

---

**M6.UF2.A3**

**HIBERNATE**

**PERSISTENCIA EN**

**APLICACIONES DE**

**ESCRITORIO**

**Eduard Lara**

# INDICE

---

1. INTRODUCCIÓN HIBERNATE
2. HIBERNATE DE FORMA MANUAL
3. HIBERNATE AUTOMATICO CON EL PLUGIN JBOSS
4. CONSULTAS HIBERNATE

# 1. INTRODUCCIÓN HIBERNATE

---

## **Mapeo objeto-relacional (ORM Object-Relational Mapping)**

Técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una bbdd relacional

- **Ventajas del uso de un ORM:**

- Ayuda a reducir tiempo de desarrollo
- Abstracción de la base de datos. Reutilización
- Independencia de la BD → Migrar con facilidad
- Lenguaje propio para realizar consultas

- **Inconvenientes:**

- Aplicaciones más lentas
- Configuración del entorno más compleja

- **Herramientas:** Doctrine, Propel, ADOdb Active Record, Hibernate, Oracle Toplink, iPersist, etc.

# 1. INTRODUCCIÓN HIBERNATE

---

## Que es Hibernate

- Hibernate es una herramienta de mapeo objeto-relacional para la plataforma Java (disponible para .NET con el nombre de Nhibernate) que facilita el mapeo de atributos mediante ficheros declarativos (XML)
- Se está convirtiendo en el estándar de facto para almacenamiento persistente cuando queremos independizar la capa de negocio del almacenamiento de la información.
- Con Hibernate no emplearemos habitualmente SQL para acceder a datos, sino que el propio motor de Hibernate, mediante el uso de factorías (patrón de diseño **Factory**) construirá esas consultas para nosotros.
- Hibernate pone a disposición del diseñador un lenguaje llamado **HQL (Hibernate Query Language)** que permite acceder a los datos mediante POO.

# 1. INTRODUCCIÓN HIBERNATE

---

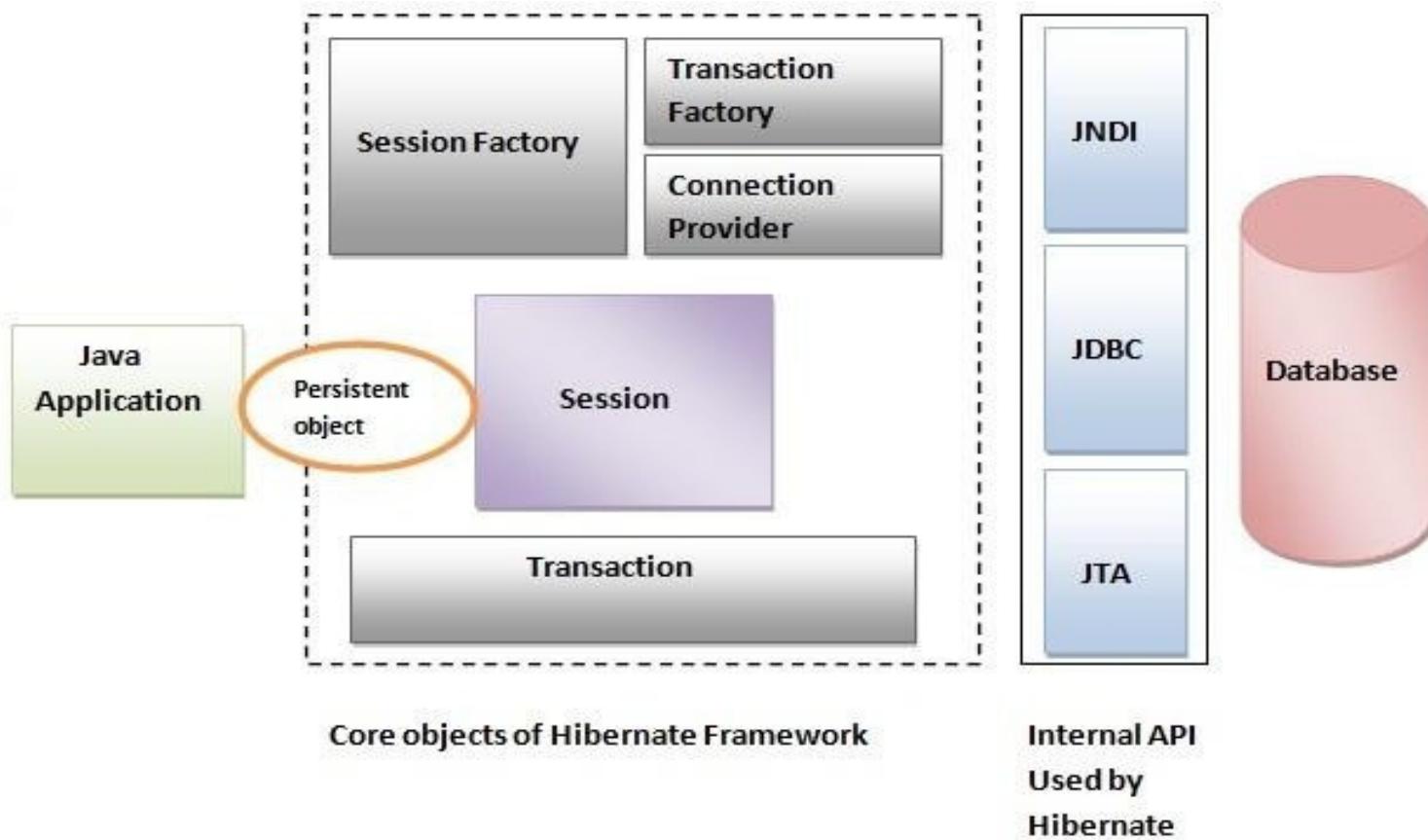
## Interfaces de Hibernate

- La interfaz **SessionFactory** (`org.hibernate.SessionFactory`): permite obtener instancias de **Session**. Esta interfaz debe compartirse entre muchos hilos de ejecución. Normalmente hay una única **SessionFactory** para toda la aplicación. Si la aplicación accede a varias bases datos se necesitará una **SessionFactory** por cada base de datos. Por otro lado, la clase `Session` nos ofrece métodos como **save(Object)**, **createQuery(String)**, **beginTransaction()**, **close()**, etc.
- La interfaz **Configuration** (`org.hibernate.cfg.Configuration`): se utiliza para configurar Hibernate. La aplicación utiliza una instancia de **Configuration** para especificar la ubicación de los documentos que indican el mapeado de los objetos y a continuación crea la **SessionFactory**
- La interfaz **Query** (`org.hibernate.Query`): permite ejecutar consultas y controla cómo se realizan. Las consultas se escriben en **HQL**
- La interfaz **Transaction** (`org.hibernate.Transaction`): permite realizar modificaciones o consultas en la base de datos según el paradigma ACID

# 1. INTRODUCCIÓN HIBERNATE

## Arquitectura Hibernate

Una forma de imaginarse esta arquitectura es:



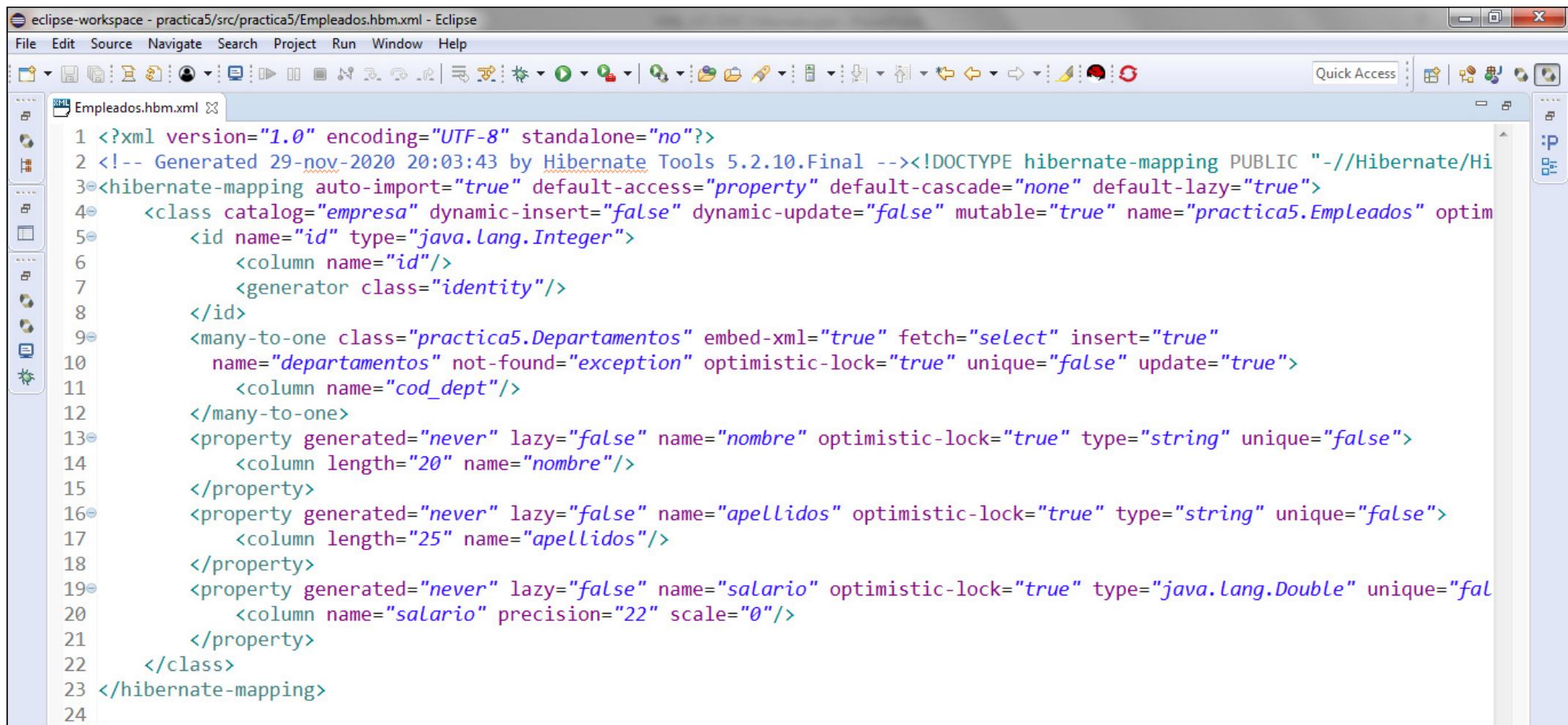
# 1. INTRODUCCIÓN HIBERNATE

---

## Estructura de los ficheros de mapeo (**hbm.xml**)

- Hibernate utiliza unos ficheros de mapeo para relacionar las tablas de la base de datos con los objetos Java. Estos ficheros están en formato XML y tienen la extensión **.hbm.xml**.
- En el proyecto anterior se han creado los ficheros **Empleados.hbm.xml** y **Departamentos.hbm.xml** asociados a las tablas emple y depart respectivamente.
- Veamos la estructura de estos ficheros

# 1. INTRODUCCIÓN HIBERNATE



The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace - practica5/src/practica5/Empleados.hbm.xml - Eclipse". The menu bar includes File, Edit, Source, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, and Run. The left sidebar contains project navigation and quick access icons. The main editor area displays the XML configuration file "Empleados.hbm.xml" with syntax highlighting for XML tags and attributes. The code defines a Hibernate mapping for the "Empleados" class.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate-Mapping//EN" "http://hibernate.sourceforge.net/hibernate-mapping/3.0.dtd">
3 <hibernate-mapping auto-import="true" default-access="property" default-cascade="none" default-lazy="true">
4   <class catalog="empresa" dynamic-insert="false" dynamic-update="false" mutable="true" name="practica5.Empleados" optimistic-lock="true">
5     <id name="id" type="java.lang.Integer">
6       <column name="id"/>
7       <generator class="identity"/>
8     </id>
9     <many-to-one class="practica5.Departamentos" embed-xml="true" fetch="select" insert="true"
10      name="departamentos" not-found="exception" optimistic-lock="true" unique="false" update="true">
11       <column name="cod_dept"/>
12     </many-to-one>
13     <property generated="never" lazy="false" name="nombre" optimistic-lock="true" type="string" unique="false">
14       <column length="20" name="nombre"/>
15     </property>
16     <property generated="never" lazy="false" name="apellidos" optimistic-lock="true" type="string" unique="false">
17       <column length="25" name="apellidos"/>
18     </property>
19     <property generated="never" lazy="false" name="salario" optimistic-lock="true" type="java.lang.Double" unique="false">
20       <column name="salario" precision="22" scale="0"/>
21     </property>
22   </class>
23 </hibernate-mapping>
24
```

# 1. INTRODUCCIÓN HIBERNATE

---

- **hibernate-mapping**: todos los ficheros de mapeo comienzan y acaban con esta etiqueta.
- **class**: esta etiqueta engloba la clase con sus atributos indicando el mapeo a la tabla de la base de datos.
  - **name**: nombre de la clase
  - **table**: nombre de la tabla que representa el objeto
  - **catalog**: nombre de la base de datos
- Dentro de class distinguimos la etiqueta **id** en la cual se indica en **name** el campo que representa al atributo clave y en **column** su nombre sobre la tabla, en **type** el tipo de datos. También tenemos la propiedad **generator** que indica la naturaleza del campo clave. En este caso es “identity” porque es un campo generado por la base de datos. Este atributo se correspondería con el campo id de la tabla Empleados.

# 1. INTRODUCCIÓN HIBERNATE

---

- Resto de atributos se indican en las etiquetas **property** asociando el nombre del campo de la clase con el nombre de la columna de la tabla y el tipo de datos.
- La etiqueta **set** se utiliza para mapear colecciones. Dentro de set se definen varios atributos:
  - **name**: indica el nombre del atributo generado
  - **table**: el nombre de la tabla de donde se tomará la colección
  - El elemento **key** define el nombre de la columna identificadora en la asociación
  - El elemento **one-to-many** define la relación (un departamentos puede tener muchos empleados)
  - **class**: indica de qué tipo son los elementos de la colección.
- Los tipos que declaramos en los ficheros de mapeo no son tipos de datos Java ni SQL. Se llaman **tipos de mapeo Hibernate**.

# 1. INTRODUCCIÓN HIBERNATE

---

## Clases persistentes

- Las clases referenciadas en el elemento **class** de los ficheros de mapeo hacen referencia a las clases generadas en nuestro proyecto como Empleados.java y Departamentos.java. A estas clases se les llama **clases persistentes**.
- Las clases persistentes son las clases que implementan las entidades del problema, deben implementar la interfaz **Serializable**. Equivalen a una tabla de la base de datos, y un registro o fila es un objeto persistente de esa clase
- Estas clases representan un objeto emple y un objeto depart, por lo tanto, podemos crear objetos empleados y departamentos a partir de ellas. Tienen unos atributos privados y unos métodos públicos (*getters* y *setters*) para acceder a los mismos.
- A estas reglas se les suele llamar modelo de programación **POJO** (Plain Old Java Objects)

# 1. INTRODUCCIÓN HIBERNATE

```
Empleados.java ✘
1 package practica5;
2 // Generated 29-nov-2020 20:03:33 by Hibernate Tools 5.2.10.Final
3
4 /**
5  * Empleados generated by hbm2java
6  */
7 public class Empleados implements java.io.Serializable {
8
9     private Integer id;
10    private Departamentos departamentos;
11    private String nombre;
12    private String apellidos;
13    private Double salario;
14
15    public Empleados() {
16    }
17
18    public Empleados(Departamentos departamentos, String nombre,
19                      String apellidos, Double salario) {
20        this.departamentos = departamentos;
21        this.nombre = nombre;
22        this.apellidos = apellidos;
23        this.salario = salario;
24    }
25
26    public Integer getId() {
27        return this.id;
28    }
29
30    public void setId(Integer id) {
31        this.id = id;
32    }
33
34    public Departamentos getDepartamentos() {
35        return this.departamentos;
36    }
37
38    public void setDepartamentos(Departamentos departamentos) {
39        this.departamentos = departamentos;
40    }
41
42    public String getNombre() {
43        return this.nombre;
44    }
45
46    public void setNombre(String nombre) {
47        this.nombre = nombre;
48    }
49
50}
```

# 1. INTRODUCCIÓN HIBERNATE

## Sesiones y objetos Hibernate

- Como ya hemos visto, para poder utilizar los mecanismos de persistencia de Hibernate se debe inicializar el entorno Hibernate y obtener un objeto **Session** utilizando la clase **SessionFactory**.

```
//Inicializa el entorno Hibernate: carga el fichero hibernate.cfg.xml
Configuration cfg = new Configuration().configure();

//Crea el ejemplar de SessionFactory. Se necesita crear un objeto
//StandardServiceRegistry que contiene la lista de servicios de Hibernate
//El ejemplar de SessionFactory se crea normalmente solo una vez y se utiliza
//para crear todas las sesiones relacionadas con el contexto dado (singleton)
StandardServiceRegistry registry=
    new StandardServiceRegistryBuilder().configure().build();

final SessionFactory sessionFactory = cfg.buildSessionFactory (registry) ;

//Obtiene un objeto Session
Session session = sessionFactory.openSession();
```

# 1. INTRODUCCIÓN HIBERNATE

## Transacciones

- Un objeto **Session** de Hibernate representa una única unidad de trabajo. Al crear la sesión se crea la transacción para dicha sesión. Se deben cerrar las sesiones.

```
//Abrimos sesión e iniciamos una transacción
Session session = sesion.openSession();
Transaction tx = session.beginTransaction();

// ... código de persistencia ...

//Hacemos el commit de la transaccion y cerramos la sesión
tx.commit();
session.close();
```

- El método **beginTransaction()** marca el inicio de una transacción. El método **commit()** la valida; y el método **rollback()** deshace los cambios.

# 1. INTRODUCCIÓN HIBERNATE

---

## Estados de un objeto Hibernate

- **Transitorio** (Transient) : Un objeto es transitorio si ha sido recién instanciado y no está asociado a una Session de Hibernate. No tiene una representación persistente en la bbdd y no se le ha asignado un identificador.
- **Persistente** (Persistent) : Un objeto estará en este estado cuando ya está almacenado en la bbdd. Puede haber sido guardado o cargado pero se encuentra en el ámbito de una **Session**.
- **Separado** (Detached) : Una instancia separada es un objeto que se ha hecho persistente pero su sesión ha sido cerrada. La referencia al objeto todavía es válida, y podría ser modificado. Para hacer persistentes dichas modificaciones debemos asociar el objeto a una nueva **Session**

# 1. INTRODUCCIÓN HIBERNATE

---

## Carga de objetos

Para la carga de objetos usaremos los siguientes métodos de **Session**:

- **<T> T load ( Class<T> Clase, Serializable id )** → Devuelve la instancia persistente de la clase indicada con el identificador dado. Si la instancia no existe el método lanza una excepción.
- **Object load (String nombreClase, Serializable id)** → Lo mismo que antes pero indicando nombre de la clase
- **<T> T get (Class<T> Clase, Serializable id )** → Lo mismo pero si la instancia no existe devuelve *null*
- **Object get (String nombreClase, Serializable id)** → Lo mismo que en el caso anterior pero indicando el nombre de la clase.

# 1. INTRODUCCIÓN HIBERNATE

## Carga de objetos: Ejemplos

```
Departamentos dep = new Departamentos();
try {
    dep = (Departamentos) session.load(Departamentos.class, id);
    System.out.println(dep);
} catch (ObjectNotFoundException o) {
    System.out.println ("No existe el departamento");
}
```

El método load funciona con excepciones si no existe el objeto

```
Departamentos dep = (Departamentos) session.get (Departamentos.class, id);
if (dep==null) {
    System.out.println ("No existe el departamento");
}
else {
    System.out.println("Nombre Dep: "+dep.getNombre());
}
```

El método get no necesita de excepciones.  
Si no existe devuelve un null

# 1. INTRODUCCIÓN HIBERNATE

---

## **Almacenamiento, modificación y borrado de objetos**

Disponemos de los siguientes métodos:

- Serializable save (Object obj) → Guarda el objeto que se pasa como argumento en la base de datos. Hace que la instancia transitoria del objeto sea persistente
- void update (Object obj) → Actualiza en la bbdd el objeto que se pasa como argumento. El objeto a modificar debe ser cargado con el método load() o get()
- void delete (Object obj) → Elimina de la bbdd el objeto que se pasa como argumento. El objeto a eliminar debe ser cargado con el método load() o get().

# 1. INTRODUCCIÓN HIBERNATE

## Almacenamiento, modificación y borrado de objetos: Ejemplos

```
//Departamentos dep = new Departamentos(nombre, direccion, objetivo);
Departamentos dep = new Departamentos();
dep.setNombre(nombre);
dep.setDireccion(direccion);
dep.setObjetivos(objetivo);
session.save(dep);
```

Generacion y  
guardado de un  
nuevo departamento  
con el método save

