

Acceso a base de datos con Hibernate

Marcos Núñez Celeiro – Acceso a datos - 2019/2020

Plain Old Java Objects (POJO)

Los objetos contienen propiedades (o atributos) y métodos que modifican esos atributos o que simplemente los traen (los obtienen).

Los métodos que obtienen esos atributos son los *getters* o *retrievers*, mientras que los que los modifican son los *setters* o *mutators* (mutadores)*.

Los objetos con estas características se conocen habitualmente como POJOS (*Plain Old Java Objects*).

Como ejemplo, una clase “Persona” con sus atributos y métodos para modificar y obtener esos atributos, si no extiende ni implementa nada especial, sus objetos serían considerados POJOs.

* Se ponen los términos en inglés ya que no hay traducción literal corta al español (podría ser mutadores y accesoros o algo similar).

¿Que significa ORM?

Las siglas ORM significan *Object Relational Mapping* o mapeo objeto-relacional. ORM es la conversión de objetos a un modelo de base de datos y viceversa.

Un código de ejemplo de como almacenar un POJO en base de datos sería algo similar a esto:

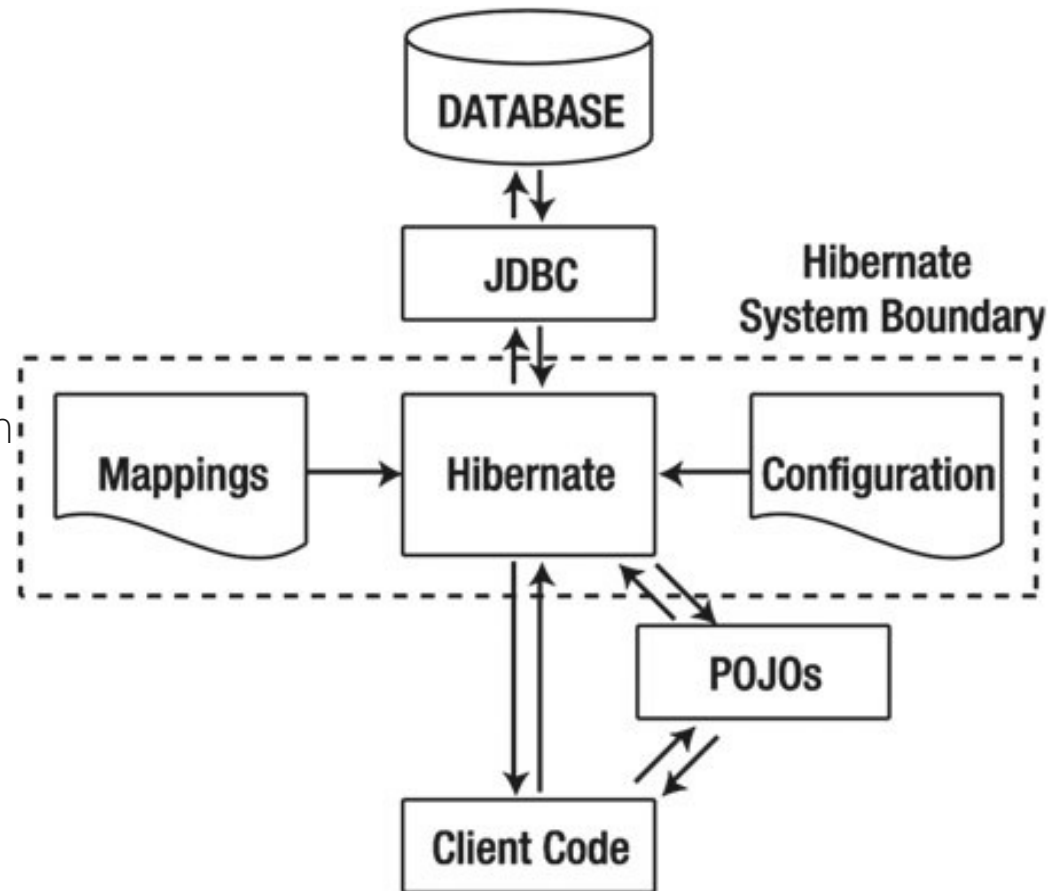
```
POJO pojo = new POJO(); // Creo un objeto
```

```
// Creo una instancia de mi framework ORM  
FrameworkORM magia = new FrameworkORM();
```

```
// Almaceno el objeto  
magia.guardar(pojo);
```

Hibernate

- **Código cliente (*client code*):** código de la aplicación.
- **POJOs:** Objetos Java con sus *getters* y *setters* (Persona, Mascota, Animal, etc.).
- **Hibernate:**
 - Mapeos (*mappings*): Indican como se van a convertir los objetos Java al modelo de base de datos.
 - Configuración: configuración de base de datos (usuario, contraseña, driver a utilizar, tipo de base de datos, etc).
- **JDBC:** Hibernate de manera automática se encarga de transformar las operaciones realizadas a JDBC.



* Imagen obtenida del libro: Beginning Hibernate 4ª edition

Hibernate (II)

Explicado de manera más simple:

En la aplicación se disponen una serie de clases y objetos Java. Mediante Hibernate se realiza la transformación de estos objetos a base de datos relacional y viceversa.



Imagen obtenida de: https://www.tutorialspoint.com/hibernate/hibernate_overview.htm

Hibernate (III)

El programa más simple posible con acceso a base de datos mediante Hibernate debe tener:

- **Clases entidad (POJOs)**: p. ej: Almacén, Producto, Pedido, etc.
- **Una clase *main* donde**: cargo la configuración de Hibernate y realizo las operaciones que quiera con la BBDD.
- **Un fichero XML de configuración**: con los datos de acceso a base de datos (ip, contraseña, etc.).
- **Ficheros XML de *mapping***: que indiquen la correspondencia de las tablas con las clases Java. Por ejemplo: el atributo nombre en Java se corresponde con “name” en la BBDD, id con id, apellidos con surname, etc.

