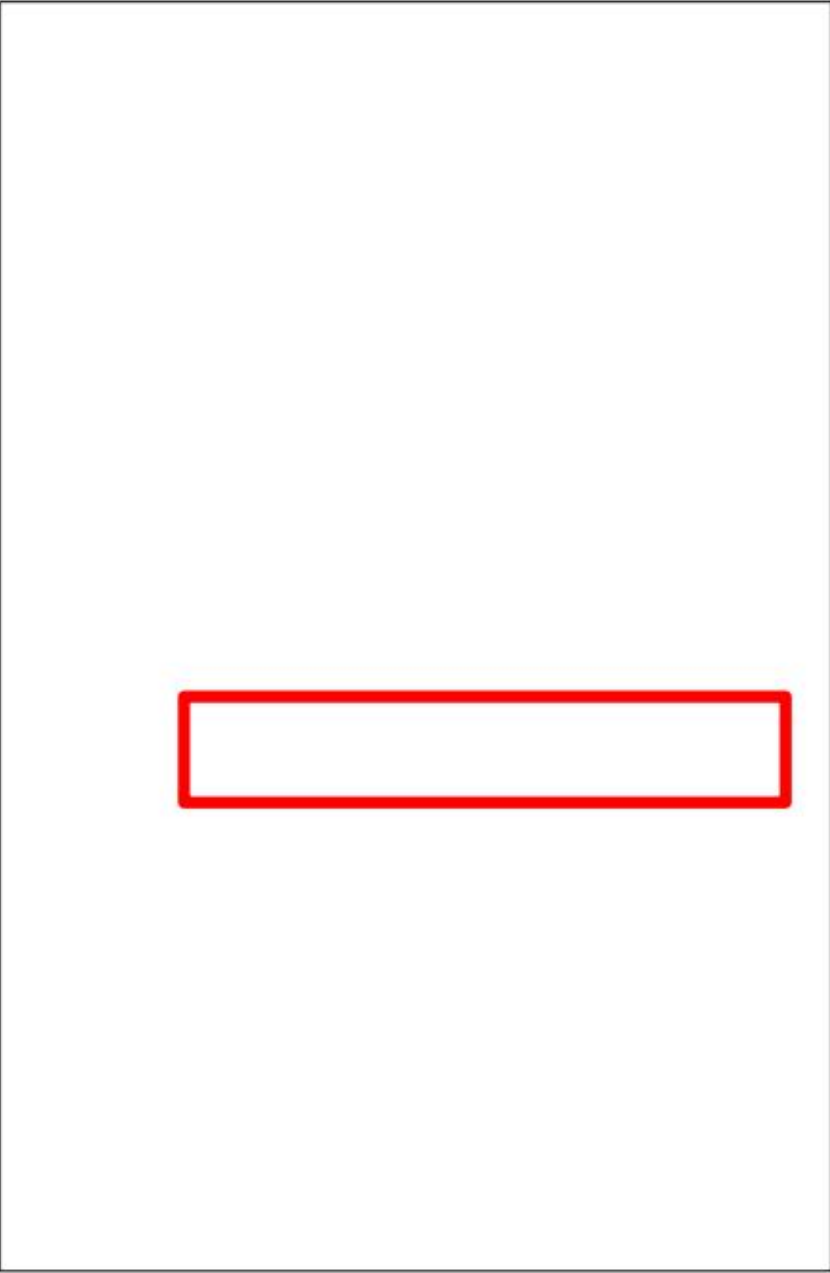
```
<?xml versión = "1.0" codificación = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//
Hibernate/Hibernate Mapping DTD//ES"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<mapeo-hibernación>
  < nombre de clase = "DOMINIO.Empleado" tabla = "Empleado"> < atributo
    meta = "descripción-clase">
      Esta clase contiene los detalles del empleado. </meta> <id
    nombre =
      "iEmpID" tipo = "int" columna = "emplID">
        <generador clase="nativo"/>
      </id>
      <!-- https://www.tutorialspoint.com/hibernate/hibernate_many_to_many_mapping.htm -->
      <set nombre = "relCertificates" cascade="save-update" table="EmpCert">
        < columna clave = "ID de empleado"/>
        < columna de muchos a muchos = "certificadoID" clase="DOMINIO.Certificado"/>
      </conjunto>

      < nombre de propiedad = "stFirstName" columna = "firstname" tipo = "cadena"/>
      < nombre de propiedad = "apellido" columna = "apellido" tipo = "cadena"/>
      < nombre de propiedad = "dSalario" columna = "salario" tipo = "doble"/>
    </clase>
  </hibernate-mapping>
```