```
Departamentos dep = session.get(Departamentos.class, id);
if (dep==null) System.out.print("No existe departamento con ese id");
else {
    System.out.print("Introduce nombre nuevo de departamento ("+dep.getNombre()+ "):");
    dep.setNombre(reader.next());
    System.out.print("Introduce objetivos nuevos ("+dep.getObjetivos() + "):");
    dep.setObjetivos(reader.nextInt());
    session.update(dep);
}
```

Para modificar un  
objeto, primero hemos  
de cargarlo, realizar la  
modificaciones y, por  
último, utilizar el  
método update().

# 1. INTRODUCCIÓN HIBERNATE

## Almacenamiento, modificación y borrado de objetos: Ejemplos

```
Empleados emp = session.get (Empleados.class, id);
if (emp==null)
    System.out.print("Empleado no existe");
else
    session.delete (emp);
```

Borrado de un empleado usando previamente el método get

```
Departamentos dep = new Departamentos();
try {
    dep = (Departamentos) session.load(Departamentos.class, id);
    session.delete(dep);
    tx.commit();
    System.out.println ("Departamento eliminado");
}catch (ObjectNotFoundException c) {
    System.out.println ("No existe el departamento");
}catch (ConstraintViolationException c) {
    System.out.println ("No se puede eliminar. Tiene empleados");
}catch (Exception e) {
    e.printStackTrace();
}
```

Eliminación de un departamento con el método load, controlando las distintas excepciones: que el departamento no exista y que tenga empleados

## 2. HIBERNATE DE FORMA MANUAL

---

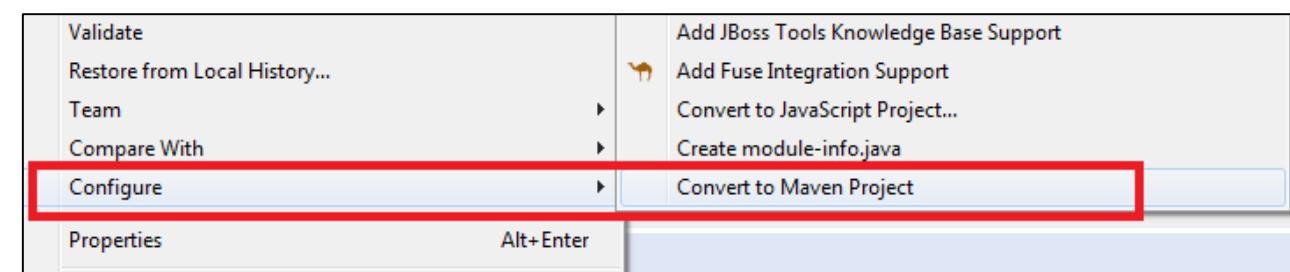
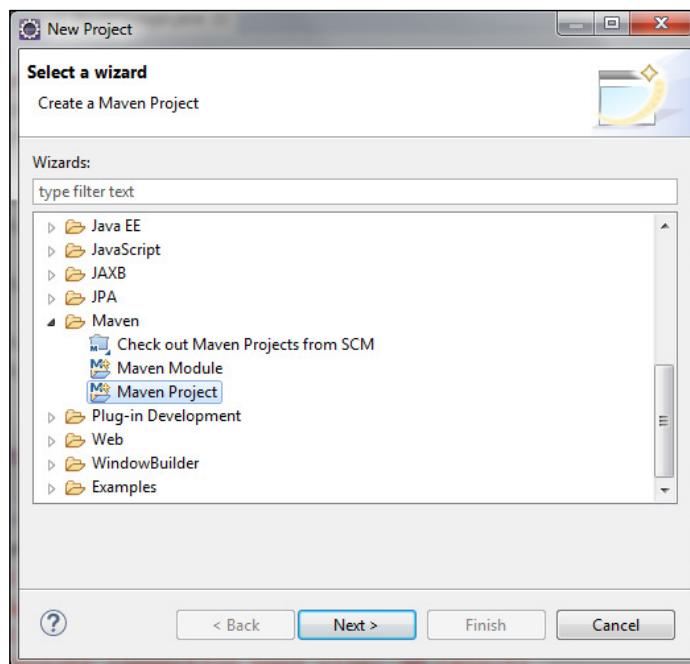
### Apache Maven

- Herramienta de gestión de proyectos de software. Basado en el concepto de un modelo de objetos de proyecto (POM),
- Maven puede administrar la construcción, informes y documentación de un proyecto a partir de una pieza central de información: el fichero POM.xml.
- Maven es un repositorio de Internet que permite añadir dependencias a nuestro proyecto sin necesidad de descargar las librerías físicamente.
- **Para ello utiliza el fichero de configuración del Proyecto Maven (pom.xml). Basta con escribir las dependencias necesarias en el fichero pom.xml y Maven se encarga de descargar automáticamente todas esas librerías.**
- **MAVEN no es necesario para HIBERNATE, pero ayuda a la obtención de las librerías**

## 2. HIBERNATE DE FORMA MANUAL

**Paso 1 (opcional con Maven).** Tenemos dos opciones:

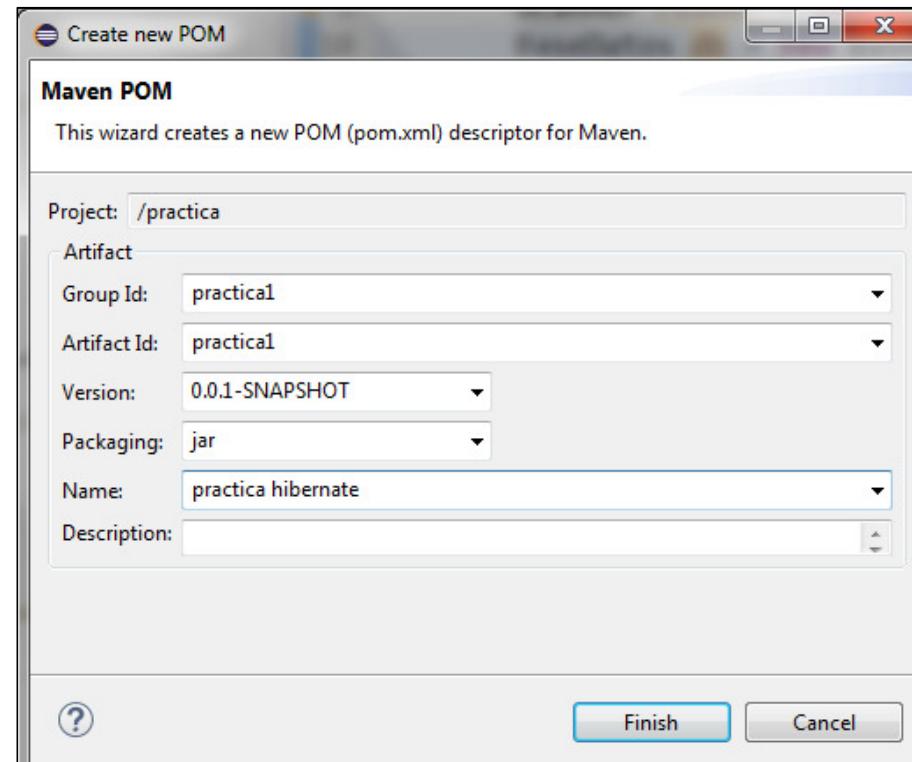
- Crear un nuevo proyecto Maven yendo a **File/New/Project/Maven Project**
- Convertir nuestro proyecto a Maven, en **Configure/Convert to Maven Project.** Esta será la opción elegida.



## 2. HIBERNATE DE FORMA MANUAL

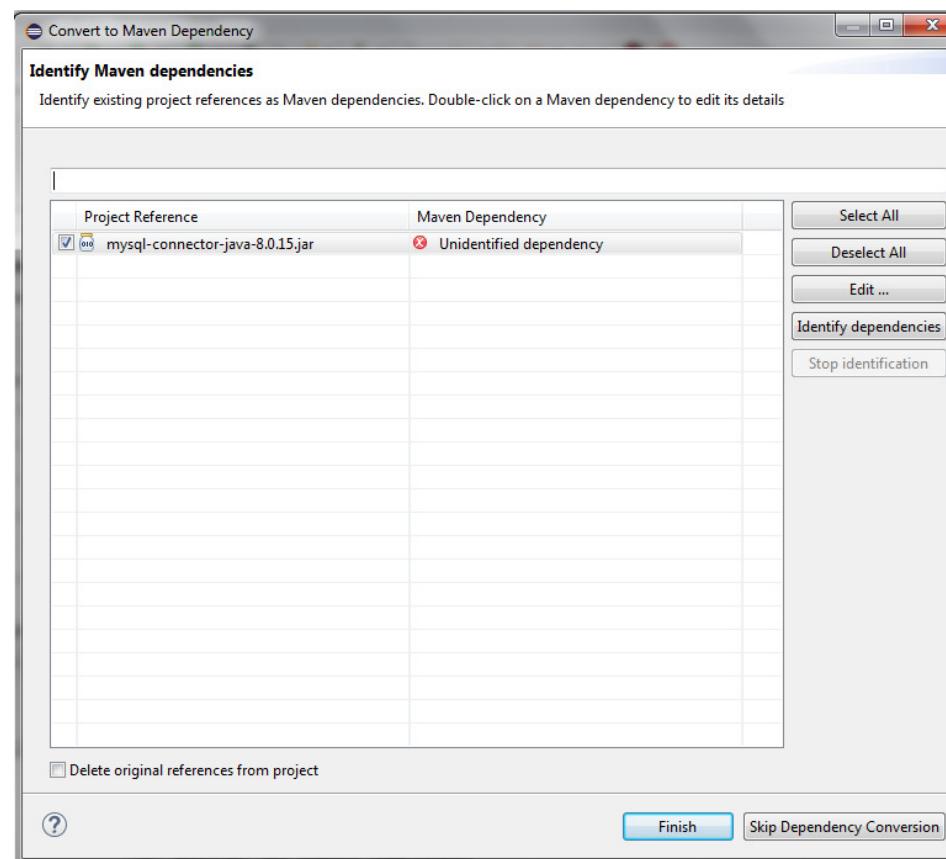
**Paso 2 (opcional con Maven).** Entra la información para el artifact y haz click en **Finish**:

- Group Id: practica1
- Artifact Id: practica1
- Name: practica Hibernate



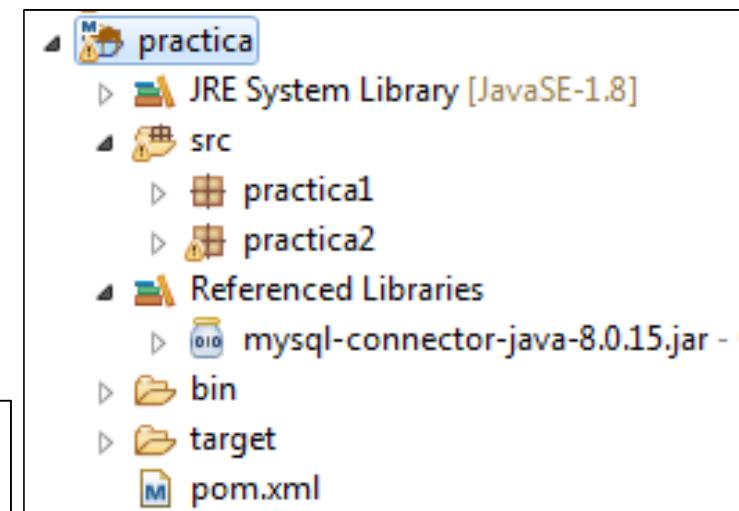
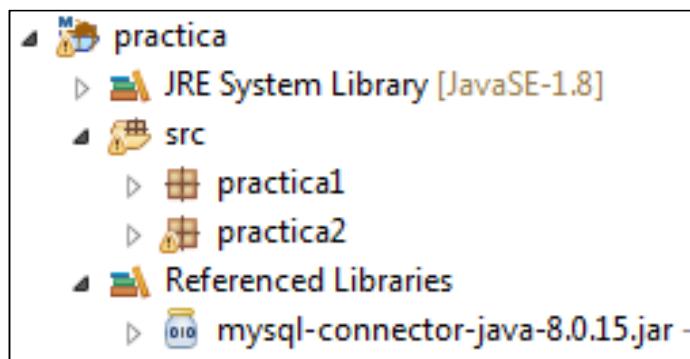
## 2. HIBERNATE DE FORMA MANUAL

**Paso 3 (opcional con Maven).** Detecta una dependencia en el proyecto pero no la reconoce. Hacemos click en Finish.

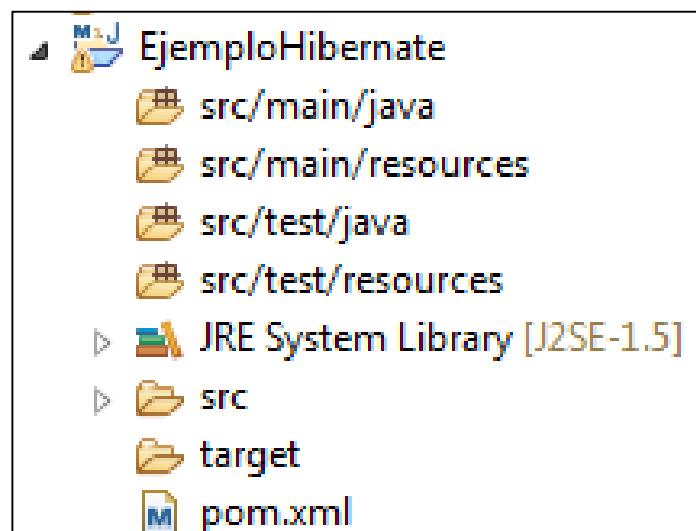


## 2. HIBERNATE DE FORMA MANUAL

**Paso 4 (opcional con Maven).** Eclipse generará la estructura para el proyecto:



Si hubiésemos creado un Nuevo Proyecto Maven, se hubiera generado la siguiente estructura del proyecto



## 2. HIBERNATE DE FORMA MANUAL

**Paso 5 (opcional con Maven).** Accede al fichero **pom.xml** para sumar las dependencias para las librerías Hibernate y MySQL Connector Java. Agrega el siguiente código XML entre el elemento `<project></project>`:

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.2.6.Final</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.40</version>
    </dependency>
</dependencies>

</project>
```

## 2. HIBERNATE DE FORMA MANUAL

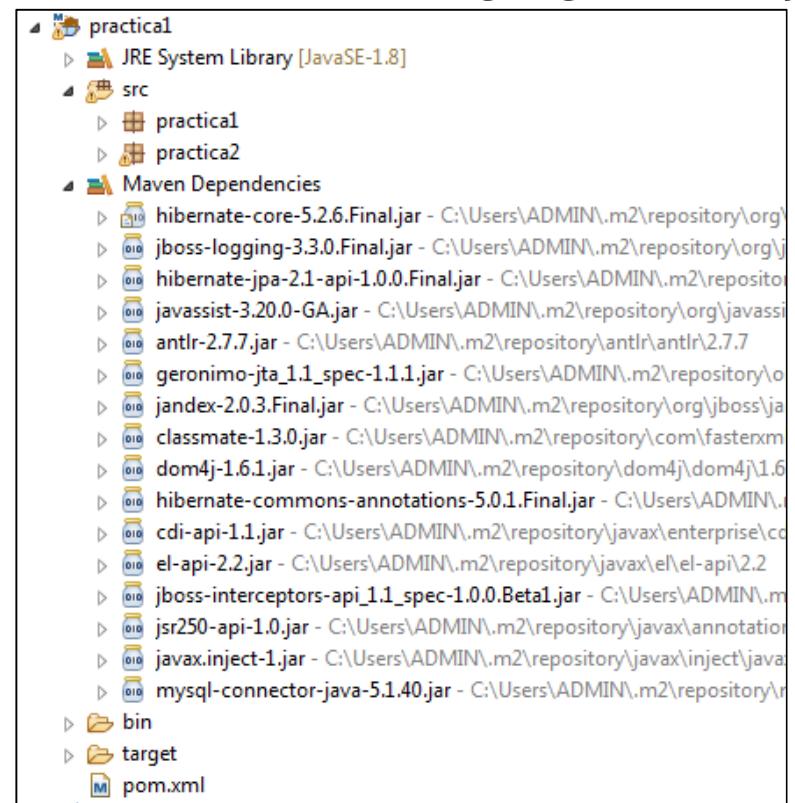
**Paso 6 (opcional con Maven).** Si se tiene instalada la versión del jdk de java 9 o superior, se debe de agregar en el fichero **pom.xml** las siguientes dependencias:

```
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
    <version>2.3.2</version>
</dependency>
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>2.3.2</version>
</dependency>
</dependencies>
```

Importante

## 2. HIBERNATE DE FORMA MANUAL

**Paso 7 (opcional con Maven).** Haz click en **Save (Ctrl + S)** y Maven automáticamente descargará los ficheros JAR de dependencias: Hibernate core y el driver MySQL Connector Java. Se pueden ver los ficheros JAR agregados bajo la entrada del Proyecto **Maven Dependencies**.



# 2. HIBERNATE DE FORMA MANUAL

**Paso 1 (sin Maven).** Descarga las librerías de Hibernate de la siguiente URL:

<http://hibernate.org/orm/>

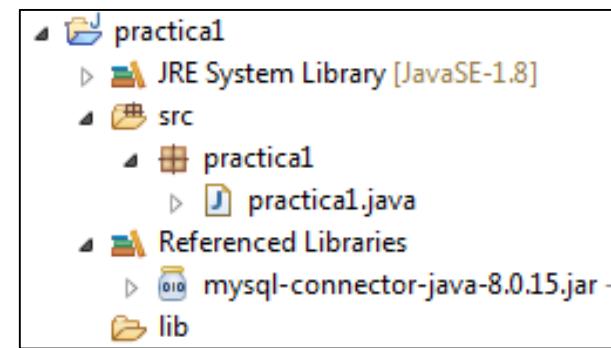
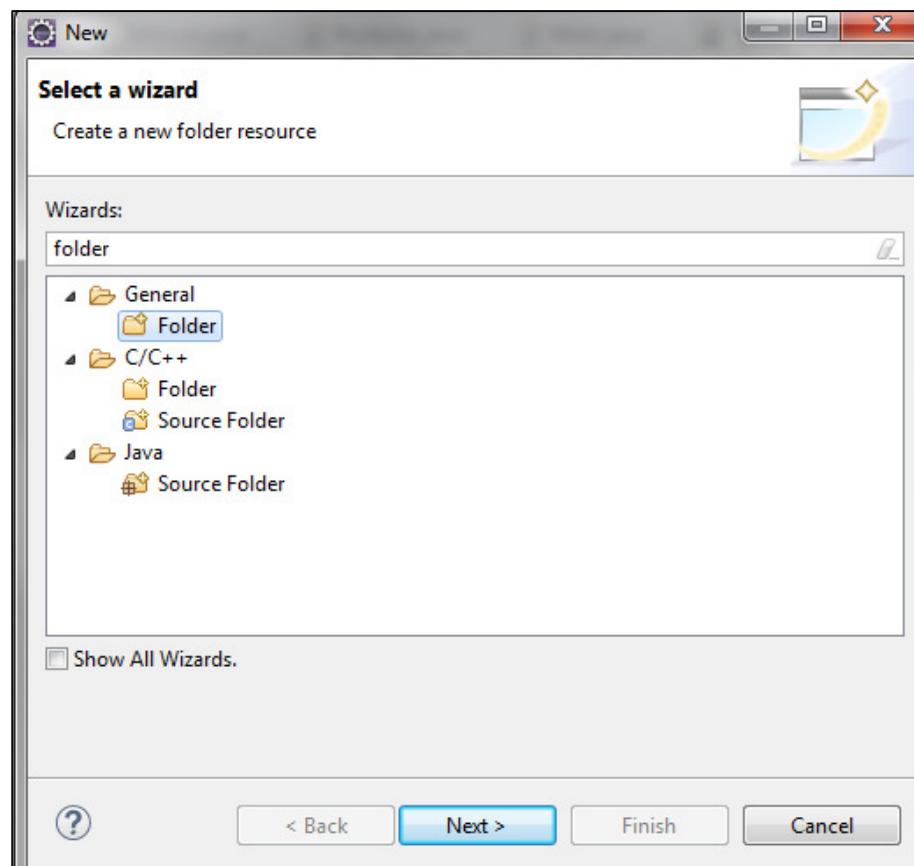
<https://sourceforge.net/projects/hibernate/>

The screenshot shows the official Hibernate website at [hibernate.org/orm/](http://hibernate.org/orm/). The page features a navigation bar with links for ORM, Search, Validator, OGM, Tools, Others, Blog, Forums, Community, and Follow. The main content area is titled "Hibernate ORM" with the subtitle "Your relational data. Objectively." It includes a brief description of idiomatic persistence for Java and relational databases, a sidebar with links for About, Releases, Documentation, and Books, and a "Latest news" section announcing the release of Hibernate ORM 5.3.20.Final.

The screenshot shows the SourceForge project page for Hibernate at [sourceforge.net/projects/hibernate/files/hibernate-orm/](https://sourceforge.net/projects/hibernate/files/hibernate-orm/). The page has a dark theme and displays the Hibernate logo. It includes tabs for Summary, Files (which is selected), Reviews, Support, Issue Tracker, GitHub, and more. A prominent green button allows users to "Download Latest Version" of hibernate-search-5.8.0.Final-dist.zip (35.0 MB). Below the download button is a "Get Updates" link and a feed icon. The file list table shows two recent releases: 5.4.24.Final (modified 2020-11-17) and 5.3.20.Final (modified 2020-11-16). A sidebar on the right encourages users to "Get latest updates about Open Source Projects".