Los ficheros XML (tanto de mapeos como de configuración) se pueden evitar y hacer todo mediante Java. Esto es opcional.

Instalación de los Hibernate Tools (I)

En caso de utilizar Eclipse deberías instalar *plugins* para facilitar el uso de Hibernate (añade menús y opciones para crear ficheros de configuración y otras utilidades).

- **Descarga los JBoss Tools en la siguiente URL:**
<https://tools.jboss.org/>
- **Instala los “Hibernate Tools” (están dentro de los JBoss Tools) en Eclipse.**
 - En la siguiente diapositiva se guía el proceso.

Instalación de los Hibernate Tools (II)

En la página web marcada anteriormente debería salir una URL y unas indicaciones como estas:

4.14.x.Nightly	Nightly
4.13.0.Final	Stable
4.2.3.Final	Stable
All downloads	

JBoss Tools 4.13.0.Final Stable

2019-11-06
[Blog Announcement](#) | [What's New](#) | [Installation Instructions](#)

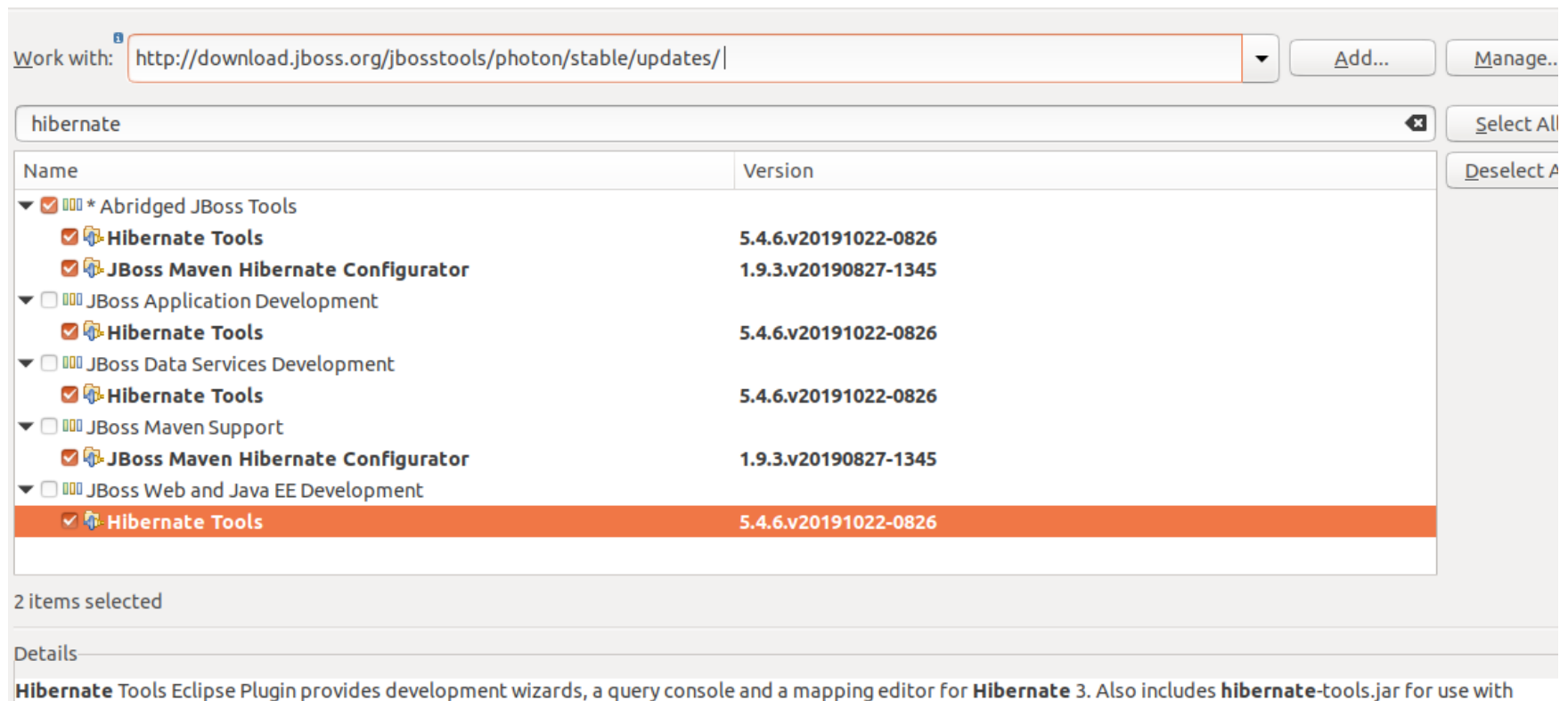
Requirements: Java 8 and [Eclipse 4.13 \(2019-09\)](#)

Update SiteArtifacts

Add the following URL to your **Eclipse 4.13 (2019-09)** installation, via:
Help > Install New Software... > Work with:

Instalación de los Hibernate Tools (III)

Pulsa en Eclipse en “Help” → Install new software. Luego filtra por “Hibernate” y selecciona todas las coincidencias.