PruebaHibernar

Y el programa principal:



```
clase pública TestHibernateMySQL {

    /*
     * -----
     * PROGRAMA PRINCIPAL
     * -----
     */

    principal vacío estático público (cadena [] stArgs) {

        //Crear nuevos objetos DAO para operaciones CRUD
        EmpleadoDAO objEmpleadoDAO = nuevo EmpleadoDAO();
        CertificateDAO objCertificateDAO = nuevo CertificateDAO();

        //TRUNCAR TABLAS. Eliminar todos los registros de las tablas
        objEmployeeDAO.deleteAllItems();
        objCertificateDAO.deleteAllItems();

        /* Agregar registros en la base de datos */ Certificado
        objCert1 = objCertificateDAO.addCertificate("MBA"); Certificado objCert2 =
        objCertificateDAO.addCertificate("PMP");

        //Conjunto de certificados
        HashSet<Certificate> hsetCertificates = new HashSet<Certificate>(); hsetCertificates.add(objCert1);
        hsetCertificates.add(objCert2);

        /* Agregar registros en la base de datos */ Empleado
        objEmp1 = objEmployeeDAO.addEmployee("Alfred", "Vincent", 4000, hsetCertificates); Empleado objEmp2 = objEmployeeDAO.addEmployee("John",
        "Gordon", 3000, hsetCertificates);

        /* Actualizar el campo de salario del empleado */
        objEmployeeDAO.updateEmployee(objEmp1.getiEmpID(), 5000); /* Enumera todos los
        empleados */ objEmployeeDAO.listEmployees(); /*
        Eliminar un empleado de la base de datos */
        objEmployeeDAO.deleteEmployee(objEmp2.getiEmpID()); /*
        Enumere todos los certificados */ objCertificateDAO.listCertificates(); /* Enumera
        todos los empleados */ objEmployeeDAO.listEmployees();

        //Cerrar la fábrica de sesiones de hibernación global
        HibernateUtil.shutdownSessionFactory();

    }

}
```

8. ACTIVIDADES PARA LA PRÓXIMA SEMANA

Actividades propuestas

Consulta las sugerencias de ejercicios que encontrarás en el “Aula Virtual”. Estas actividades son opcionales y no evaluables, pero comprenderlas es esencial para resolver la tarea evaluable que tenemos por delante.

En breve encontrará las soluciones propuestas.

9. BIBLIOGRAFÍA

Recursos

- Punto de tutoriales. Tutorial de hibernación. <https://www.tutorialspoint.com/hibernate/index.htm>

- Introducción a Hibernate 6.

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

- Guía del usuario de Hibernate ORM 6.0.0.CR1.

https://docs.jboss.org/hibernate/orm/6.0/userguide/html_single/Hibernate_User_Guide.html#pc

- Hibernar: guardar, persistir, actualizar, fusionar, guardar o actualizar.

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

- Josep Cañellas Bornas, Isidre Guixà Miranda. Accés a dades. Desarrollo de aplicaciones multiplataforma. Comunes creativos. Departamento de Enseñanza, Institut Obert de Catalunya.

Dipósito legal: B. 29430-2013. <https://ioc.xtec.cat/educacio/recursos>

- Alberto Oliva Molina. Acceso a datos. UD 3. Herramientas de mapeo de objetos relacionales (ORM). IES Tubalcaín. Tarazona (Zaragoza, España).