## 2. HIBERNATE DE FORMA MANUAL

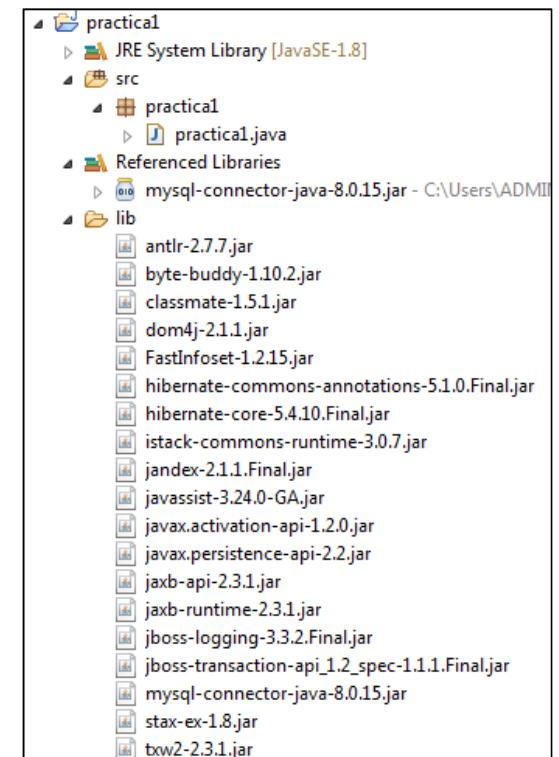
**Paso 2 (sin Maven).** Crea un nueva carpeta “lib” en el proyecto.



## 2. HIBERNATE DE FORMA MANUAL

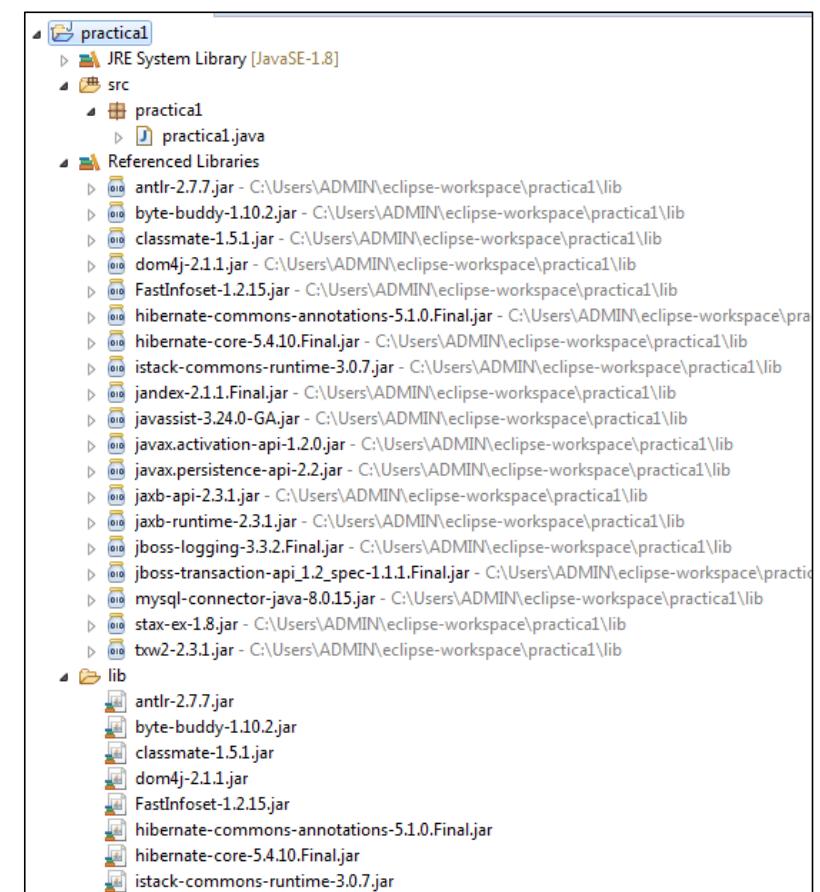
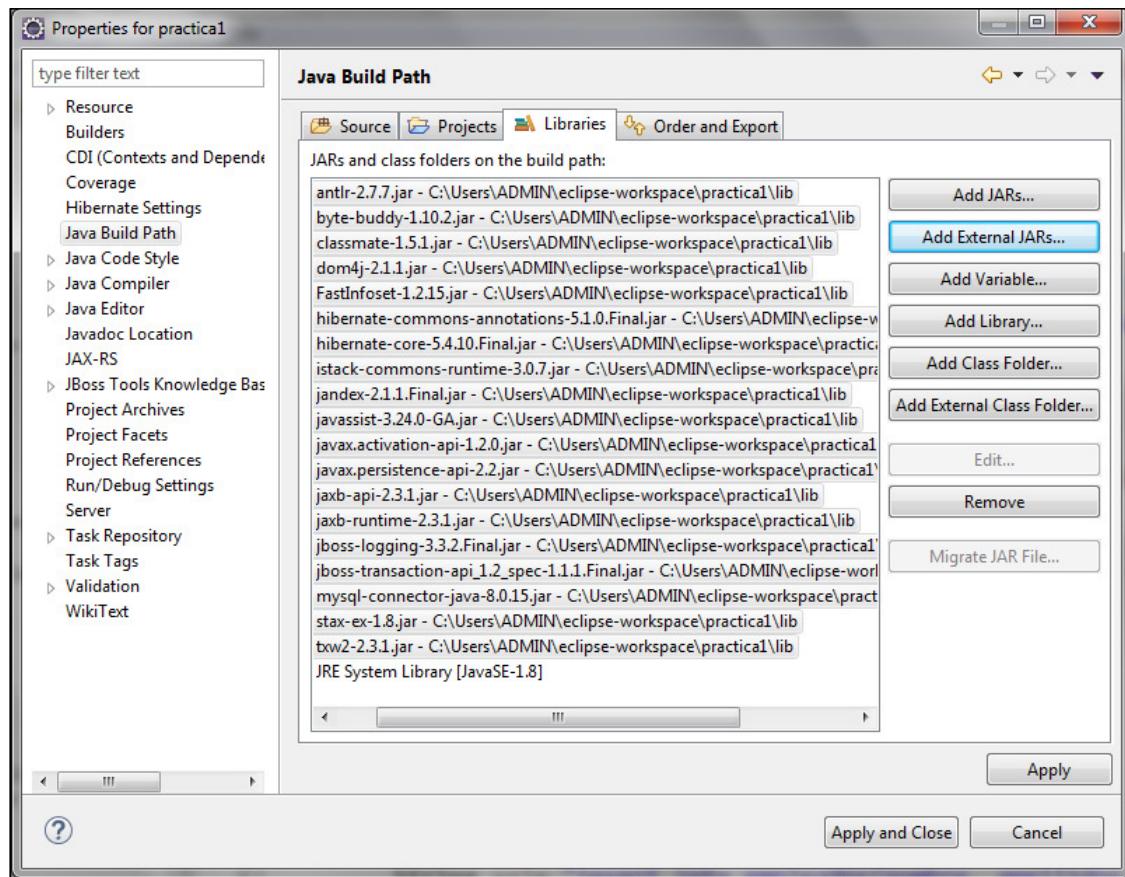
**Paso 3 (sin Maven).** Copia todos los ficheros *\*.jar* files de la carpeta “*hibernate-release-xxx/lib/required*” y pégalos en la carpeta “*lib*”. Copia también el *mysql-connector-java-xxx-bin.jar* a la carpeta “*lib*” del proyecto.

Nombre	Tipo	Tamaño comprimido	Protegido ...
antlr-2.7.7.jar	Executable Jar File	412 KB	No
byte-buddy-1.10.2.jar	Executable Jar File	2.853 KB	No
classmate-1.5.1.jar	Executable Jar File	59 KB	No
dom4j-2.1.1.jar	Executable Jar File	298 KB	No
FastInfoSet-1.2.15.jar	Executable Jar File	281 KB	No
hibernate-commons-annotations-5.1.0.Final.jar	Executable Jar File	62 KB	No
hibernate-core-5.4.10.Final.jar	Executable Jar File	6.273 KB	No
istack-commons-runtime-3.0.7.jar	Executable Jar File	22 KB	No
jandex-2.1.1.Final.jar	Executable Jar File	178 KB	No
javassist-3.24.0-GA.jar	Executable Jar File	707 KB	No
javax.activation-api-1.2.0.jar	Executable Jar File	52 KB	No
javax.persistence-api-2.2.jar	Executable Jar File	133 KB	No
jaxb-api-2.3.1.jar	Executable Jar File	110 KB	No
jaxb-runtime-2.3.1.jar	Executable Jar File	959 KB	No
jboss-logging-3.3.2.Final.jar	Executable Jar File	59 KB	No
jboss-transaction-api_1.2_spec-1.1.1.Final.jar	Executable Jar File	22 KB	No
stax-ex-1.8.jar	Executable Jar File	32 KB	No
txw2-2.3.1.jar	Executable Jar File	62 KB	No



# 2. HIBERNATE DE FORMA MANUAL

**Paso 4 (sin Maven).** Agrega todas las librerías, yendo a Properties/Java Build Path/Libraries



## 2. HIBERNATE DE FORMA MANUAL

**Paso 5.** Crea la clase Empleado bajo el package practica3 (o el que corresponda), que tenga 4 campos igual que las columnas en la tabla empleados.

Creamos la clase entity class que mapeará la tabla empleados de la base de datos, usando anotaciones.

```
Empleado.java ✘
1 package practica3;
2
3 public class Empleado {
4     private int cod_emp;
5     private String nombre;
6     private String apellidos;
7     private int salario;
8 }
9
```

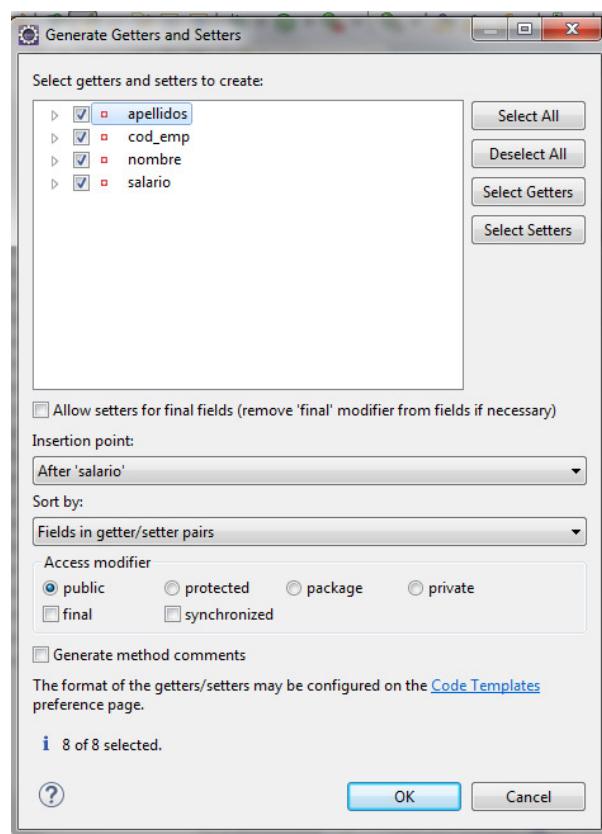
## 2. HIBERNATE DE FORMA MANUAL

**Paso 6.** Genera automáticamente los getters y los setters de estos campos (Shortcut: **Alt + Shift + S**). Genera un constructor vacío para la clase Empleado.

```
Empleado.java X
1 package practica3;
2
3 public class Empleado {
4     private int cod_emp;
5     private String nombre;
6     private String apellidos;
7     private int salario;
8 }
```

Right-click context menu:

- Toggle Comment Ctrl+ /
- Remove Block Comment Ctrl+Shift+\
- Generate Element Comment Alt+Shift+J
- Correct Indentation Ctrl+I
- Format Ctrl+Shift+F
- Format Element
- Add Import Ctrl+Shift+M
- Organize Imports Ctrl+Shift+O
- Sort Members...
- Clean Up...
- Generate Hibernate/JPA annotations...
- Override/Implement Methods...
- Generate Getters and Setters...
- Generate Delegate Methods...
- Generate hashCode() and equals()...



```
Empleado.java X
1 package practica3;
2
3 public class Empleado {
4     private int cod_emp;
5     private String nombre;
6     private String apellidos;
7     private int salario;
8
9     public int getId() {
10         return cod_emp;
11     }
12     public void setId(int id) {
13         this.cod_emp = id;
14     }
15     public String getNombre() {
16         return nombre;
17     }
18     public void setNombre(String nombre) {
19         this.nombre = nombre;
20     }
21     public String getApellidos() {
22         return apellidos;
23     }
}
```

## 2. HIBERNATE DE FORMA MANUAL

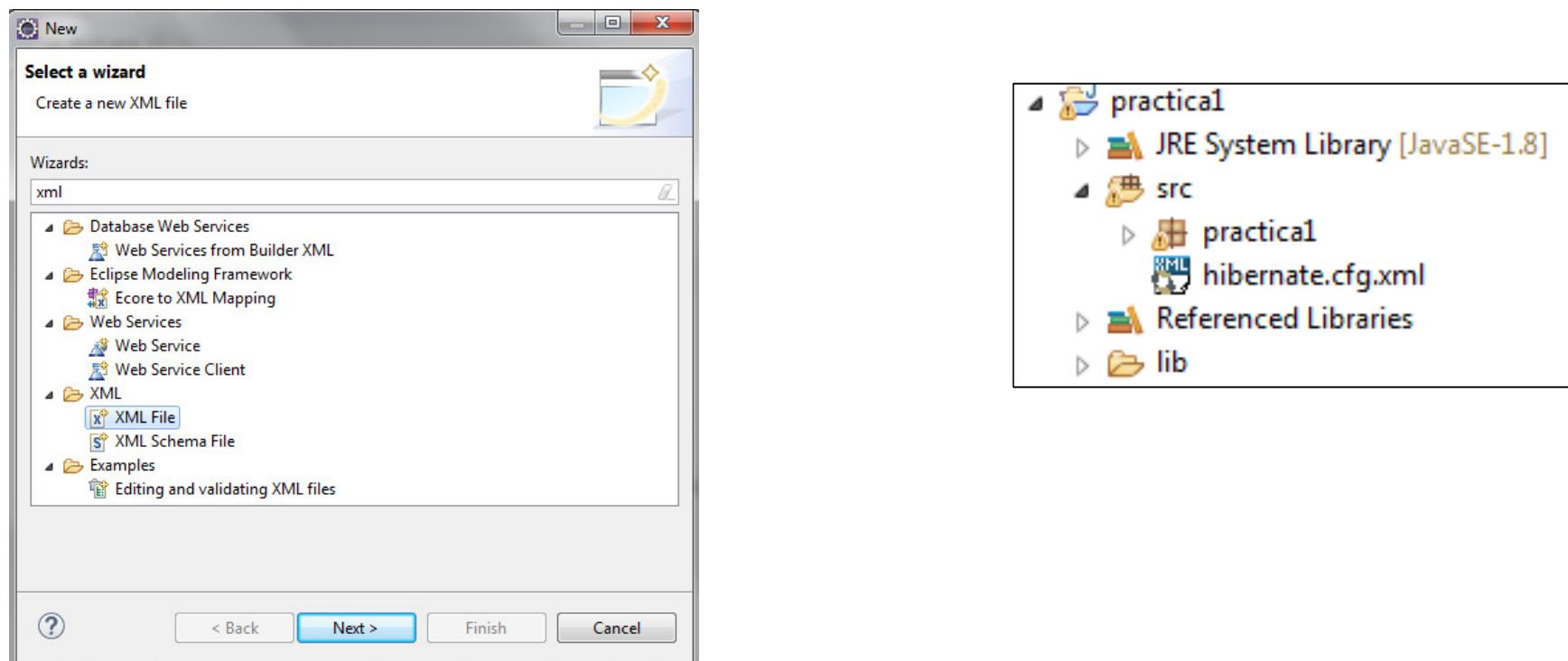
**Paso 7.** Mapea esta clase a la tabla Empleados de la base de datos, usando anotaciones de JPA:

```
Empleado.java ✘
1 package practica3;
2
3 import javax.persistence.*;
4
5 @Entity
6 @Table(name="empleados")
7 public class Empleado {
8     private int cod_emp;
9     private String nombre;
10    private String apellidos;
11    private int salario;
12
13    @Id
14    @Column(name = "cod_emp")
15    @GeneratedValue(strategy = GenerationType.IDENTITY)
16    public int getId() {
17        return cod_emp;
18    }
19    public void setId(int id) {
20        this.cod_emp = id;
21    }
}
```

- Se utiliza la anotación `@Entity` y `@Table` antes de la clase, para mapearla a la table.
- La anotación `@Id` indica a Hibernate cual es la columna ID de la tabla.
- La anotación `@Column` mapea el campo a una columna de la table de la base de datos.
- La anotación `@GeneratedValue` indica a Hibernate cual que esta columna ID es auto-increment.
- Hibernate es inteligente, y puede mapear automaticamente campos de la clase a campos de una tabla si los campos tienen el mismo nombre y automaticamente mapea tipos Java a tipos SQL.
- No se tiene que mapear explicitamente el resto de campos: nombre, apellidos y salario.

## 2. HIBERNATE DE FORMA MANUAL

**Paso 8.** Crea el fichero XML de configuración para Hibernate llamado *hibernate.cfg.xml*, en la carpeta src. Este fichero indica a Hibernate como conectar con la base de datos y que clases Java deberían ser mapeadas a que tablas de la base de datos.



## 2. HIBERNATE DE FORMA MANUAL

**Paso 9.** El fichero **hibernate.cfg.xml** especifica el connectstring de conexión a la base de datos (JDBC driver class, URL, username and password). El elemento **<mapping>** especifica la clase entidad Java necesaria para ser mapeada, en este caso `practica1.Empleado`.

```
hibernate.cfg.xml ✘
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-configuration PUBLIC
3   "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4   "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
5<hibernate-configuration>
6<session-factory>
7  <!-- Database connection settings -->
8  <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
9  <property name="connection.url">jdbc:mysql://localhost:3306/empresa</property>
10 <property name="connection.username">root</property>
11 <property name="connection.password"></property>
12 <property name="show_sql">true</property>
13 <mapping class="practica1.Empleado" />
14
15 </session-factory>
16 </hibernate-configuration>
17
```

Cambia el username y el password de la base de datos de acuerdo a nuestra base de datos. La propiedad **show\_sql** a true indica a Hibernate que imprima sentencias SQL para cada query que se haga

## 2. HIBERNATE DE FORMA MANUAL

**Paso 10.** Comenta todo lo anterior del fichero BaseDatos.java (o crea un nuevo fichero llamado GestorEmpleados), instancia una variable donde se cargará la **Session Factory** de Hibernate, e implementa todos los métodos de persistencia Hibernate:

```
BaseDatos.java ✘
1 package practica3;
2
3 import org.hibernate.Session;
4
5
6 public class BaseDatos {
7     private SessionFactory sessionFactory;
8
9     public void iniciar() {          //Iniciar hibernate.
10                                //Carga del fichero hibernate.cfg en sessionfactory
11 }
12
13     public void salir() {           //Salir Hibernate
14 }
15
16     public void insertar(Empleado emp) { //Codigo para insertar un empleado
17 }
18
19     public void listar() {           //Codigo para listar o leer un empleado
20 }
21
22     public void actualizar(Empleado emp) { //Codigo para actualizar un empleado
23 }
24
25     public void borrar(Empleado emp) {    //Codigo para borrar un empleado
26 }
```