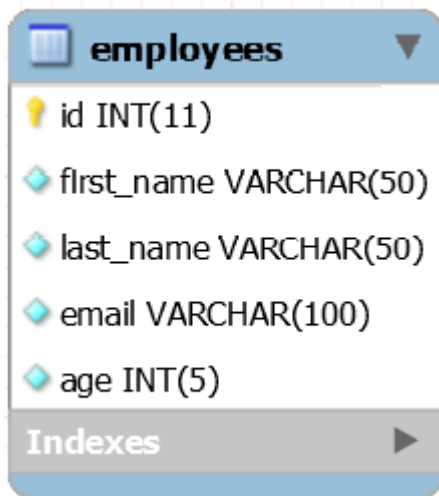
Creación de un proyecto de ejemplo con Hibernate

Creación de un proyecto de ejemplo con Hibernate

- 1. Creación de la base de datos con la que trabajar.**
- 2. Creación de un proyecto *maven* dividido en paquetes.**
- 3. Creación de las entidades del modelo.**
- 4. Configuración de Hibernate.**
- 5. Implementación del código de acceso a base de datos.**
- 6. Ficheros y/o clases de conversión o mapeo (*mapping*).**

Creación de la base de datos

Partimos de una base de datos de empleados con la siguiente estructura:



employees	
id	INT(11)
first_name	VARCHAR(50)
last_name	VARCHAR(50)
email	VARCHAR(100)
age	INT(5)
Indexes	

```
DROP DATABASE IF EXISTS EMPLOYEES_DB;
```

```
CREATE DATABASE EMPLOYEES_DB;
```

```
use EMPLOYEES_DB;
```

```
DROP TABLE IF EXISTS employee;
```

```
CREATE TABLE employee (  
    id int(11) NOT NULL AUTO_INCREMENT,  
    first_name varchar(50) NOT NULL,  
    last_name varchar(50) NOT NULL,  
    email varchar(100) NOT NULL,  
    age int(5) NOT NULL,
```

```
    PRIMARY KEY (id),  
    UNIQUE KEY (email)  
);
```

Creación de un proyecto Maven

Maven es una herramienta de creación y gestión de proyectos Java. Maven permite la gestión y descarga automática de dependencias.

Ejercicio I: crea una base de datos con la tabla de empleados de la diapositiva anterior (puedes copiar el código del *script*).

Ejercicio II: Inicializa, desde el Eclipse de Java EE, un nuevo proyecto de Maven. Puedes servirte de este tutorial.

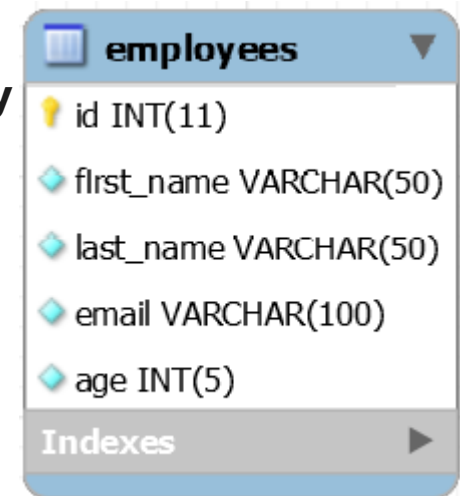
Nota: se aconseja creación de arquetipo: maven-quickstart

Estructura de paquetes y entidades del dominio

Una vez creada la base de datos anterior vamos a intentar hacer un pequeño programa simple que trabaje con Hibernate. Además, posteriormente mejoraremos el programa implementando operaciones CRUD (*create, read, update, delete*) de la tabla de empleados.

Lo primero que deberíamos hacer es preparar una “arquitectura” para el mismo, en este caso podemos hacer una en dos capas (vista y acceso a datos) con los siguientes paquetes:

- vista (interfaz de usuario, tendrá la clase principal con el método *main*).
- modelo.entidades: entidades del dominio (los POJOs).
- modelo.dao: métodos de acceso a base de datos.
- configuracion: clases de configuración (en este caso se necesita, al menos, la configuración de BBDD).



employees	
id	INT(11)
first_name	VARCHAR(50)
last_name	VARCHAR(50)
email	VARCHAR(100)
age	INT(5)
Indexes	

Ejercicio I: crea la jerarquía de paquetes indicada en esta diapositiva.

Ejercicio II: crea una clase Empleado (dentro de un paquete modelo.entidades). Utiliza los estándares de Java para nombrar los atributos (p. ej: en Java se utiliza camelCase y no “_”).

Configuración de Hibernate (I)

Para que Hibernate sea capaz de acceder a base de datos es necesario proporcionar una serie de parámetros, algunos de ellos son:

- **URL de conexión.**
- **Nombre de la base de datos a la cuál se desea acceder.**
- **Usuario y contraseña de la base de datos.**
- **Driver y dialecto de base de datos (PosgreSQL, SQLServer, Oracle, MySQL...).**

Configuración de Hibernate (II)

Existen dos formas principales de configuración para Hibernate:

- **Mediante configuración nativa:**

- Se proporcionan los datos de configuración en un fichero XML de nombre "hibernate.cfg.xml" o bien en un fichero "*hibernate.properties*".
- También es posible proporcionar los parámetros de configuración mediante código Java.

- **Siguiendo el estándar JPA:**

- Se proporcionan los datos de configuración mediante un fichero *persistence.xml*.
- Las ventajas de utilizar este estándar es que podrías cambiar de Hibernate a otro *framework* de acceso a datos sin tener que modificar la configuración. Es decir, se dispone de una configuración no dependiente de la tecnología. Si sigues el estándar JPA se puede cambiar fácilmente entre cualquier tecnología que lo siga.
- JPA utiliza los elementos definidos en el paquete javax.persistence.

Configuración de Hibernate (III)

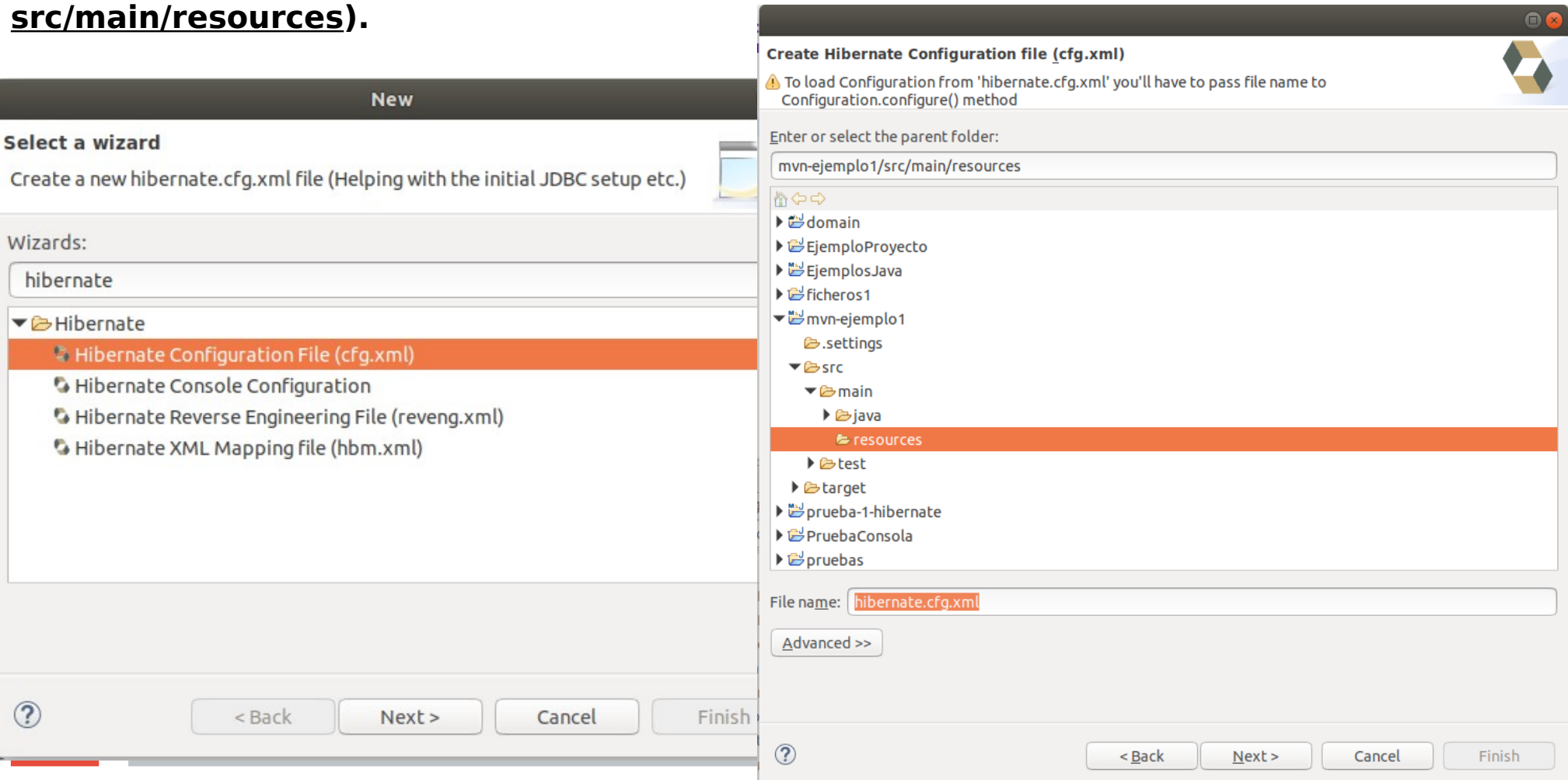
En este tutorial se va a utilizar el enfoque nativo. A continuación se muestra un ejemplo de fichero de configuración de Hibernate con los parámetros básicos (hibernate.cfg.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.password">userpassword</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/employees_db</property>
    <property name="hibernate.connection.username">username</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  </session-factory>
</hibernate-configuration>
```

Configuración de Hibernate (IV)

Ejercicio: crea un fichero de configuración en Hibernate similar al de la diapositiva anterior (pero con tus datos de conexión).

*** Si estás en Eclipse puedes ayudarte de los Hibernate Tools para ello: botón derecho en el proyecto → new → Other → Hibernate configuration file (en src/main/resources).**



Configuración de Hibernate (IV)

A continuación se utilizan las clases de Hibernate *StandardServiceRegistry* y *SessionFactory*:

- La primera parte busca un fichero hibernate.cfg.xml en el *classpath* y construye un objeto *StandardServiceRegistry*.
- La segunda parte crea una fábrica de sesiones (las cuales podrás utilizar posteriormente para realizar operaciones contra base de datos).