Los nombres de los métodos son totalmente libres

## 2. HIBERNATE DE FORMA MANUAL

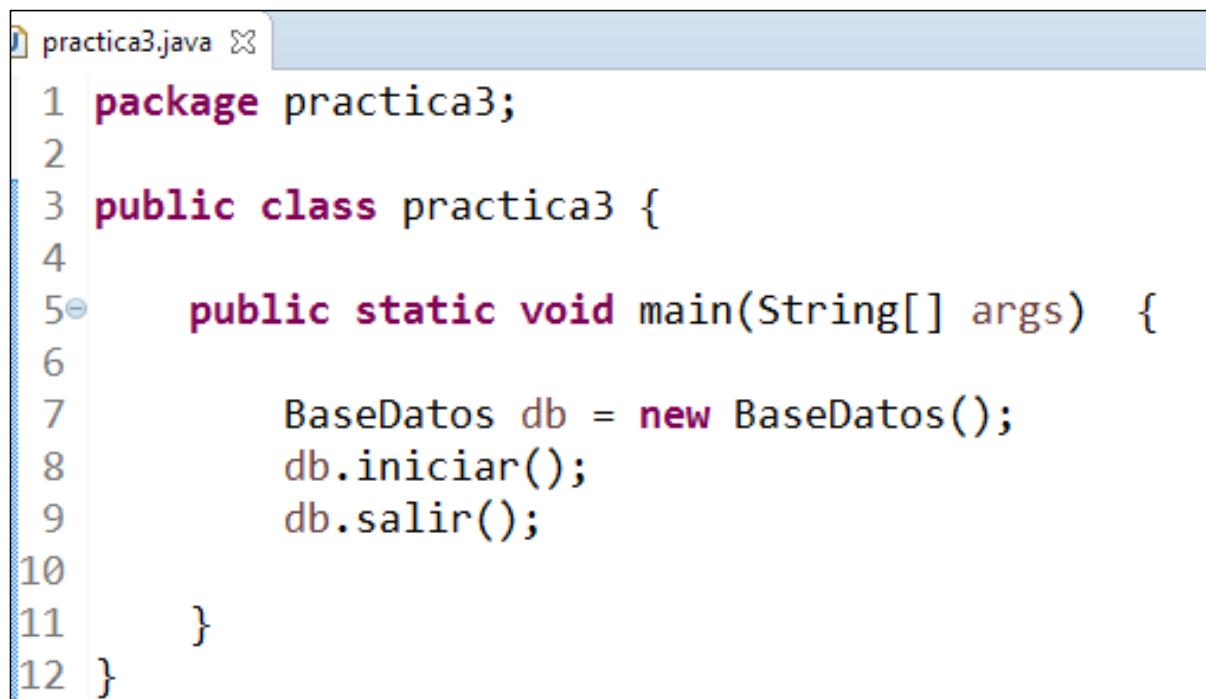
**Paso 11.** En Hibernate se realizan operaciones de base de datos a través del objeto **SessionFactory**. **SessionFactory** carga el fichero de configuración Hibernate, analiza el mapeo y crea la conexión con la base de datos. En el método **setup()** **SessionFactory** queda configurado según los parámetros del fichero **hibernate.cfg.xml**. En el método **exit()** se cierra la **SessionFactory**.

```
protected void setup() { // code to load Hibernate Session factory
    final StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
        .configure() // configures settings from hibernate.cfg.xml
        .build();
    try {
        sessionFactory = new MetadataSources(registry).buildMetadata().buildSessionFactory();
    } catch (Exception ex) {
        System.out.println("The sessionFactory was not created. Could not connect to the database");
        StandardServiceRegistryBuilder.destroy(registry);
    }
    if (sessionFactory != null)
        System.out.println("Successfully connected to the database");
}

protected void exit() {
    // code to close Hibernate Session factory
    sessionFactory.close();
}
```

## 2. HIBERNATE DE FORMA MANUAL

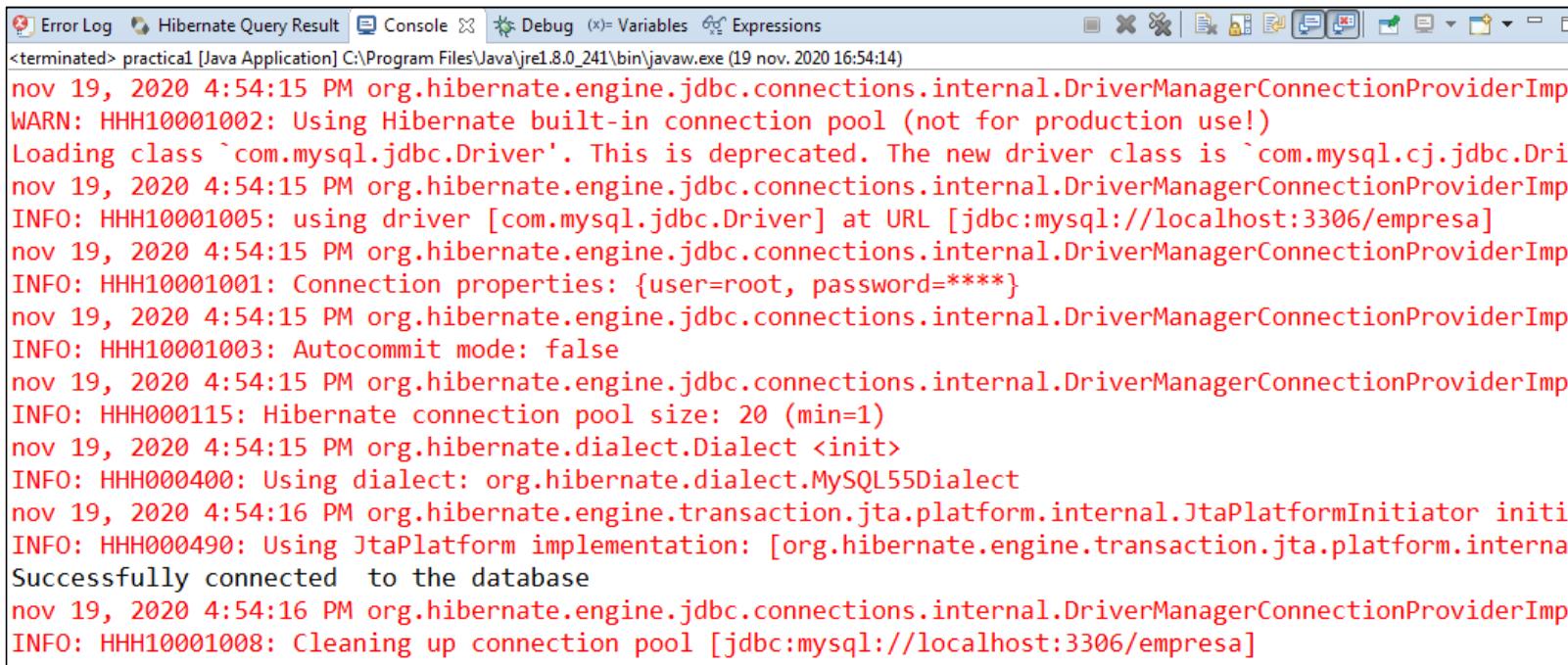
**Paso 12.** En el método main de nuestro proyecto probamos el siguiente código:



```
practica3.java ✘
1 package practica3;
2
3 public class practica3 {
4
5     public static void main(String[] args) {
6
7         BaseDatos db = new BaseDatos();
8         db.iniciar();
9         db.salir();
10    }
11 }
12 }
```

## 2. HIBERNATE DE FORMA MANUAL

**Paso 13.** Ejecutamos el programa (shortcut: **Ctrl + F11**) para comprobar si la session factory se carga exitosamente. Si vemos algo como esto en la vista **Console**, significa que la instalación de Hibernate se ha realizado correctamente (se carga correctamente sesión Factory)



The screenshot shows a Java application's console output. The title bar indicates it's a Java Application named 'practical1'. The log output is as follows:

```
<terminated> practical1 [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (19 nov. 2020 16:54:14)
nov 19, 2020 4:54:15 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImp
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Dri
nov 19, 2020 4:54:15 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImp
INFO: HHH10001005: using driver [com.mysql.jdbc.Driver] at URL [jdbc:mysql://localhost:3306/empresa]
nov 19, 2020 4:54:15 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImp
INFO: HHH10001001: Connection properties: {user=root, password=*****}
nov 19, 2020 4:54:15 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImp
INFO: HHH10001003: Autocommit mode: false
nov 19, 2020 4:54:15 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImp
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
nov 19, 2020 4:54:15 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL55Dialect
nov 19, 2020 4:54:16 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initi
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.interna
Successfully connected to the database
nov 19, 2020 4:54:16 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImp
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/empresa]
```

## 2. HIBERNATE DE FORMA MANUAL

**Paso 14.** Una vez SessionFactory de Hibernate está construido, se pueden construir los diferentes métodos para realizar las operaciones CRUD: Insertar, leer, actualizar y borrar. El denominador común de todos estos métodos es que todos:

- Abren una sessionFactory y empieza una transacción.
- Se hace la acción a realizar,
- Finalmente se hace el commit de la transaction y se cierra la sesión

```
protected void read() {  
    Session session = sessionFactory.openSession();  
    session.beginTransaction();  
  
    // code to get a book  
  
    session.getTransaction().commit();  
    session.close();  
}
```

# PRACTICA 1

Transforma el fichero controlador de la BBDD según el estándar Hibernate.

- Cambia el atributo Connection por el SessionFactory
- Crea las nuevas funciones setup() y exit()
- Reescribe los métodos: actualizar, borrar, insertar, etc

```
public class BaseDatos {  
  
    protected SessionFactory sessionFactory;  
  
    public BaseDatos() {  
        setup();  
    }  
    protected void setup() { // code to load Hibernate Session factory  
        final StandardServiceRegistry registry = new StandardServiceRegistryBuilder()  
            .configure() // configures settings from hibernate.cfg.xml  
            .build();  
    }  
}
```

```
protected void actualizar(Empleado emp) {  
    // code to modify a employee  
    Session session = sessionFactory.openSession();  
    session.beginTransaction();  
    session.update(emp);  
    session.getTransaction().commit();  
    session.close();  
}  
  
public void borrar(Empleado emp) {  
    // code to remove a employee  
    Session session = sessionFactory.openSession();  
    session.beginTransaction();  
    session.delete(emp);  
    session.getTransaction().commit();  
    session.close();  
}
```

# 3. HIBERNATE CON EL PLUGIN JBOSS

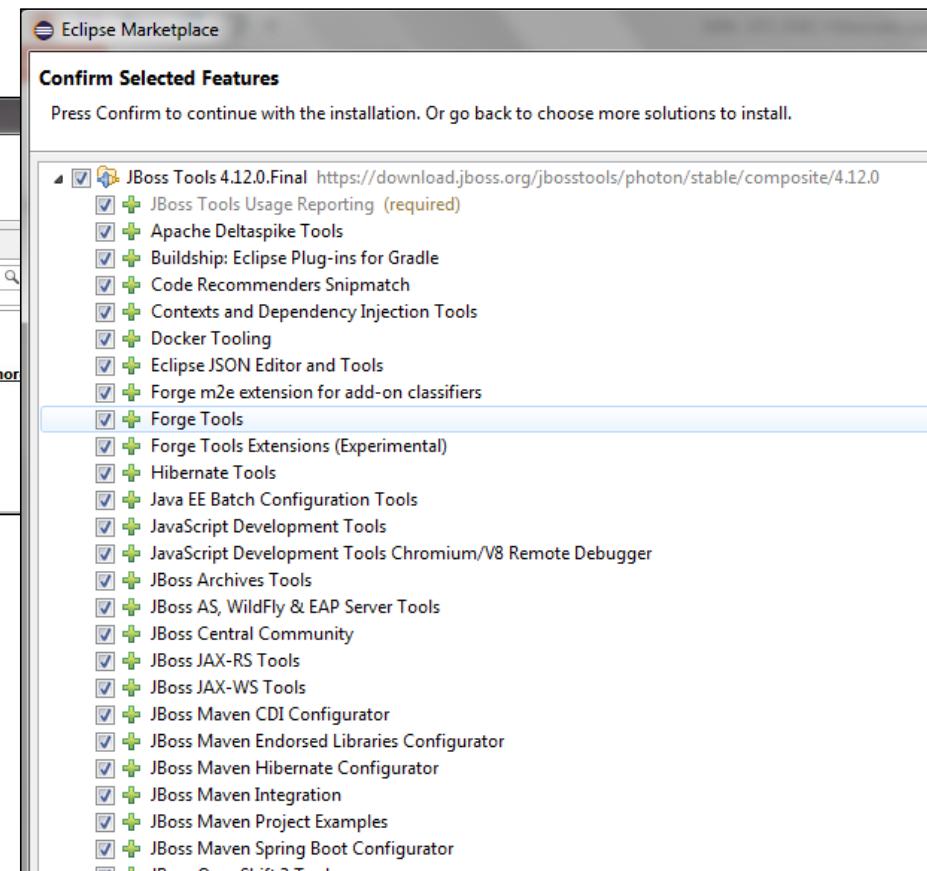
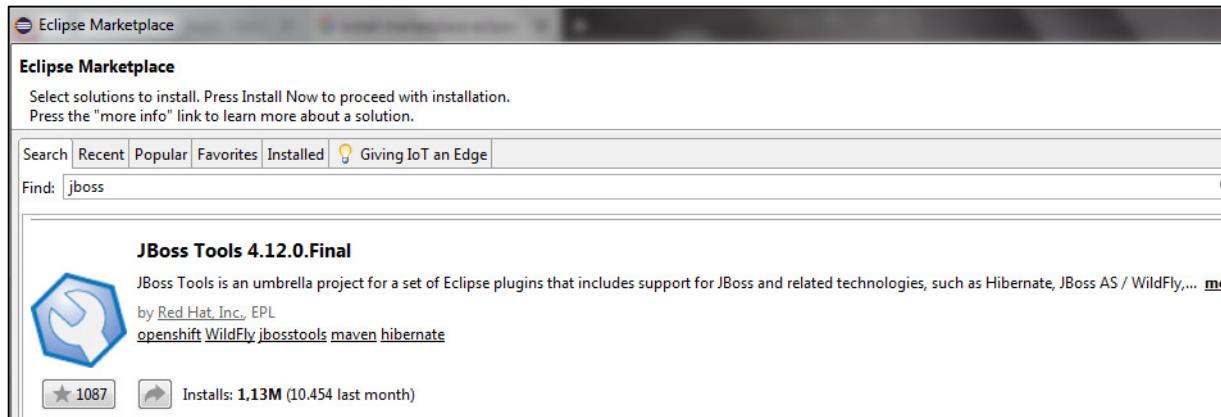
**Paso 0.** Script de la base de datos “empresa” en mysql

```
create table departamentos (
cod_dept INT PRIMARY KEY auto_increment,
nombre VARCHAR(20),
direccion VARCHAR(10),
objetivos int
);
```

```
create table empleados(
id int primary key auto_increment,
nombre VARCHAR(20),
apellidos VARCHAR(25),
salario double,
cod_dept int
);
alter table empleados add foreign key (cod_dept)
references departamentos(cod_dept);
```

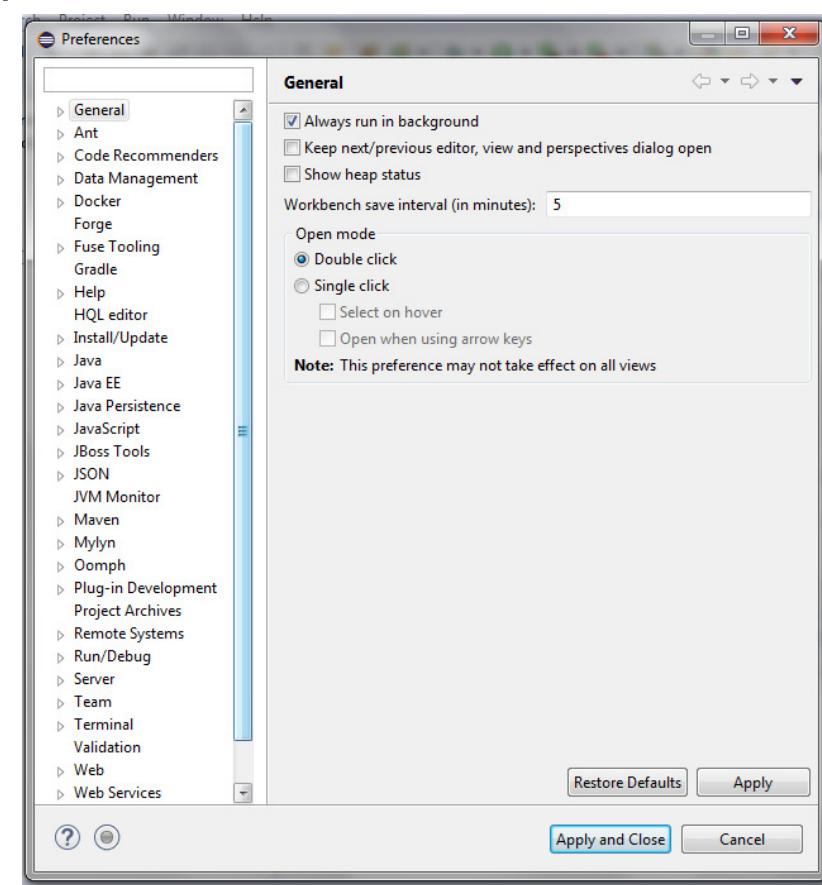
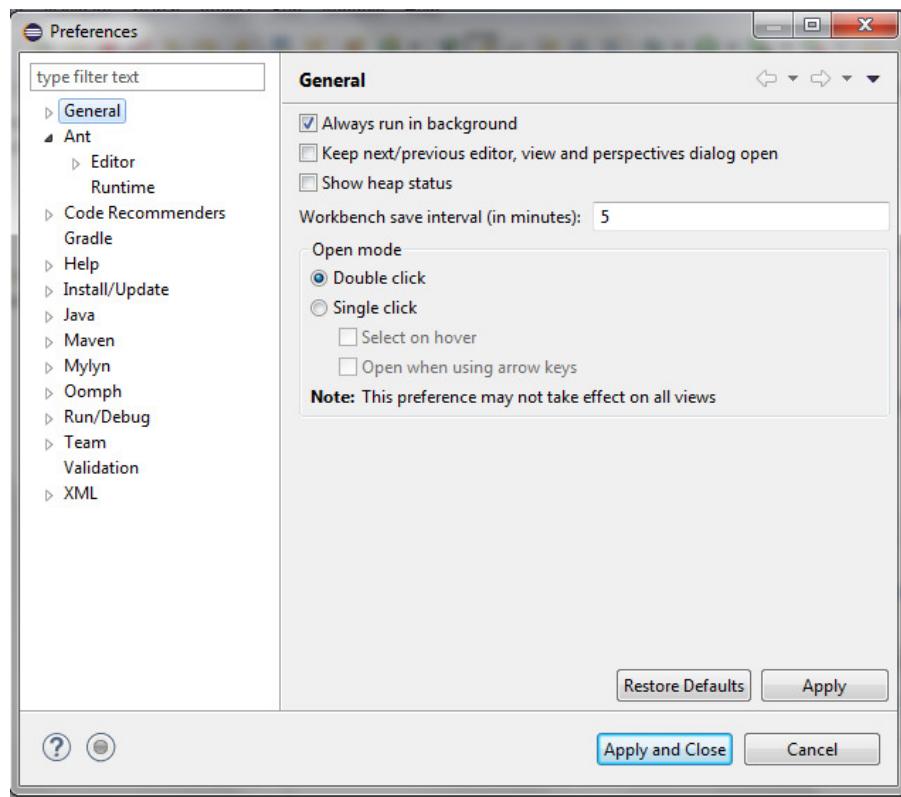
# 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 1.** Instalación del plugin para Eclipse “JBoss Tools”. Accedemos al menú **Help/Eclipse MarketPlace**, buscamos el plugin **JBoss Tools** y lo instalamos (Requerirá reiniciar Eclipse)