```
public class App {  
    public static void main(String[] args) {  
        // Lee la configuracion del hibernate.cfg.xml  
        StandardServiceRegistry registry = new StandardServiceRegistryBuilder()  
            .configure()  
            .build();  
  
        // Fabrica (factory) de sesiones de acceso a base de datos  
        // Una SessionFactory contiene todos los datos de acceso a bd  
        SessionFactory factory = new MetadataSources(registry)  
            .buildMetadata()  
            .buildSessionFactory();  
    }  
}
```

Configuración de Hibernate (V)

Hasta aquí se dispone de los siguientes componentes:

- **Un fichero de configuración hibernate.cfg.xml con los datos de acceso a base de datos y de configuración básica de hibernate.**
- **En el método *main*, de un objeto SessionFactory que contiene los parámetros de configuración necesarios para acceder a la base de datos.**

Ejercicio: carga el fichero hibernate.cfg.xml desde código (esto se hace mediante el código de la diapositiva anterior). Ten en cuenta que Hibernate no viene con el JDK: tendrás que añadir la biblioteca al pom.xml (búscala en mvnrepository).

Código de acceso a base de datos (I)

Ahora mismo no hay nada en base de datos (la tabla *employees* está vacía). A continuación se presenta un código de ejemplo que abre una nueva sesión y almacena un empleado:

```
Session session = factory.openSession();  
  
Employee e = new Employee("Lionel", "Messi", "", 31);  
  
session.save(e);  
session.close();
```

Código de acceso a base de datos (II)

El código del método *main* quedaría de la siguiente manera:

```
46 public class App {
47     public static void main(String[] args) {
48         // Lee la configuracion del hibernate.cfg.xml
49         StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
50             .configure()
51             .build();
52
53         // Fabrica (factory) de sesiones de acceso a base de datos
54         // Una SessionFactory contiene todos los datos de acceso a bd
55         SessionFactory factory = new MetadataSources(registry)
56             .buildMetadata()
57             .buildSessionFactory();
58
59         Session session = factory.openSession();
60
61         Employee e = new Employee("Lionel", "Messi", "", 31);
62
63         session.save(e);
64         session.close();
65
66         factory.close();
67     }
68 }
```

Ejercicio

- 1. ¿Qué crees que hace el código anterior?
Ejecútalo y comprueba el resultado.**
- 2. ¿Por qué crees que falla el programa?
Imagínate que tu eres el *framework*, piensa con cuidado en que datos tienes y en cuáles te faltan y piensa que mas crees que necesita saber (consejo: lee lo que te dice la excepción).**
 1. Una pista: Imagínate que en tu base de datos tienes dos tablas para los empleados, una antigua que ya no usas y una nueva. Una tabla (la vieja) se llama EmployeesOLD y la nueva se llama Employees... ¿con cuál va a trabajar hibernate?

Ejercicio (II)

3. En estas diapositivas se está trabajando con un enfoque nativo de Hibernate. Revisa el ejemplo definido en [este tutorial donde se utiliza el estándar JPA](#) e identifica las diferencias que encuentras entre ambos y señálalas.

Ficheros y/o clases de conversión o mapeo (*mapping*).

- **El programa anterior falla porque a Hibernate no se le ha indicado en ningún sitio que la entidad “Employees” de base de datos debe mapearse con la clase Java “Employee” o “Empleado”.**
- **Hay dos maneras de indicar los mapeos en Hibernate:**
 - **Anotaciones:** añadiendo anotaciones a lo largo de la clase “Empleado” se pueden dar las indicaciones necesarias a Hibernate para el mapeo.
 - **Fichero XML:** se puede crear un fichero XML para indicar toda la información de conversión (atributoX de la clase es atributo Y de base de datos, clase X es tabla Y de base de datos, etc). En este ejemplo el fichero sería empleado.hbm.xml.

Mapeo mediante anotaciones

Primero, en el fichero de configuración de Hibernate (¿recuerdas cuál es?) debe añadirse una línea para cada una de las clases que se mapeen a base de datos. En el caso de una clase “Employee” sería una línea como la siguiente:

```
<mapping class="aulanosa.segundo.ad.maven.ejemplo1.ejemploshibernate.model.entities.Employee" />
```

En el atributo “*class*” debe indicarse toda la ruta de la clase (esto es, el nombre del paquete y la clase al final).

Ejercicio: añade una línea con la referencia a tu clase “Empleado” en tu fichero de configuración (una línea como la anterior).

Ejercicio

Consulta la documentación de la siguiente página:

<http://www.techferry.com/articles/hibernate-jpa-annotations.html>

Posteriormente añade las anotaciones (una a una) a la clase “Empleado” hasta que el empleado que se ha creado en el *main* se almacene correctamente. Utiliza las mínimas que sean necesarias.

NOTA: Cada vez que añadas una nueva anotación piensa en que otra información debes aportar a la clase (que otras anotaciones tendrías que añadir). Ejecuta el programa con cada anotación que añadas para consultar las excepciones que se producen.

Ejercicio (II)

- **Al principio del tema se han creado una serie de paquetes (con la intención de que el programa quede lo más modular y limpio posible).**
 - Ahora que se ha completado la configuración de Hibernate y el código ya funciona correctamente refactorízalo creando una clase ConfiguracionHibernate con un método que devuelva la SessionFactory para dejar más limpio el método *main*.
 - Mejora un poco el diseño del programa asegurándote de que las sesiones se cierran siempre al acabar, ¿recuerdas como se podía hacer?
 - Ejecuta el programa y asegúrate de que funciona.

Mapeo mediante XML (I)

Para indicar las características de la conversión mediante XML es necesario crear el fichero “nombreEntidad.hbm.xml”. En este caso se ha creado “Employee.hbm.xml”.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="aulanosa.segundo.ad.maven.ejemplo1.ejemploshibernate.model.entities.Employee"
    table="EMPLOYEES">

    <id name="id" type="java.lang.Integer">
      <column name="ID" />
      <generator class="identity" />
    </id>
    <property name="firstName" type="java.lang.String">
      <column name="first_name" />
    </property>
    <property name="lastName" type="java.lang.String">
      <column name="last_name" />
    </property>
    <property name="email" type="java.lang.String">
      <column name="email" not-null="true" unique="true" />
    </property>
    <property name="age" type="int">
      <column name="age" />
    </property>
  </class>
</hibernate-mapping>
```

En este fichero se indica inicialmente el nombre de la clase y la tabla a la que se mapea y, posteriormente, las propiedades (atributos) con su columna correspondiente y su tipo de dato.