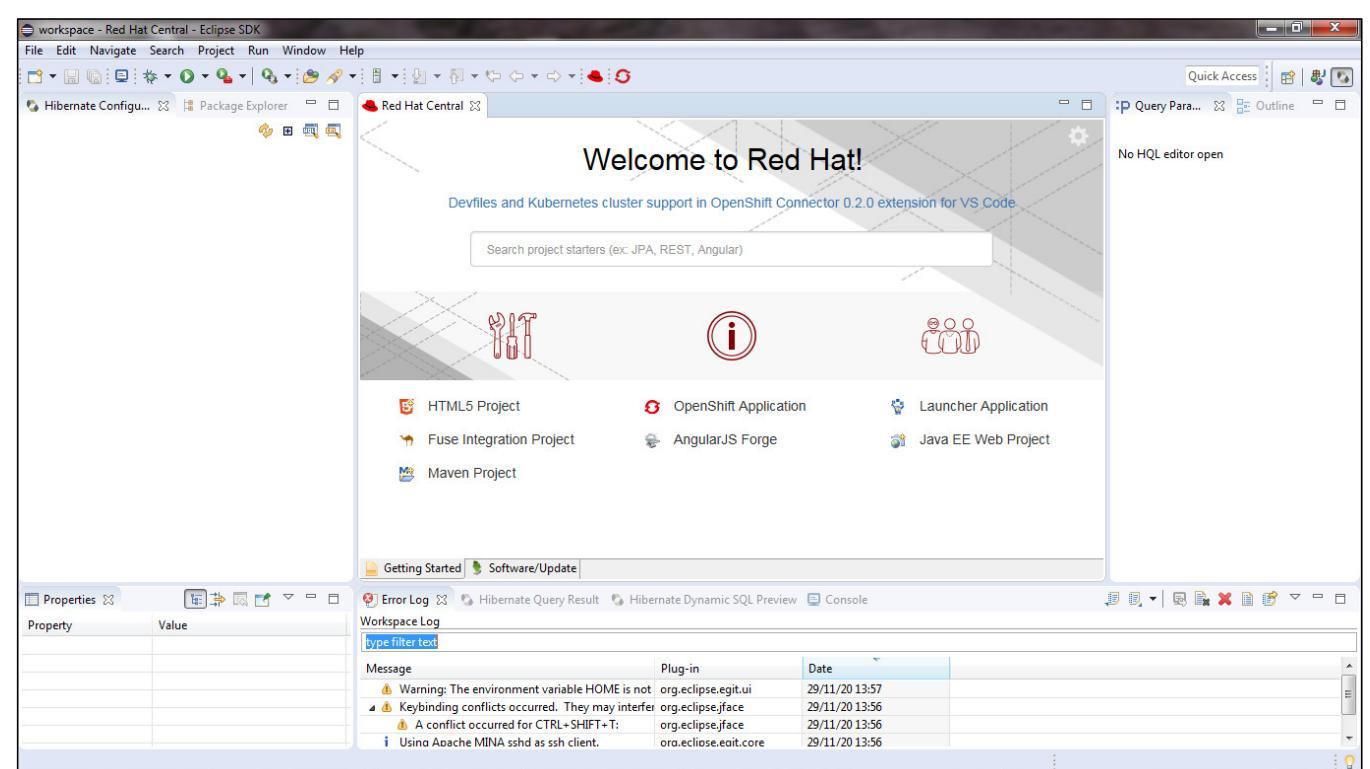
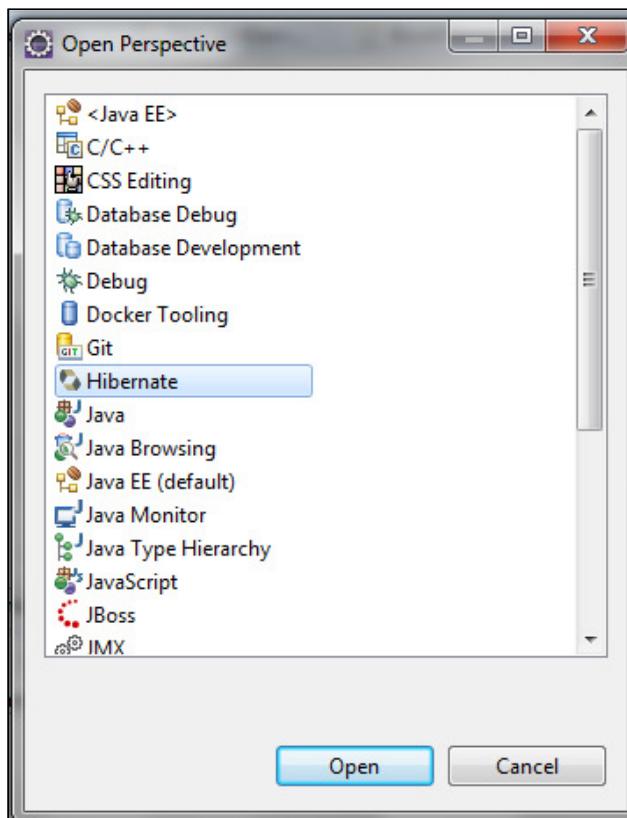
# 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 2.** Podemos comprobar que se ha instalado correctamente el plugin mirando que las opciones de preferencias se han duplicado:



# 3. HIBERNATE CON EL PLUGIN JBOSS

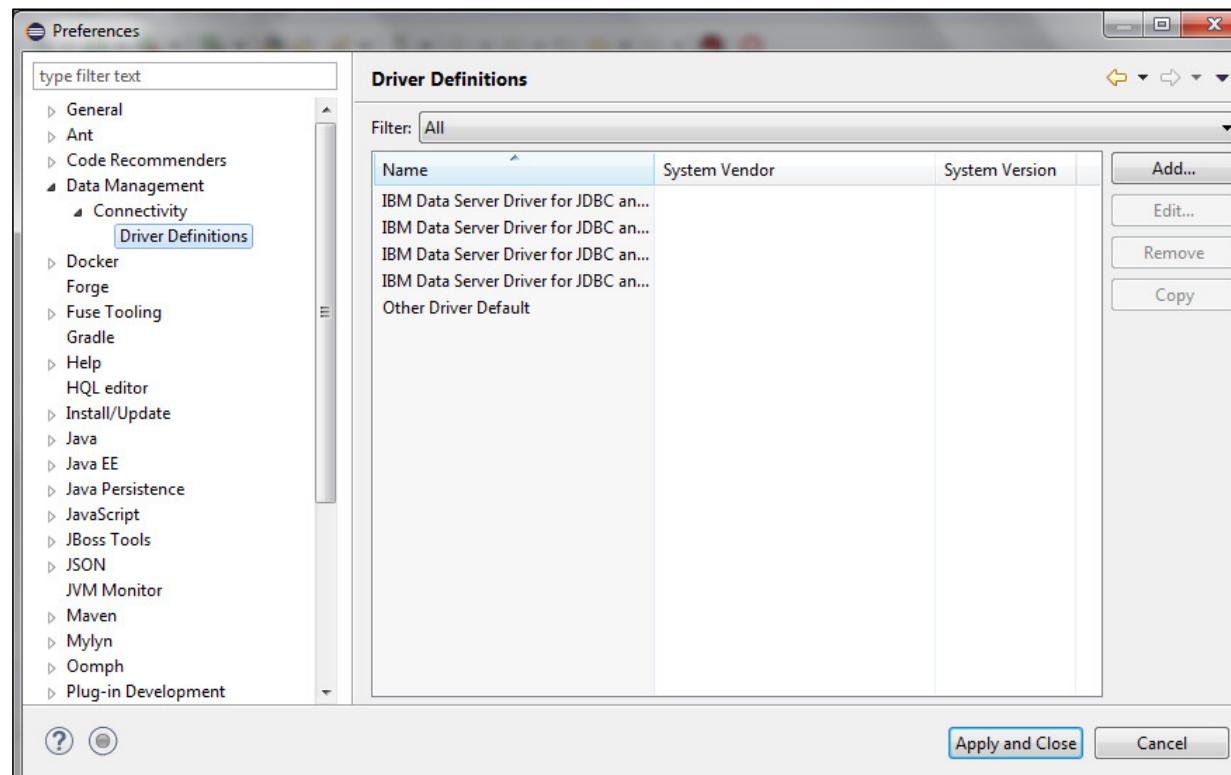
**Paso 3.** En **Window/perspective/other**, entre las opciones que aparecen debe estar la perspectiva **Hibernate**.



# 5. HIBERNATE CON EL PLUGIN JBOSS

## Configuración driver Mysql en Eclipse

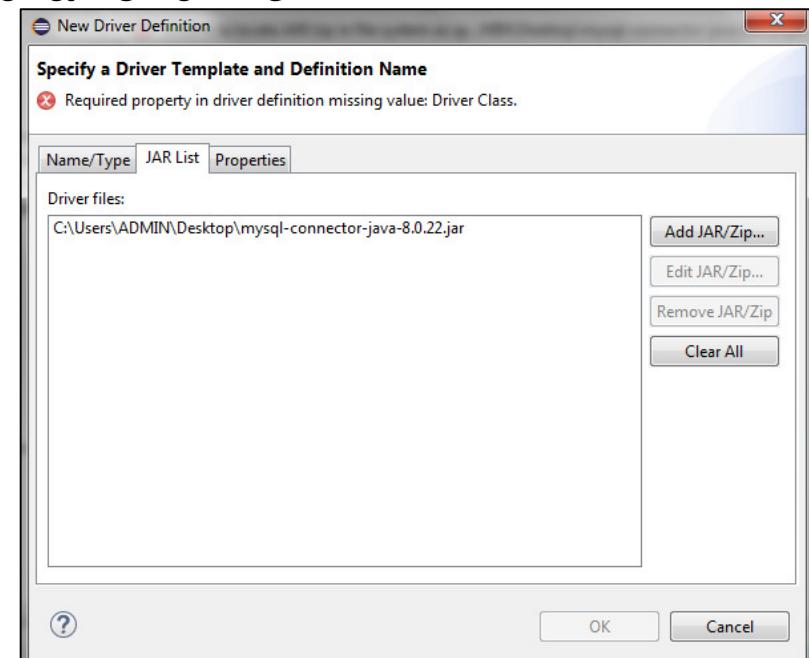
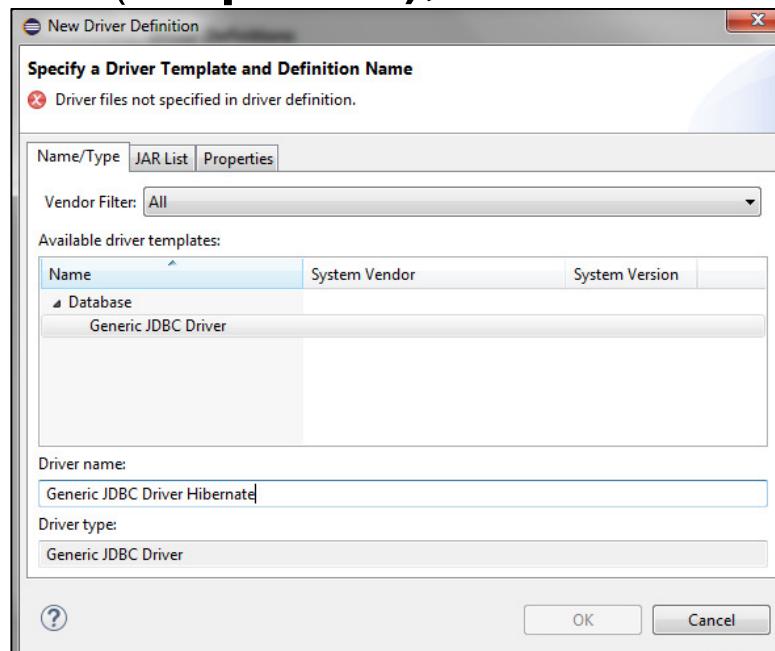
**Paso 4.** Vamos a **Window/preferences** y en el menú de la izquierda escogemos **DataManagement/Connectivity/Driver definitions.**



### 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 5.** Añadiremos el nuevo conector haciendo click en “Add”. Nos aparecerá un cuadro de diálogo rotulado “**New Driver Definition**”, que dispone de 3 pestañas:

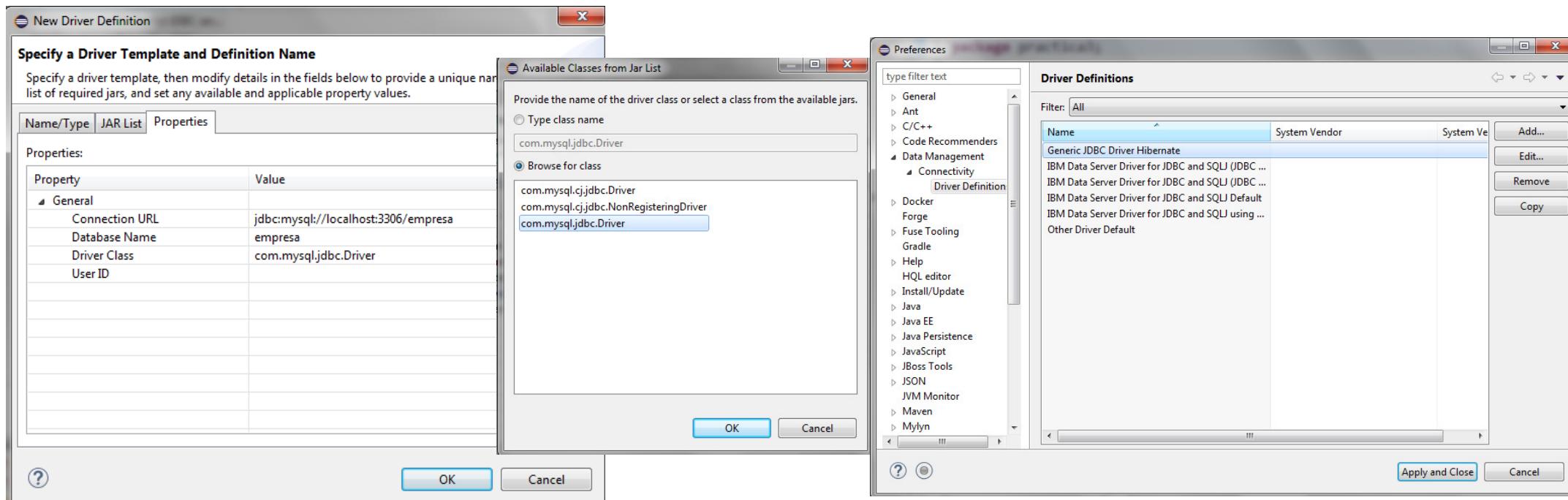
- La 1º (**Name/Type**) nos pide un nombre para el driver, el Vendor y el tipo conector.
- En la 2º (**JAR list**) se debe de indicar la ruta al paquete .jar del conector.
- Y en la 3º (**Properties**), se debe de configurar el driver.



# 3. HIBERNATE CON EL PLUGIN JBOSS

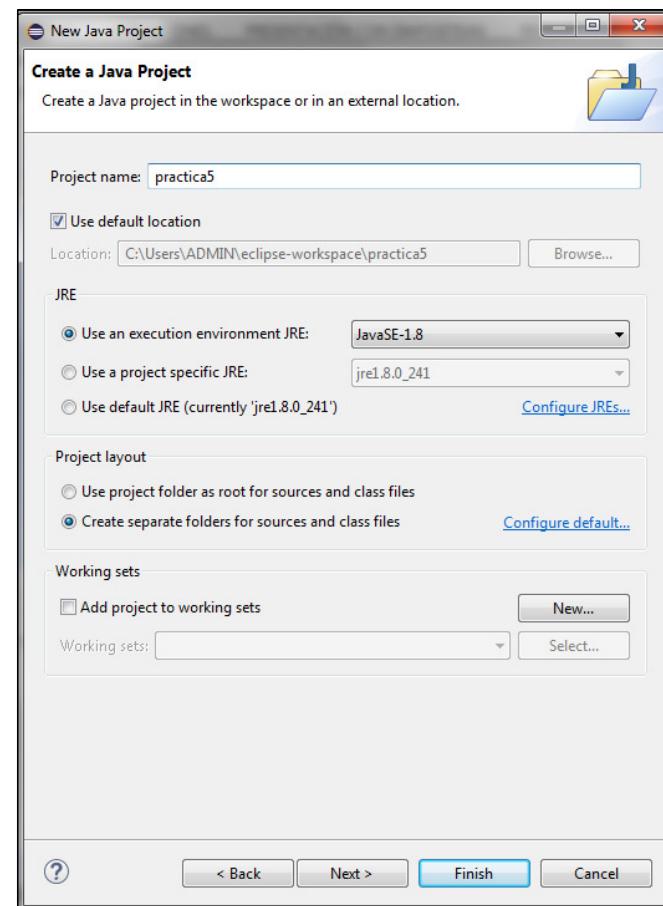
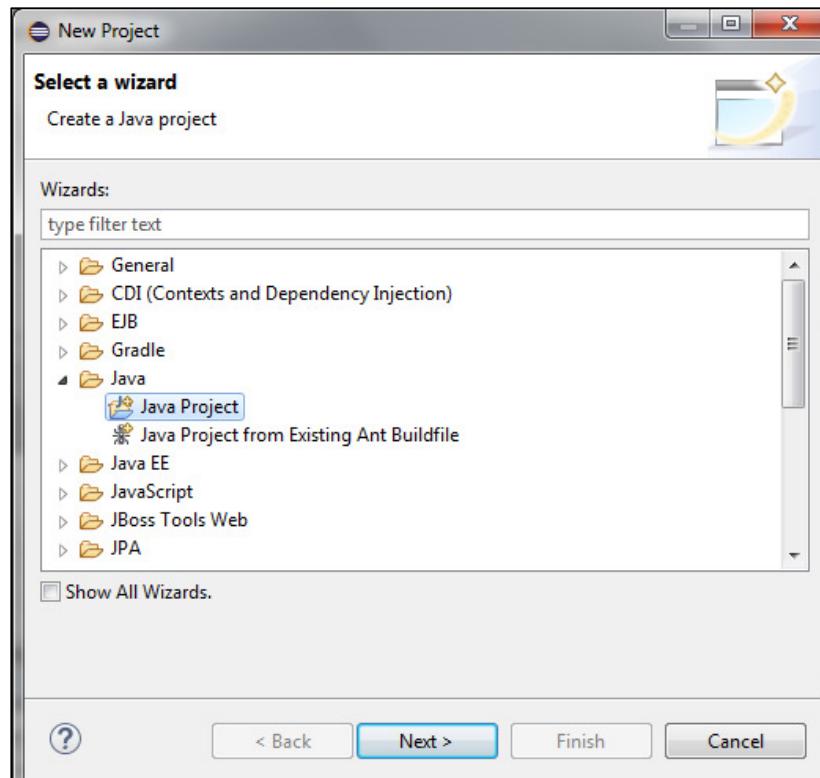
**Paso 6.** En la pestaña **properties** se realiza la configuración:

- Connection URL: Indicamos el localizador del recurso.
- Database Name: Indicamos el nombre de la bbdd.
- Driver Class: La clase del paquete que utilizaremos como driver



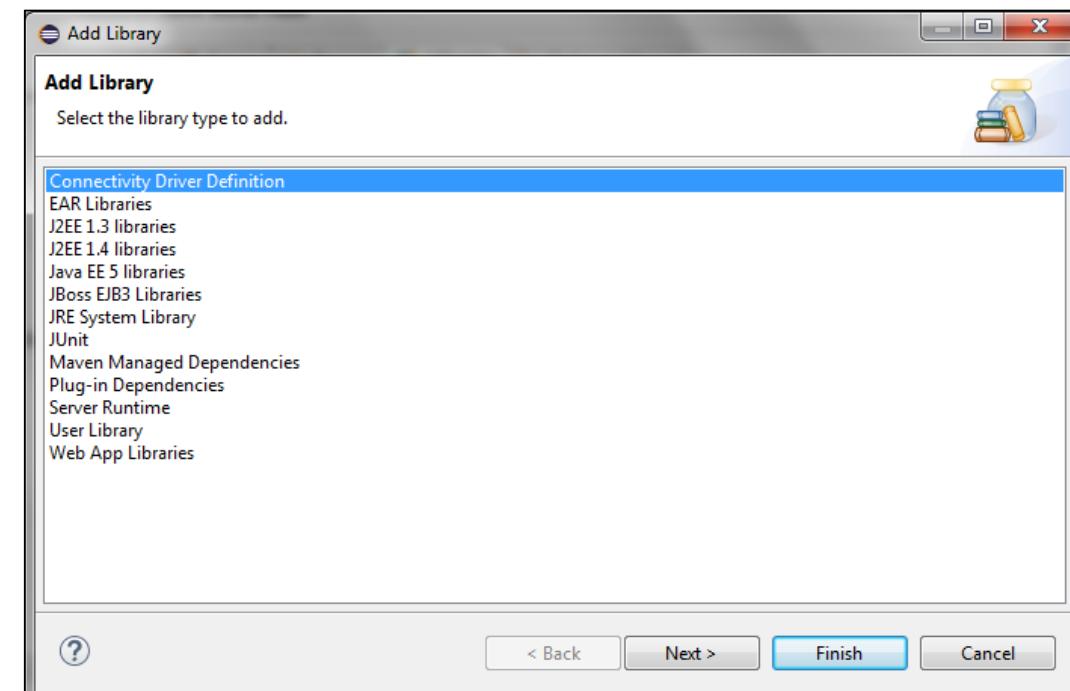
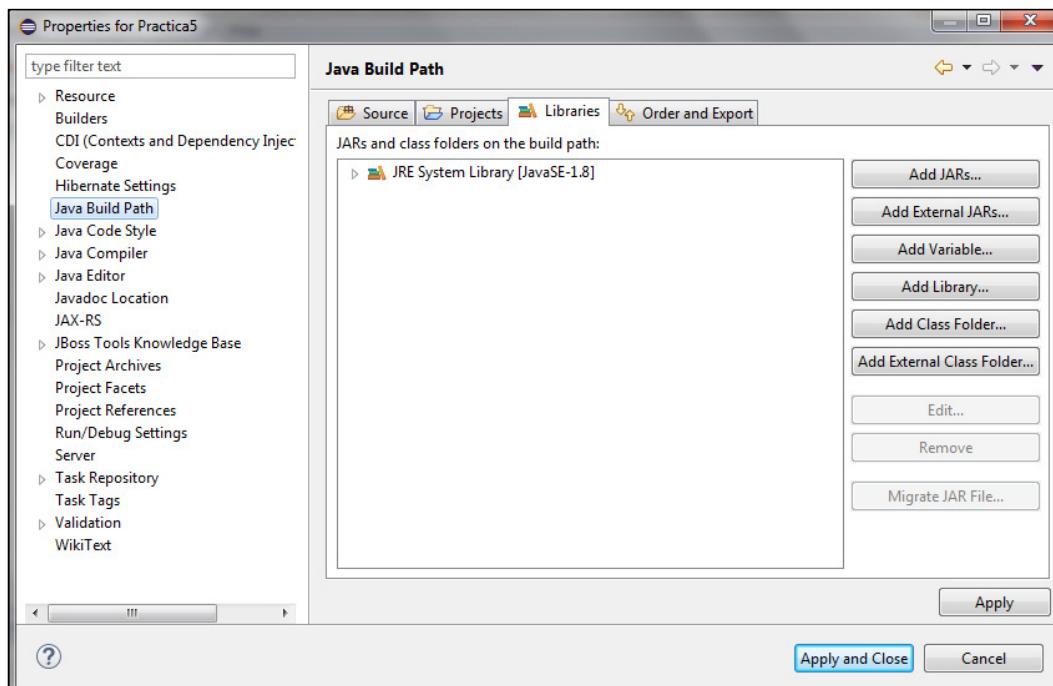
# 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 7.** Creamos un nuevo proyecto Java en Eclipse (por ejemplo practica5):