Mapeo mediante XML (II)

```
<class name="aulanosa.segundo.ad.maven.ejemplo1.ejemploshibernate.model.entities.Employee"
table="EMPLOYEES">

  <id name="id" type="java.lang.Integer">
    <column name="ID" />
    <generator class="identity" />
  </id>
  <property name="firstName" type="java.lang.String">
    <column name="first_name" />
  </property>
  <property name="lastName" type="java.lang.String">
    <column name="last_name" />
  </property>
  <property name="email" type="java.lang.String">
    <column name="email" not-null="true" unique="true" />
  </property>
  <property name="age" type="int">
    <column name="age" />
  </property>
</class>
```

Los elementos de un fichero de mapeo **son los siguientes**:

- **Elemento class**: define la clase Java y la tabla a la que mapea. En este caso la clase “*Employee*” mapea a la tabla “*employees*”.
- **Elemento id**: indica cual es la clave primaria de la tabla. En este caso “*id*” mapea con la *column* “*id*” (se llaman igual).
- **Elemento generator**: permite decir a hibernate como se generan las IDS. En este caso se asigna *identity*, que indica que es la base de datos la que se encarga de asignar las ids (en la base de datos se ha puesto autoincremental). ([Ver más](#))
- **Elemento property**: Mapea una propiedad o atributo de la clase a la columna de base de datos: *firstName* con *first_name*, etc.

Mapeo mediante XML (III)

Es necesario indicarle a Hibernate el sitio donde se ha colocado el fichero. Para ello, en el fichero de configuración `hibernate.cfg.xml` es necesario escribir la siguiente línea:

```
<mapping resource="aulanosa\segundo\ad\maven\ejemplo1\ejemplohibernate\model\entities\Employee.hbm.xml" />
```

Con anotaciones se indicaba `class="paquetes.clase"`. En este caso, al tratarse de un fichero, se indica con *resource* y separando con barras (\).

Ejercicio

1. Cambia la configuración de tu proyecto para que el *mapping* se haga con XML en vez de con anotaciones.

- No es necesario que elimines las anotaciones. Es suficiente con cambiar el *mapping* que se ha indicado en el fichero de configuración de Hibernate para que utilice el fichero que le digas (atributo *resource* en vez de *class*).
- Gracias a los Hibernate Tools que hemos instalado disponemos de interfaz gráfica para crear el fichero de mapeo (puedes hacer uso de ellos). Recuerda, es un XML y por tanto va en la carpeta *src/main/resources*.

Ejemplo: Código ORM vs JDBC

A continuación se muestra un código que almacena un objeto “Vuelo” utilizando el Driver de JDBC:

```
public void insertar(Vuelo v) throws SQLException {  
    String sql = "INSERT INTO vuelo (codVuelo, modeloAvion, numPasajeros, capacidad) "  
        + "VALUES (?, ?, ?, ?)";  
  
    PreparedStatement insertVuelo = connection.prepareStatement(sql);  
  
    insertVuelo.setString(1, v.getCodigoVuelo());  
    insertVuelo.setString(2, v.getModeloAvion());  
    insertVuelo.setInt(3, v.getNumPasajeros());  
    insertVuelo.setInt(4, v.getMaxPasajeros());  
  
    insertVuelo.executeUpdate();  
}
```

- Piensa en como quedaría el código de almacenamiento de esta entidad Vuelo utilizando Hibernate (acabas de almacenar un objeto “Empleado”).
- Infórmate en *internet* sobre las ventajas e inconvenientes de utilizar Hibernate en vez de JDBC para manejar información de bases de datos.

Hibernate Query Language (HQL)

En Hibernate se pueden ejecutar consultas similares a SQL sobre las clases. Por ejemplo, si se desean obtener todos los empleados de la tabla “*employees*” la consulta en SQL sería la siguiente:

SELECT * FROM EMPLOYEES;

Cuando trabajamos con Hibernate se puede utilizar un lenguaje más amigable y más cercano a las clases Java que a base de datos. Utilizando Hibernate y HQL la consulta anterior se puede hacer de la siguiente manera:

```
// Obtener lista de empleados (metodo createQuery de Session)
String selectAll = "from Employee";
List<Employee> empsDB = session.createQuery(selectAll,
Employee.class).getResultList();
```

Hibernate Query Language (HQL) - Observaciones

Escribiendo la consulta de abajo se genera una consulta como la de arriba.

En Hibernate no es necesario poner el **SELECT *** en las consultas (si se inicia la consulta desde el *from* se asume que se trata de un **SELECT ***). Por lo que la consulta anterior podría quedar de la siguiente manera:

```
String selectAll = "from Employee";
```

Con esto se generaría exactamente un **SELECT * FROM Employees**.

Preguntas:

¿Qué diferencias aprecias entre el código HQL y el SQL?

¿Te has fijado en que el nombre de la tabla en un caso es *Employee* y en el otro *Employees*? ¿Por qué en HQL se utiliza *Employee*?