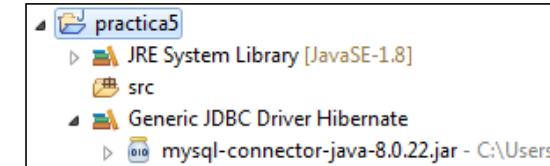
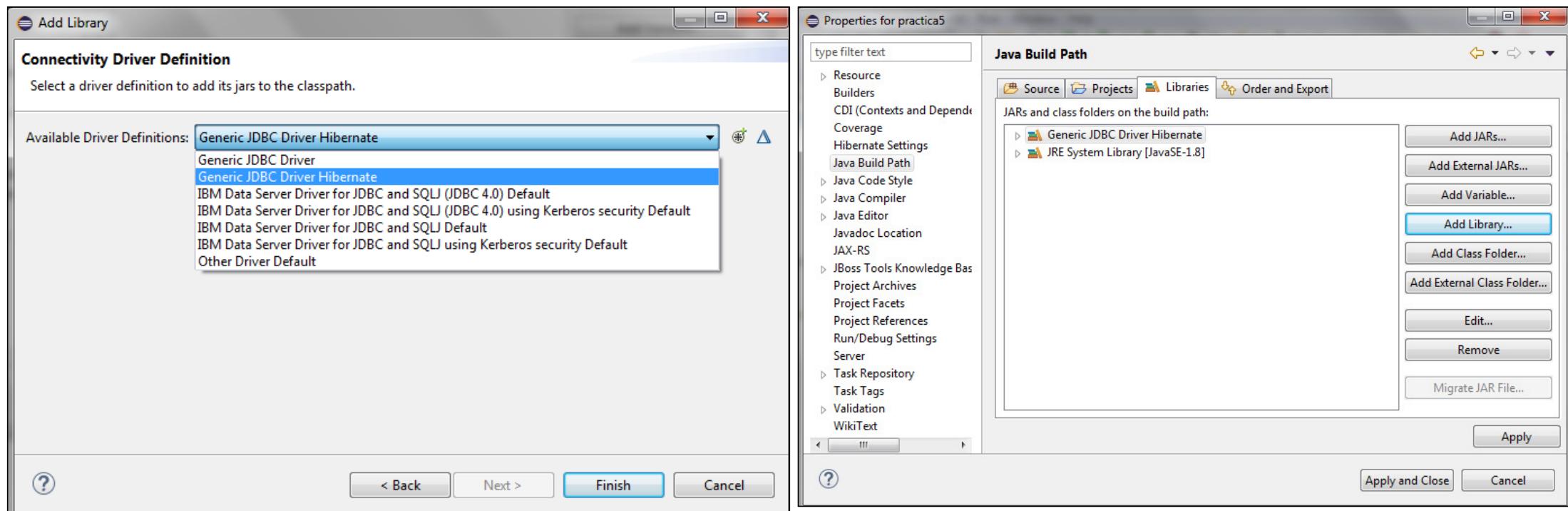
# 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 8.** Vamos a **Properties/Java Build Path** y hacemos click en **Add Libraries**. Se visualiza una ventana, elegimos **Connectivity Driver Definition**.



# 3. HIBERNATE CON EL PLUGIN JBOSS

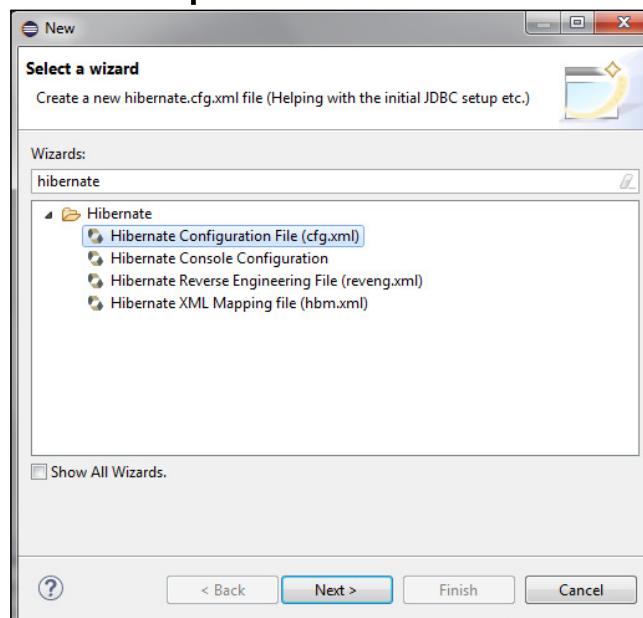
**Paso 9.** Tras el *Next* correspondiente elegimos el driver recién definido.



# 3. HIBERNATE CON EL PLUGIN JBOSS

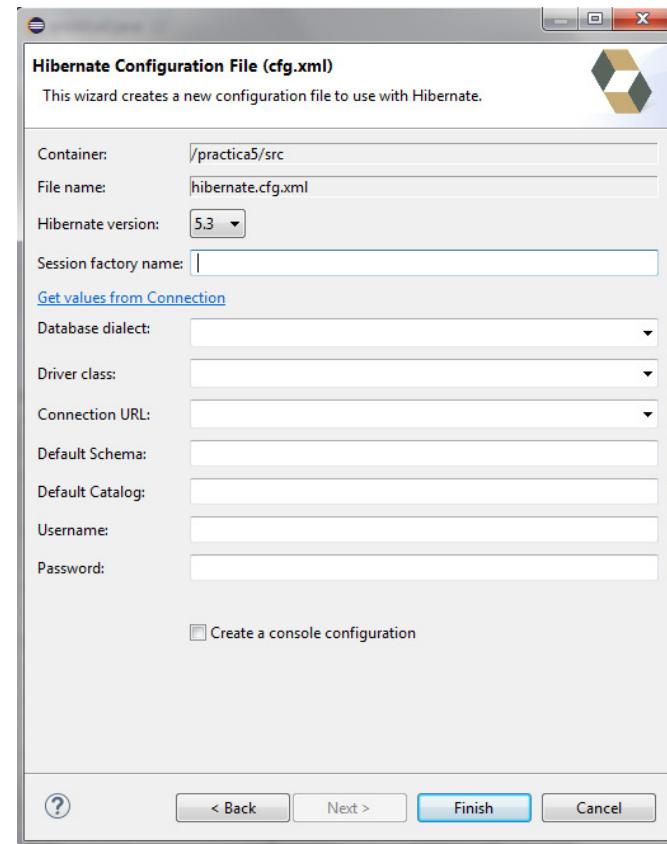
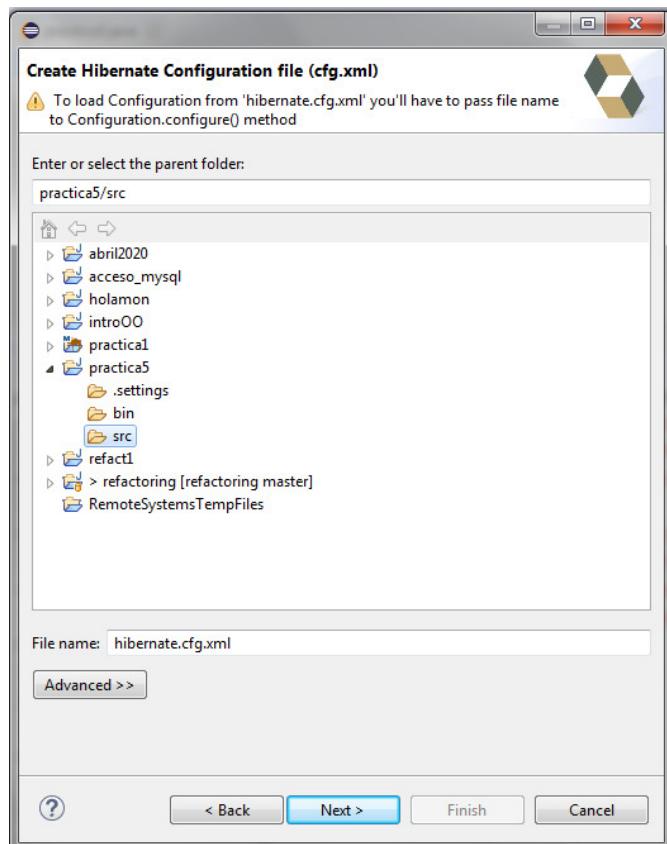
## Configuración Hibernate

**Paso 10.** Una vez tenemos la librería MySql en nuestro proyecto hemos de crear el fichero de configuración de Hibernate llamado hibernate.cfg.xml. En nuestro proyecto, hacemos click botón derecho y seleccionamos **New/Other/Hibernate/Hibernate Configuration File (cfg.xml)**. Este fichero es un XML que contiene todo lo necesario para realizar la conexión a la base de datos.



# 3. HIBERNATE CON EL PLUGIN JBOSS

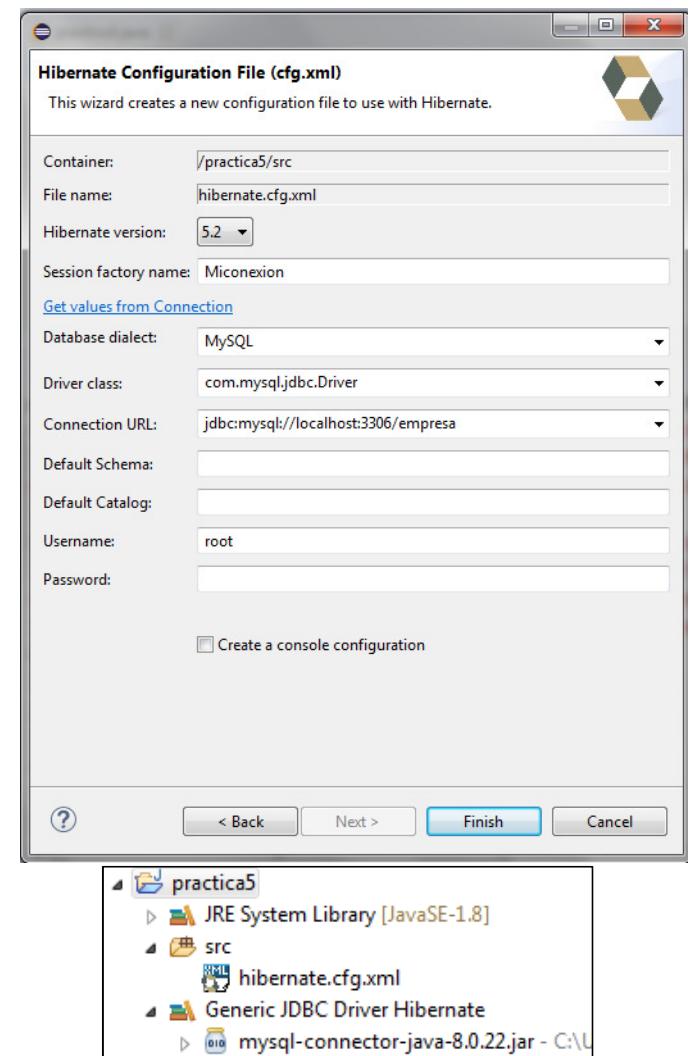
**Paso 11.** A continuación, nos pide dónde crear el fichero. Lo haremos en la carpeta src del proyecto. Otro *Next* y pasamos a configurar la conexión.



# 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 12.** Los campos a rellenar son:

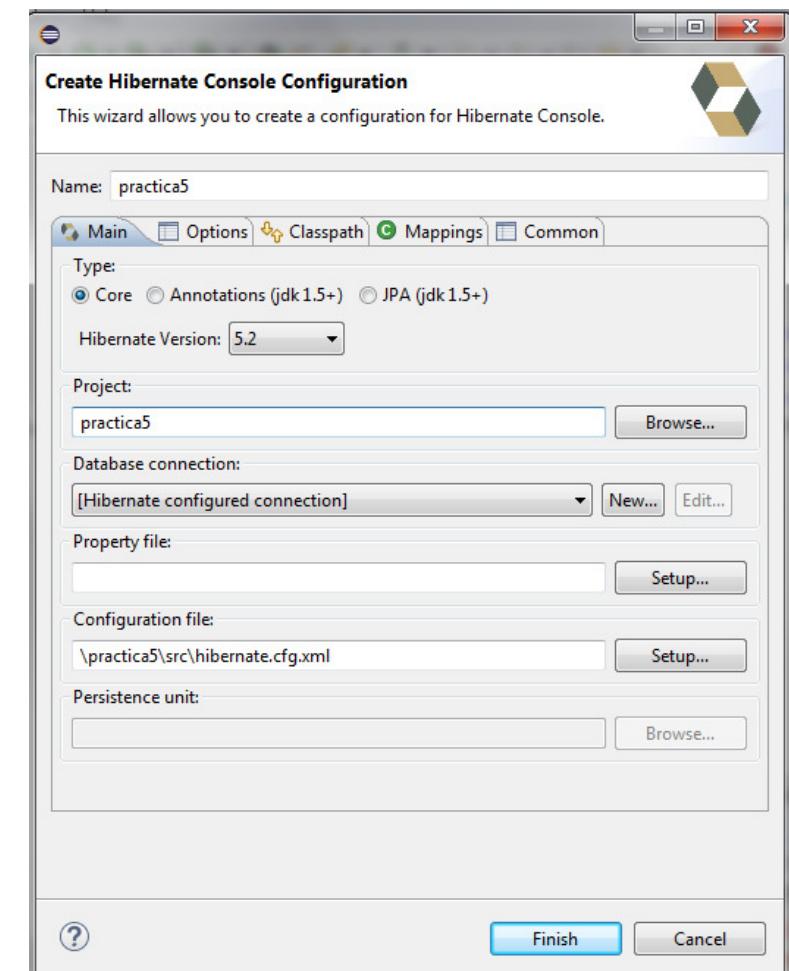
- **Session factory name** → nombre de la conexión. Pondremos “Miconexion”.
- **Database dialect** → elegimos tipo de comunicación JDBC. En concreto, “MySQL”
- **Driver Class** → clase JDBC para la conexión → **com.mysql.jdbc.Driver**
- **Conection URL** → ruta de conexión a la bbdd indicada cuando definíamos el driver
- **Username** → Usuario que se conectará a MySql. Debe de tener permisos
- **Password** → Contraseña correspondiente.



# 3. HIBERNATE CON EL PLUGIN JBOSS

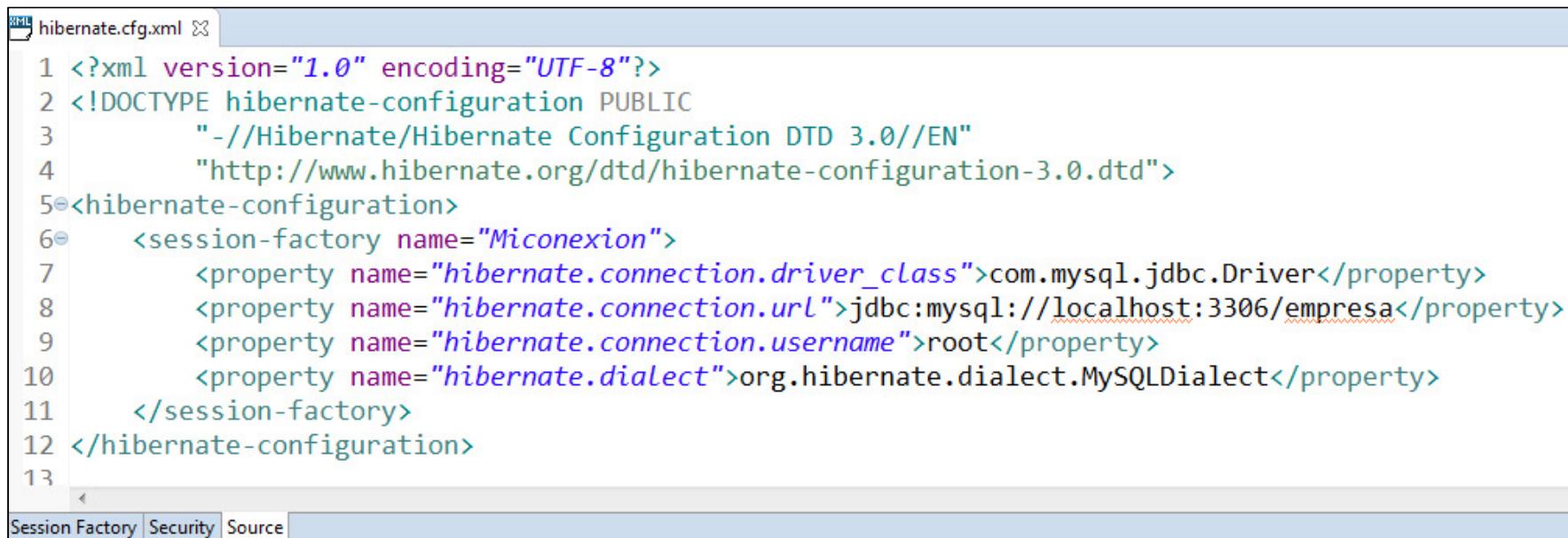
**Paso 13.** Una vez creado el **hibernate.cfg.xml**, hemos de crear el fichero **XML Hibernate Console Configuration**.

- Seleccionamos nuestro proyecto, botón derecho: **New /Other / Hibernate / Hibernate Console Configuration**.
- En el campo Name de la nueva ventana podemos dejar el nombre por defecto “practica5” para nuestra configuración de consola Hibernate.
- Finalmente hacemos click en Finish



# 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 14.** Aparece el editor de configuración de Hibernate. Desde la pestaña **Source** se puede editar el fichero XML generado:

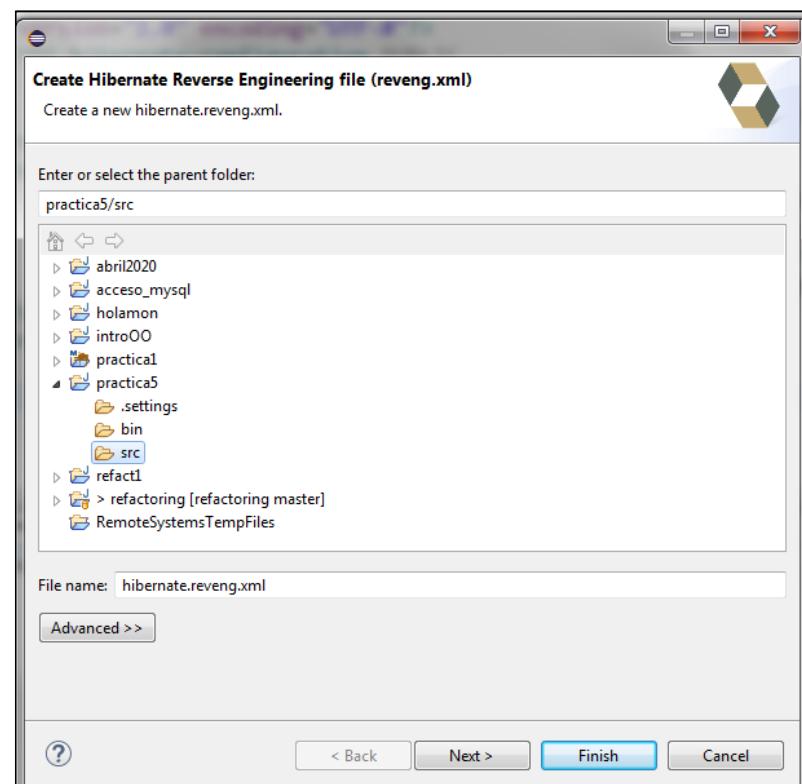


The screenshot shows a software interface for editing Hibernate configuration files. The title bar says "hibernate.cfg.xml". The main area contains the XML code for a session factory named "Miconexion". The code includes properties for connection driver, URL, username, and dialect. The bottom navigation bar has tabs for "Session Factory", "Security", and "Source", with "Source" being the active tab.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-configuration PUBLIC
3         "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4         "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
5<hibernate-configuration>
6    <session-factory name="Miconexion">
7        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
8        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/empresa</property>
9        <property name="hibernate.connection.username">root</property>
10       <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
11    </session-factory>
12 </hibernate-configuration>
13
```

### 3. HIBERNATE CON EL PLUGIN JBOSS

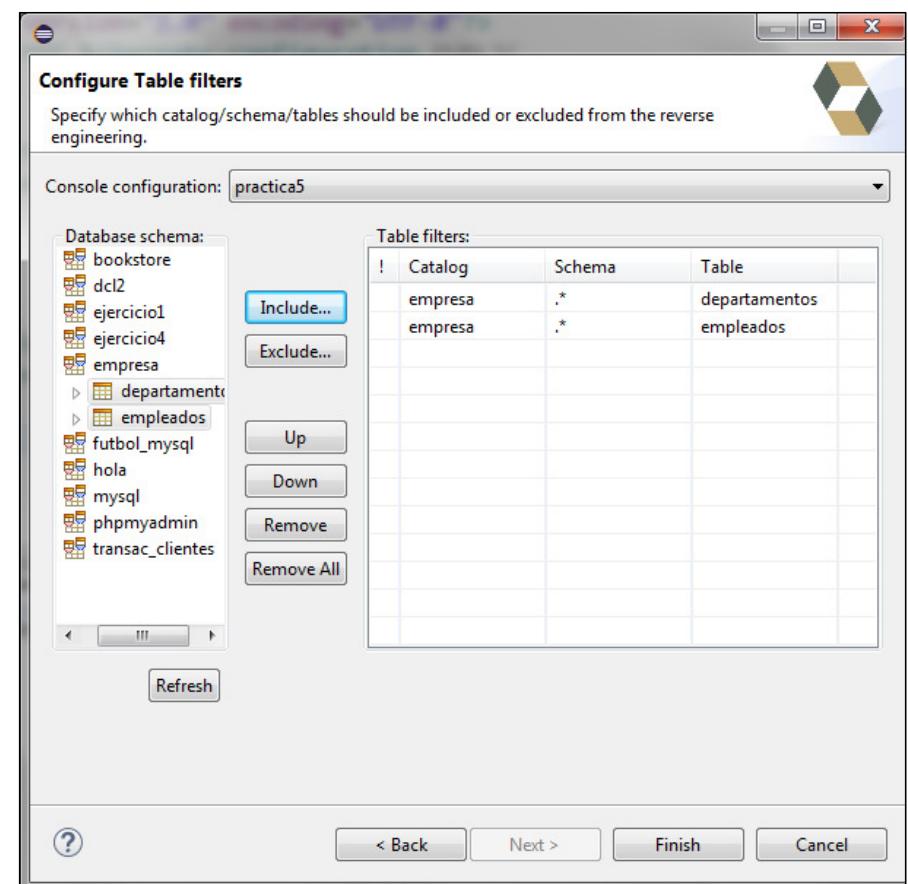
**Paso 15.** Por último, para acabar de configurar Hibernate, hemos de crear el fichero **XML Hibernate Reverse Engineering (reveng.xml)**. Vamos a **New / Other / Hibernate / Hibernate Reverse Engineering File**. Primero nos pide la ubicación, se debe indicar la carpeta **src** del proyecto.



### 3. HIBERNATE CON EL PLUGIN JBOSS

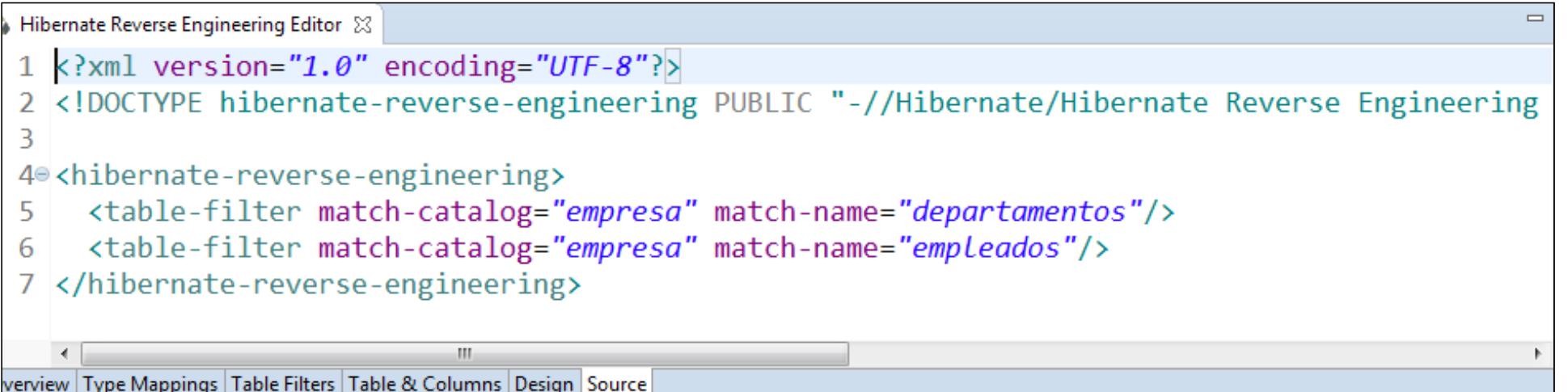
**Paso 16.** Al hacer click en *Next*, se visualiza una ventana desde donde indicaremos las tablas que queremos mapear.

- Primero elegimos nuestra **Console Configuration** creada en el paso anterior
- A continuación, pulsamos en *Refresh* para que muestre la base de datos “empresa” y sus tablas.
- Se seleccionan las tablas “departamentos” y “empleados” y pulsamos el botón *Include*.
- Finalmente hacemos click en *Finish*



# 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 17.** Se abrirá el editor de Hibernate Reverse Engineering y en la pestaña **Source** podemos visualizar el XML generado



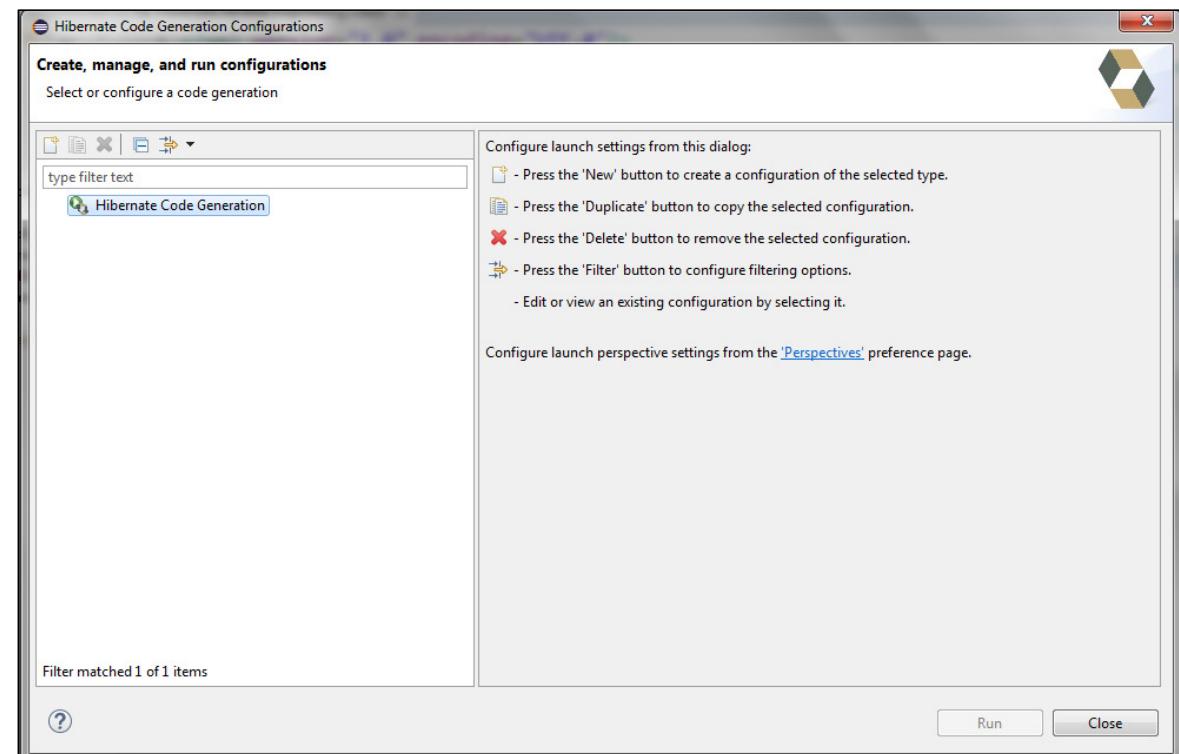
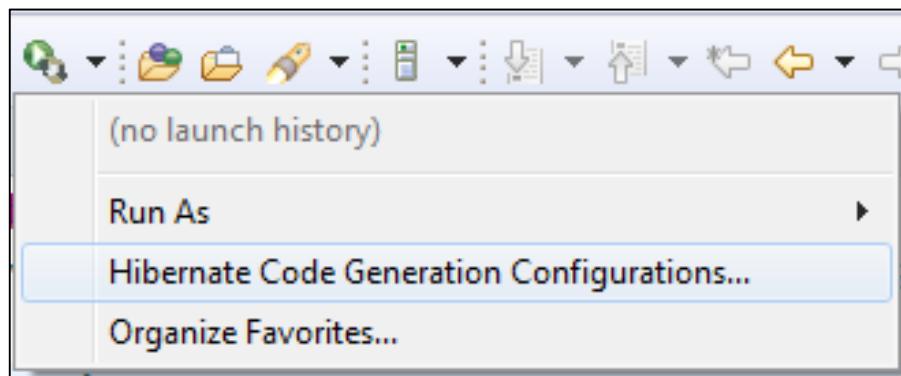
The screenshot shows a window titled "Hibernate Reverse Engineering Editor". The main area displays the following XML code:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-reverse-engineering PUBLIC "-//Hibernate/Hibernate Reverse Engineering
3
4<hibernate-reverse-engineering>
5   <table-filter match-catalog="empresa" match-name="departamentos"/>
6   <table-filter match-catalog="empresa" match-name="empleados"/>
7 </hibernate-reverse-engineering>
```

Below the code, there is a navigation bar with tabs: Overview, Type Mappings, Table Filters, Table & Columns, Design, and Source. The "Source" tab is currently selected.

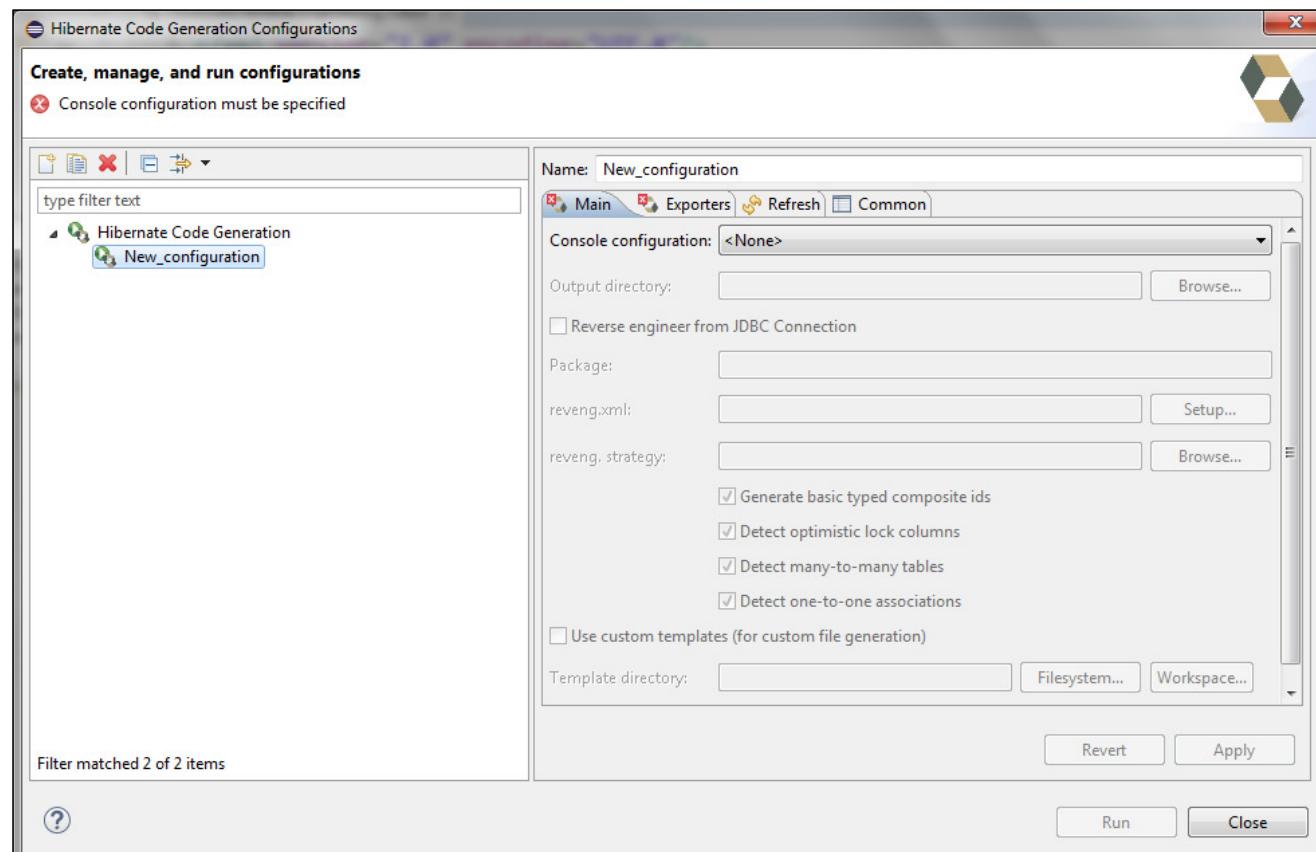
### 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 18.** Generamos las clases de nuestra base de datos “empresa”. Para ello pulsamos en la flechita situada a la derecha del botón **Run As** y seleccionamos **Hibernate Code Generation Configurations**.



### 3. HIBERNATE CON EL PLUGIN JBOSS

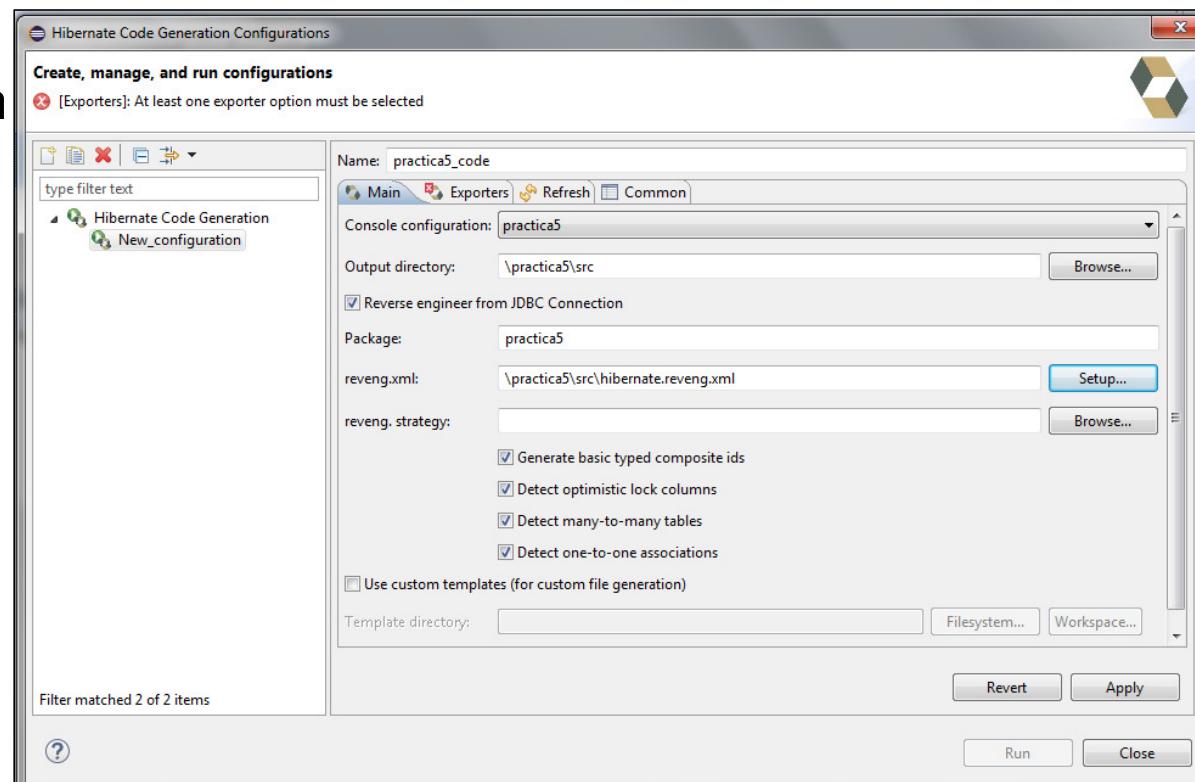
**Paso 19.** En la nueva ventana hacemos doble clic en la opción **Hibernate Code Generation** del marco de la izquierda. Aparecerán varias pestañas:



# 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 20.** En la pestaña **Main** configuramos:

- **Name:** Nombre para la configuración
- **Console configuration:** Indicamos proyecto5
- **Output directory:** debe ser la carpeta **src**.
- **Package:** el nombre del paquete donde se crearán las clases, por ejemplo “practica5”
- **reveng.xml:** localizamos el fichero reveng.xml creado anteriormente.

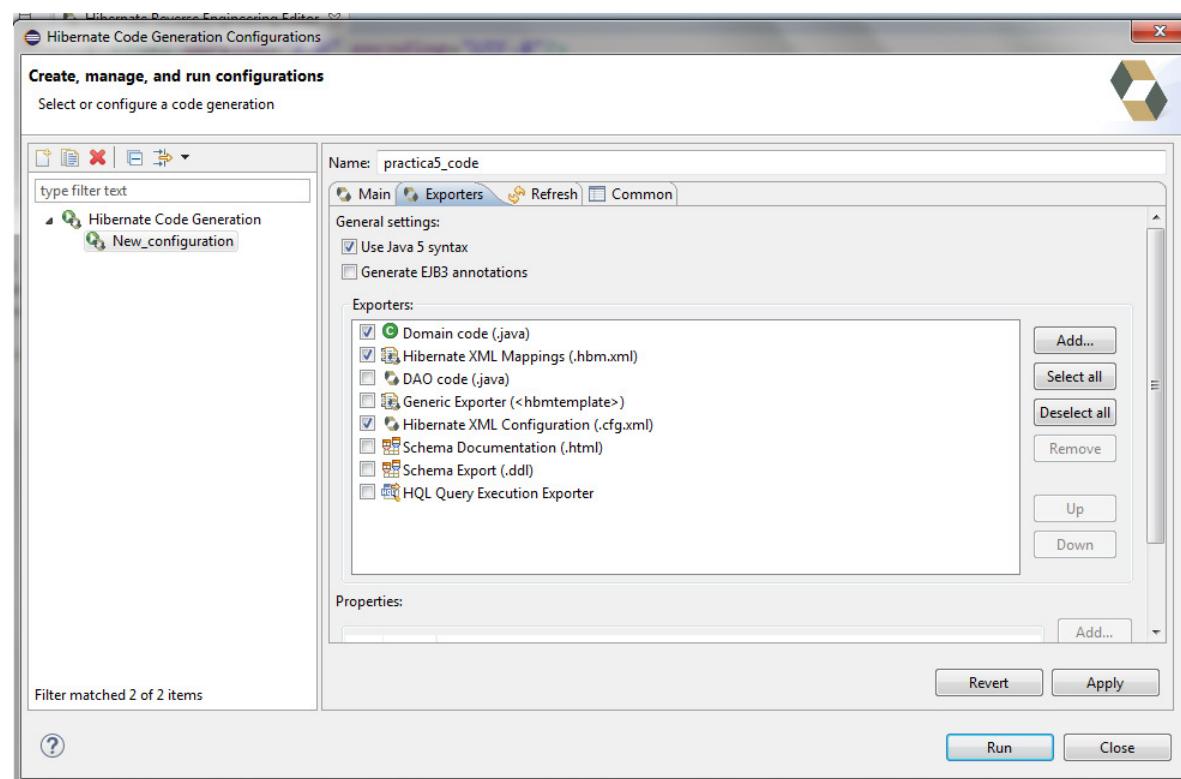


# 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 21.** En la pestaña **Exporters** se indica los fichero a generar. Se marcarán:

- **Use Java 5 syntax**
- **Domain code**
- **Hibernate XML Mappings**
- **Hibernate XML Configuration**

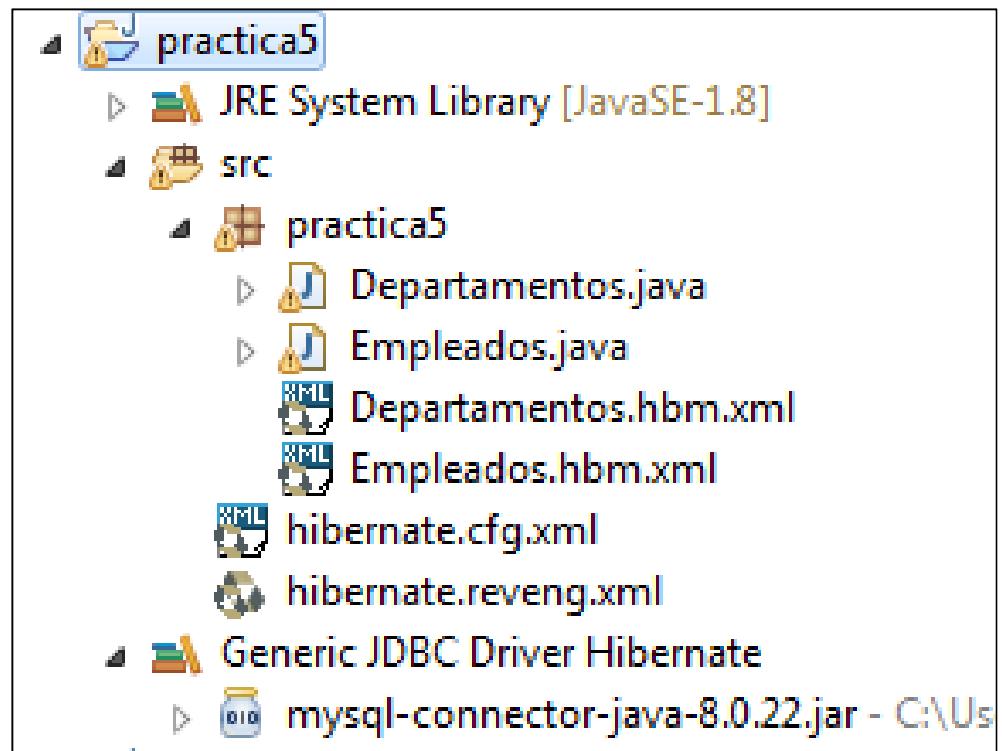
Clicamos en *Apply* y  
posteriormente en *Run*.