Ejercicio

Siguiendo los ejemplos del código anterior y documentación que encuentres en internet realiza en tu código las siguientes operaciones:

- 1. Obtener de base de datos todos los empleados y mostrar sus datos (puedes crear a mano o autogenerar el *toString()*).**
- 2. Obtener de base de datos solo los nombres de todos los empleados y posteriormente mostrarlos.**
- 3. Obtener el número de empleados existentes en la base de datos.**
- 4. Obtener de base de datos el empleado de id 2 y mostrar su nombre y apellidos.**

Hibernate Query Language (HQL)

- 1) Creación del String con la consulta (recuerda que se puede omitir el **SELECT *** en HQL).
- 2) Creación de un objeto Query en el que se le indica que el parámetro **id (:id)** es 129.
- 3) En la consulta que se está realizando se espera la devolución de una única fila. El método **getResultList()** devuelve una lista, por tanto hay que comprobar si se ha devuelto el elemento buscado (el empleado de id 129) o no. Si la lista está vacía es que no existe el empleado de id 129 y si no lo está se obtiene el empleado y se muestran sus datos.

```
// Obtener el empleado de id 129
```

```
String queryStringemp129 = "from Employee where id = :id";
```

```
Query<Employee> queryId129 = session.createQuery(queryStringemp129, Employee.class);
```

```
queryId129.setParameter("id", 129);
```

```
List<Employee> emps = queryId129.getResultList();
```

```
if (!emps.isEmpty()) {
```

```
    Employee eId129 = emps.get(0);
```

```
    System.out.println("Empleado de id 129: " + eId129.getFirstName() + " " +  
eId129.getLastName());
```

```
} else {
```

```
    System.out.println("No se ha encontrado el empleado 129.");
```

```
}
```

Ejercicio: refactorización de código

Crea una interfaz “**IEmpleadosDao**” con su implementación “**EmpleadosDao**” de manera que contengan los siguientes métodos:

- Obtener todos los empleados (getAll).
- Obtener un empleado por id (getById).
- Añadir un empleado (add, create, insert...).
- Actualizar un empleado (update).
- Eliminar un empleado (delete o remove).

Finalmente, tendrás que refactorizar y ampliar el código que has utilizado hasta ahora para que, en vez de hacer todo en el *método main*, se hagan las llamadas al DAO.

Ejercicio: JUnit

Con la resolución del ejercicio anterior se ha conseguido separar la capa de lógica e interfaz del modelo de datos.

Aún así, ahora mismo en el *main* se dispone de un código “de pruebas”. Este código no es algo que utilizarías para gestionar los empleados de una empresa en el mundo real, pero sí que se podría utilizar para comprobar si la aplicación funciona correctamente.

Ejercicios:

- 1) Añade a *maven* la biblioteca de JUnit 5 y crea un fichero de test.
- 2) Crea un fichero EmpleadosTest y, con ayuda del profesor, mueve todo el código de las pruebas que se han hecho hasta ahora a ese fichero como tests unitarios.

Para realizar este ejercicio sírrete de la ayuda del profesor y de internet. Puedes consultar este tutorial.

Consejo: Utiliza las anotaciones `@BeforeAll` para realizar la configuración de Hibernate y `@AfterAll` para cerrar la sessionFactory.

Relaciones en hibernate: 1 a 1 (*one to one*)

En una relación uno a uno en hibernate la clave primaria de ambas tablas debe ser la misma. La clave foránea de la segunda tabla es también su clave primaria y referencia a la primera tabla.

Por ejemplo: en una relación de base de datos entre una tabla Usuario y una tabla investigador, si investigador tiene como clave primaria “id_usuario”, la cual referencia al id de la tabla “Usuario”, estamos hablando de una relación uno a uno.

Ejercicio opcional: haz que los empleados tengan un detalle (por ejemplo EmpleadoDetalle) donde se establezcan los años trabajados y una descripción.

Relaciones en hibernate: 1 a n (*one to many*) y n a 1 (*many to one*)

En una relación uno a “n” en base de datos la parte “n” debe tener como clave foránea la clave de la primera tabla. En las clases hibernate esto se modela dando a la parte “1” un atributo que contenga un conjunto de datos de la parte “n”, mientras que la parte “n” debe tener la referencia al objeto de la parte “1”.

Por ejemplo, en una empresa se disponen de una lista de empleados. La clase Empresa debe tener un List o Set de objetos empleado y a su vez la clase Empleado debe tener un objeto de tipo Empresa (la empresa que lo contiene).

Ejercicio: haz que los empleados puedan tener una o varias direcciones (necesitas crear una clase Direccion). La dirección debe tener: calle, código postal, portal, piso y puerta.

Crea también una tabla “Direccion” en la base de datos.

Relaciones en hibernate: n a n (*many to many*)

En una relación muchos a muchos en base de datos es necesaria la creación de una tabla intermedia. Por ejemplo, considerando lo realizado anteriormente de que un empleado tiene una lista de direcciones, si además varios empleados pueden vivir en la misma dirección haría falta la creación de una tabla intermedia, por ejemplo EMPLEADOS_DIRECCIONES que tenga la id del empleado y la id de la dirección para representar la relación entre los elementos.

En Java sería necesario indicar que un empleado tiene una lista o conjunto (List o Set) de direcciones y que la clase Direccion tiene, a su vez, una lista o conjunto de empleados.

Ejercicio: haz que los empleados puedan trabajar en varias empresas (debes crear una tabla Empresa) y las empresas contengan empleados.

FetchTypes: Eager y Lazy

En los mapeos de hibernate se puede indicar a una entidad si se desea que los datos asociados se obtengan inmediatamente o según vayan siendo necesarios.

En una relación de un empleado que tiene una lista de proyectos, si esta última se pone como lazy, al obtener un empleado los proyectos no son recuperados (al menos no inmediatamente). Con Eager, en cambio, se obtienen los datos de los proyectos del empleado de manera inmediata.

Es importante destacar que si hacemos un `empleado.getProyectos()` con la lista establecida como perezosa (*lazy*) la consulta a base de datos se hace en ese momento. Si la sesión está cerrada, en cambio, los datos no podrán obtenerse.