### 3. HIBERNATE CON EL PLUGIN JBOSS

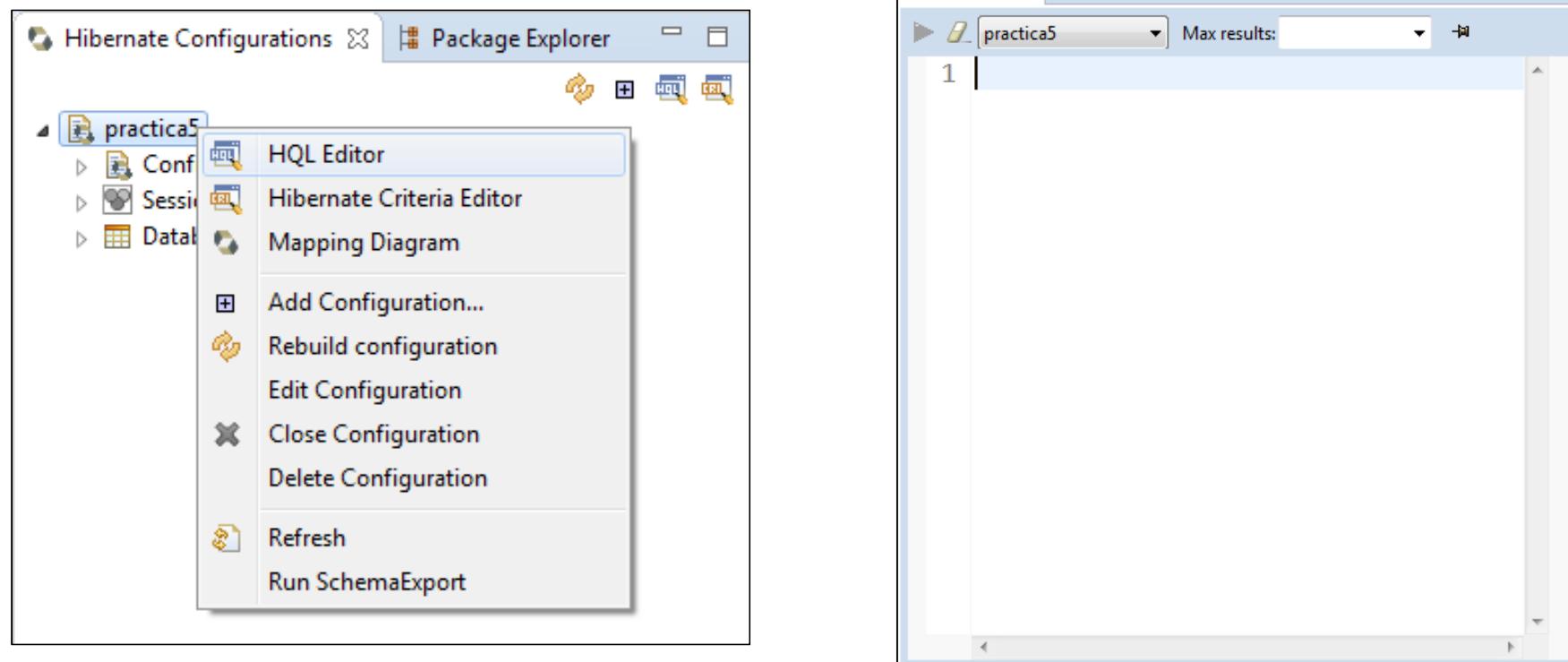
**Paso 22.** Al ejecutarse nos genera un paquete llamado primero con las clases Java de las tablas emple y depart que contienen los métodos getters y setters de cada campo de la tabla. También contendrá los XML con información del mapeo de cada tabla.

Conviene detenerse un poco y analizar cómo ha mapeado Hibernate cada tabla. Sobre todo es interesante la implementación de las claves ajenas.



### 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 23.** Para realizar la primera consulta en HQL (Hibernate Query language), debemos de abrir la vista Hibernate Configurations. En la nueva pestaña, se tiene que pulsar sobre nuestro proyecto con el botón derecho y seleccionar HQL Editor



### 3. HIBERNATE CON EL PLUGIN JBOSS

**Paso 24.** En el editor HQL podemos realizar consultas. Por ejemplo, si se escribe "from Empleados" y pulsamos la flechita verde nos debe aparecer en la pestaña **Hibernate Query Result** el resultado de la consulta.

Desde este entorno también se pueden realizar consultas SQL aunque hay ciertas restricciones. Por ejemplo, no se puede utilizar \* en el SELECT

The image displays two side-by-side screenshots of the JBoss Tools HQL editor interface. Both windows have a title bar labeled 'HQL \*practica5' and a toolbar with a green play button, a dropdown menu, and a 'Max results:' dropdown.

**Left Window (Employee Query):**

- Query: `1 from Empleados`
- Result tab: 'Hibernate Query Result'
- Results:
  - practica5.Empleados
  - practica5.Empleados@6b19c059
  - practica5.Empleados@6d3cb4d9
  - practica5.Empleados@36c3f997

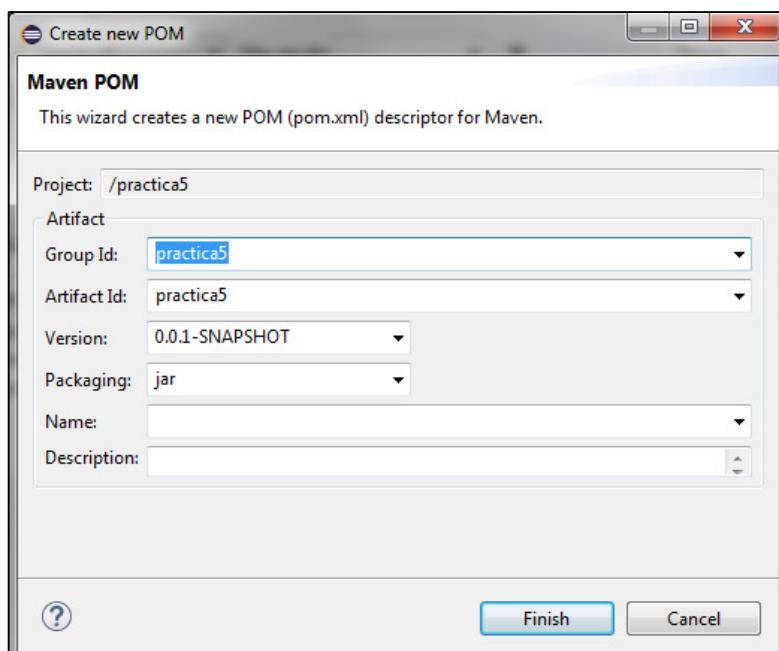
**Right Window (Department Query):**

- Query: `1 from Departamentos`
- Result tab: 'Hibernate Query Result'
- Results:
  - practica5.Departamentos
  - practica5.Departamentos@2fb7db85
  - practica5.Departamentos@84cd98d
  - practica5.Departamentos@33f69610

# 3. HIBERNATE CON EL PLUGIN JBOSS

## Librerías de Hibernate

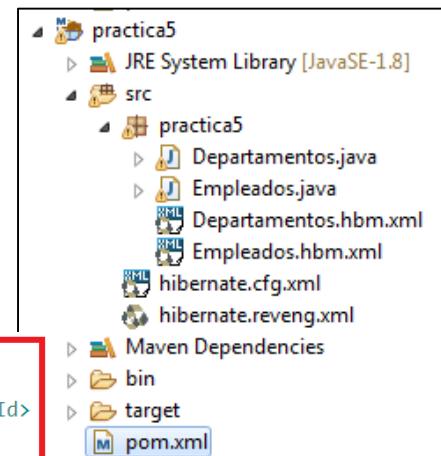
**Paso 25.** Tenemos dos formas de agregar las librerías Hibernate a nuestro proyecto. Optaremos por la vía Maven. Convertimos nuestro proyecto a Maven y configuraremos el fichero pom.xml



```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.2.6.Final</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.22</version>
    </dependency>
</dependencies>

</project>
```

The code above is the generated pom.xml content. A red box highlights the section from <dependencies> to </dependencies>, which contains the dependencies for Hibernate and MySQL.



### 3. HIBERNATE CON EL PLUGIN JBOSS

---

**Paso 26.** Se debe de crear un **Singleton**: Es un patrón diseñado para restringir la creación de objetos pertenecientes a un clase. Su intención consiste en garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella (**es como el fichero de prácticas anteriores BaseDatos.java**):

- Nuestro Singleton será una clase de ayuda que accede a **SessionFactory** para obtener un objeto de sesión, hay una única **SessionFactory** para toda la aplicación.
- El nombre de la clase es *HibernateUtil.java* y se incluirá en el paquete de nuestro proyecto.
- Con esta clase podemos obtener la sesión actual desde cualquier parte de nuestro proyecto.

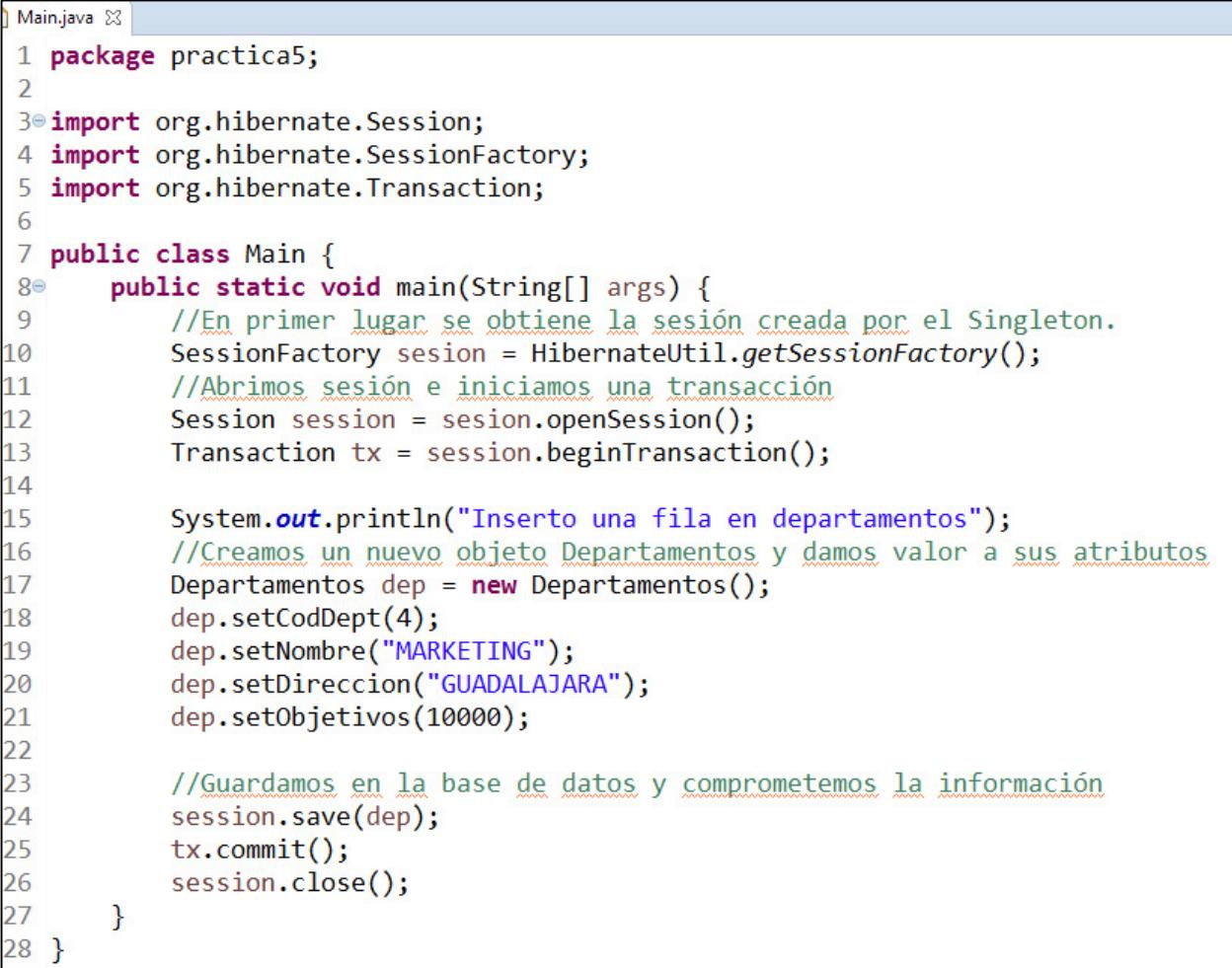
# 3. HIBERNATE CON EL PLUGIN JBOSS

```
HibernateUtil.java ✘
1 package practica5;
2
3 import org.hibernate.SessionFactory;
4 import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
5 import org.hibernate.cfg.Configuration;
6
7 public class HibernateUtil {
8     private static final SessionFactory sessionFactory = buildSessionFactory();
9
10    private static SessionFactory buildSessionFactory(){
11        try{
12            return new Configuration().configure().
13                buildSessionFactory(new StandardServiceRegistryBuilder().
14                    configure().build());
15        }
16        catch (Throwable ex) {
17            System.err.println("Initial SessionFactory creation failed."+ex);
18            throw new ExceptionInInitializerError(ex);
19        }
20    }
21    public static SessionFactory getSessionFactory(){
22        return sessionFactory;
23    }
24 }
```

### 3. PRACTICA 2

**Paso 27.** Para finalizar esta primera parte de instalación del plugin JBOSS y de validación, crea en nuestro proyecto una nueva clase que llamaremos *Main.java*.

Haremos un caso de uso que insertará una fila en la tabla *departamentos*



The screenshot shows a code editor window with the file 'Main.java' open. The code is written in Java and uses annotations to explain the purpose of each step. It imports the necessary Hibernate classes, defines a Main class with a main method, and creates a session factory and transaction. It then prints a message, creates a new 'Departamentos' object with specific values, saves it to the database, commits the transaction, and closes the session. The code is numbered from 1 to 28.

```
1 package practica5;
2
3 import org.hibernate.Session;
4 import org.hibernate.SessionFactory;
5 import org.hibernate.Transaction;
6
7 public class Main {
8     public static void main(String[] args) {
9         //En primer lugar se obtiene la sesión creada por el Singleton.
10        SessionFactory sesion = HibernateUtil.getSessionFactory();
11        //Abrimos sesión e iniciamos una transacción
12        Session session = sesion.openSession();
13        Transaction tx = session.beginTransaction();
14
15        System.out.println("Inserto una fila en departamentos");
16        //Creamos un nuevo objeto Departamentos y damos valor a sus atributos
17        Departamentos dep = new Departamentos();
18        dep.setCodDept(4);
19        dep.setNombre("MARKETING");
20        dep.setDireccion("GUADALAJARA");
21        dep.setObjetivos(10000);
22
23        //Guardamos en la base de datos y comprometemos la información
24        session.save(dep);
25        tx.commit();
26        session.close();
27    }
28 }
```

# 4. CONSULTAS HIBERNATE

---

## Consultas Hibernate

- Hibernate soporta un lenguaje de consulta orientado a objetos denominado **HQL** (Hibernate Query Language)
- Las consultas HQL y SQL son representadas con una instancia de **org.hibernate.Query**.
- La interfaz **Query** ofrece métodos para ligar parámetros, manejo del conjunto resultado y para la ejecución de la consulta real.
- Siempre se obtiene una Query utilizando el objeto **Session** actual.

# 4. CONSULTAS HIBERNATE

---

## Métodos de Query

- **Iterator iterate()** → Devuelve en un objeto Iterator el resultado de la consulta
- **List list()** → Devuelve el resultado de la consulta en un List
- **Query setFetchSize (int size)** → Fija el número de resultados a recuperar en cada acceso a la base de datos al valor indicado en size
- **int executeUpdate()** → Ejecuta la sentencia de modificación o borrado. Devuelve el número de entidades afectadas
- **String getQueryString()** → Devuelve la consulta en un string
- **Query setCharacter(int posición, char valor)** o también **Query setCharacter(String nombre, char valor)** → Asigna el valor indicado en el método a un parámetro de tipo char. Posición indica la posición del parámetro empezando en 0. Nombre es el nombre (:nombre) del parámetro

## 4. CONSULTAS HIBERNATE

---

- **Object uniqueResult()** → Devuelve un objeto (cuando sabemos que la consulta devuelve un objeto) o nulo si la consulta no devuelve resultados
- **Query setDate (int posición, Date fecha)** o también, **Query setDate (String nombre, Date fecha)** → Asigna la fecha a un parámetro de tipo DATE
- **Query setDouble (int posición, double valor)** o también **Query setInteger (String nombre, double valor)** → Asigna valor a un parámetro de tipo FLOAT
- **Query setInteger(int posición, String valor)** o también **Query setInteger(String nombre, String valor)** → Asigna valor a un parámetro de tipo entero
- **Query setString(int posición, String valor)** o también **Query setString(String nombre, String valor)** → Asigna valor a un parámetro de tipo VARCHAR

## 4. CONSULTAS HIBERNATE

---

- **Query setParameterList (String nombre, Collection valores):** Asigna una colección de valores al parámetro cuyo nombre se indica en nombre
- **Query setParameter (int posición, Object valor):** Asigna el valor al parámetro indicado en posición
- **Query setParameter (String nombre, Object valor):** Asigna el valor al parámetro indicado en nombre
- **int executeUpdate():** Ejecuta una sentencia UPDATE o DELETE, devuelve el número de entidades afectadas.
- API de Hibernate: <https://docs.jboss.org/hibernate/orm/current/javadocs/>

## 4. CONSULTAS HIBERNATE

- Para realizar una consulta usaremos el método **createQuery()** de la interfaz **SharedSessionContract**. Se le pasará en un String la consulta HQL. Por ejemplo, para hacer una consulta sobre la tabla departamentos, mapeada con la clase Departamentos, se escribe de la siguiente manera:

```
Query q = session.createQuery("from Departamentos");
```

- Para recuperar los datos de la consulta usaremos el método **getResultSet()** (el método **list()** está deprecated). Devuelve una colección con el conjunto de todos los resultados de la consulta.

```
List <Departamentos> lista = q.getResultSet();
System.out.println("Número de departamentos: " + lista.size());
for(Departamentos dep:lista){
    System.out.println(dep.getCodDept() + " " + dep.getNombre());
}
```

## 4. CONSULTAS HIBERNATE

- Se puede obtener de un determinado departamento, la lista de sus empleados. Basta con usar el método **getEmpleados()** de la clase Departamentos:

```
System.out.print("Introduce id de departamento para ver sus empleados: ");
int id=reader.nextInt();

Departamentos dep = session.get(Departamentos.class, id);
Set<Empleados> listaemp = dep.getEmpleados();
for(Empleados emp:listaemp){
    System.out.println(emp.getId() +" "+ emp.getNombre());
}
```

## 4. PRACTICA 3

---

Realiza un programa en Java que ofrezca las siguientes opciones:

1. Listado de departamentos → **CreateQuery**
2. Listado de empleados → **CreateQuery**
3. Consulta los datos de un determinado departamento por su id → **get**
4. Consulta los datos de un empleado determinado por su id → **load**
5. Consulta los datos de un empleado determinado por el nombre → **CreateQuery**
6. Consulta los datos de los empleados de un departamento, mediante consulta HQL → **CreateQuery**
7. Consulta los datos de los empleados de un departamento, sin consulta HQL. → **get departamento + lista empleados**
8. Consulta el salario medio, máximo y mínimo de todos los empleados. → **createQuery**

## 4. PRACTICA 3

---

9. Consulta los nombres de los empleados junto con el nombre de su departamento → **createQuery**
10. Consulta por cada departamento el número de departamento, el nombre, el número de empleados que tiene y el salario medio. → **createQuery**
11. Inserte un departamento. El programa recibirá los datos de un departamento
12. Inserte un empleado en la tabla empleados. El programa recibe del usuario los valores a insertar (incluido el id). Se deben de comprobar los siguientes casos:
  - Que el departamento no exista en la tabla departamentos
  - Que el empleado ya exista
13. Suba el salario en 10000 a los empleados de un departamento. El programa recibirá el número de departamento y el incremento.
14. Realiza un programa que modifique el salario y el departamento de un empleado determinado, sumando 1000 al salario y asignándole a otro departamento.