Criteria API

Aunque no se va a profundizar más en el trabajo con Hibernate, se recomienda consultar este enlace como documentación sobre la realización de consultas con *Criteria*. Esta clase permite realizar diversas consultas con filtros y condiciones de manera simple sin necesidad de utilizar SQL.

SQL Nativo

En Hibernate también es posible realizar **consultas nativas de la manera que lo harías utilizando JDBC.**

Para recordar...

Existen dos posibilidades de utilización de Hibernate:

- **Mediante el estandar JPA 2.1:** si se sigue totalmente este estándar la configuración se realiza en el fichero `persistence.xml`, las bibliotecas a importar son las de javax y las clases Java de configuración son `EntityManagerFactory` y `EntityManager`.
- **Mediante configuración nativa de Hibernate:** el fichero de configuración es el `hibernate.cfg.xml` (también se pueden almacenar en un `hibernate.properties` como clave valor). Se usan las anotaciones de JPA (`import javax...` las de hibernate nativas se consideran *deprecated*) y las clases Java de configuración son `SessionFactory` y `Session`. Desde la versión 5.2 el API nativa de Hibernate para trabajar con *Criteria* está deprecated, por tanto se utiliza de nuevo la de JPA.

Para recordar...

- **Se realiza la configuración mediante el fichero [hibernate.cfg.xml](#) (hibernate) o el fichero [persistence.xml](#) (JPA).**
- **Las clases de configuración que cargan la información de los ficheros anteriores son (dos opciones):**
 - EntityManagerFactory y EntityManager (JPA).
 - SessionFactory y Session (Hibernate nativo).
- **Los mappings pueden realizarse utilizando ficheros XML o anotaciones (siempre importando las anotaciones de JPA). Las anotaciones de Hibernate se consideran *deprecated*.**
- **Se pueden realizar consultas con Hibernate de diversas maneras:**
 - [Métodos save o persist para almacenar, update, delete...](#)
 - Utilizando HQL (recuerda, con HQL no se ponen nombres de tablas y columnas, sino de las clases y atributos mapeados).
 - Utilizando [Criteria](#).
 - Utilizando [SQL nativo](#) (se pueden realizar consultas nativas sobre tablas como harías con JDBC).

Para recordar...

- **Las relaciones pueden ser:**

- Uno a uno (one to one): una entidad tiene otra asociada y ambas usan la misma clave primaria.
- Uno a muchos (one to many): una clase tiene una lista o conjunto de objetos de otra (persona guarda un Set o List de direcciones)
- Muchos a uno (many to one): la clase “muchos” tiene que tener una referencia a la que lo contiene (direccion tiene una referencia a la persona).
- Muchos a muchos (many to many): ambas clases tienen una lista de la otra entidad. No se crea clase intermedia, pero sí tabla en base de datos.

- **Se recomienda utilizar siempre conjuntos (Set, HashSet) en vez de Listas (List, ArrayList, LinkedList) para las relaciones en Hibernate para evitar problemas.**

Para recordar...

- Los objetos de las relaciones pueden obtenerse de manera perezosa (LAZY) o inmediata (EAGER). Una vez has hecho la consulta, si sus entidades relacionadas están definidas como *lazy* sus consultas a base de datos se realizan en el momento de hacer la llamada para obtener sus datos. En caso de que la sesión esté cerrada no podrían obtenerse.

Parámetros adicionales de configuración

En el fichero de configuración de hibernate es posible indicarle al *framework* gran cantidad de opciones, entre ellas se pueden destacar las siguientes:

- Mostrar en consola consultas que va realizando a base de datos (formateándolas) mediante los atributos `show_sql` y `format_sql`.
- Modificar el número máximo de hilos permitido.
- Teniendo creadas y mapeadas las clases Java, se puede decir a Hibernate que se ocupe el mismo de crear todas las tablas (sin tener que hacerlo tu manualmente). Las opciones son “create”, “create-drop”, “update” y “validate”.

```
<!-- Muestra las consultas SQL en consola (formateadas) -->  
<property name="hibernate.show_sql">true</property>  
<property name="hibernate.format_sql">true</property>
```

```
<!-- Crea automaticamente las tablas en base de datos -->  
<property name="hibernate.hbm2ddl.auto">create-drop</property>
```

```
<!-- Commit automatico (por defecto ya es false) -->  
<property name="hibernate.connection.autocommit">>false</property>
```

Hibernate tools (Eclipse)

En la diapositiva anterior hemos visto que se puede configurar Hibernate para **generar automáticamente las tablas** de base de datos habiendo creado manualmente las clases Java.

Eclipse tiene herramientas que permiten hacer también el proceso inverso: crear las clases Java (con sus respectivos XML u anotaciones de mapeo) automáticamente a partir de las tablas de base de datos.

Para esto puedes instalar las *hibernate tools* (quizás ya las tengas, se ha indicado al principio del tema) y, indicándole a Eclipse la URL de conexión y las tablas a utilizar él mismo se encarga de crear todas las clases correspondientes en las carpetas que se le indiquen.

Referencias

- **Beginning Hibernate 5, 4ª edición**
- <https://o7planning.org/en/10201/java-hibernate-tutorial-for-beginners#a77362>
- <https://es.wikipedia.org/wiki/Maven>
- <https://www.journaldev.com/3422/hibernate-native-sql-query-example>
- <https://www.baeldung.com/hibernate-criteria-queries>
- <https://dzone.com/articles/hibernate-5-xml-configuration-example>
- <https://o7planning.org/en/10125/using-hibernate-tools-generate-entity-classes-from-tables>

Acceso a base de datos con Hibernate

Marcos Núñez Celeiro – Acceso a datos - 2019/2020