Acceso a Datos

UT4. HERRAMIENTAS DE MAPEO OBJETO RELACIONAL (ORM)

1. Introducción

- En esta unidad vamos a trabajar con Bases de Datos Relacionales desde aplicaciones Java pero a diferencia de lo sucedido en la unidad anterior vamos a trabajar directamente sobre el *desfase objeto - relacional.*
- **El desfase objeto-relacional** hace referencia a la evidente diferencia que presenta el tratamiento de la información por parte de una aplicación orientada a objetos frente a cómo trabaja una base de datos relacional.
  - Mientras que las bases de datos relacionales tratan la información como relaciones y conjuntos (No deja de ser un modelo matemático) el paradigma de programación orientada a objetos trata con instancias de clases (Objetos) y las relaciones entre ellas.
  - De modo que, cada vez que queremos leer (o escribir) información de una BBDD relacional desde nuestra aplicación OO necesitamos un proceso de “adaptación” de la misma.

1. Introducción

- La gran mayoría de aplicaciones con las que tratamos en diario hacen uso de fuentes de información (BBDD) cuyo diseño presenta una enorme complejidad:
  - Relaciones de uno a muchos con diferentes tablas.
  - Relaciones que dependen de valores específicos en terceras tablas.
  - Restricciones que implican cambios en diferentes puntos de la BBDD cuando se producen determinadas circunstancias.
  - Etc.
- Estas dificultades inherentes a la gran diferencia existente entre ambos paradigmas hace que la programación de aplicaciones con acceso a bases de datos relacionales pueda resultar ardua, compleja y sea una importante fuente de errores debido a la casuísticas que no lleguen a contemplarse en el diseño de las aplicaciones.
- Por todo ello nacieron las **herramientas de mapeo objeto-relacional o Herramientas ORM.**
  - Estas herramientas se encargan de *"traducir" las representaciones de los datos en el modelo relacional a una representación de clases y objetos que nos ayuden en el desarrollo de nuestra aplicación.*

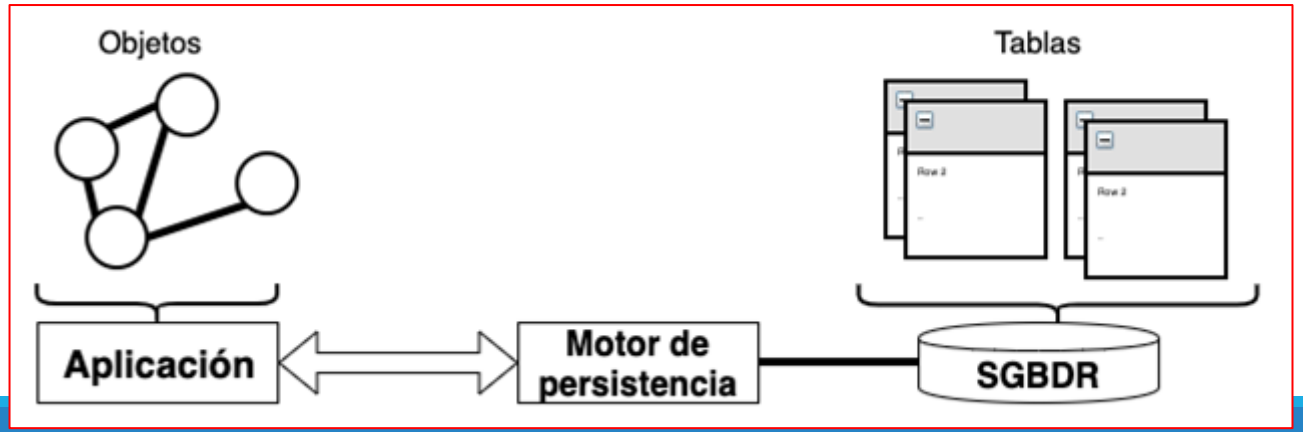
1. Introducción

- Ejemplo de consulta a base de datos relacional con JDBC

```
public List<Alumno> findAll() throws ClassNotFoundException, SQLException {
        String sql = "SELECT ID_ALUMNO, NOMBRE, APELLIDOS,"
                + " TO_CHAR(FECHA_NACIMIENTO,'DD/MM/YYYY') AS FEC_NAC, "
                + " CORREO_ELECTRONICO, HORAS_FALTAS "
                + " FROM ALUM";

        List<Alumno> result = null;

    //Este método nos devolvería un objeto PreparedStatement
    //listo para ejecutar la sentencia de más arriba
        PreparedStatement ps = DBConnection.getPreparedStatement(sql);

        ResultSet rs = ps.executeQuery();

        while(rs.next()) {
            if (result == null)
                result = new ArrayList<Alumno>();

            result.add(new Alumno(rs.getInt("ID_ALUMNO"),
                    rs.getString("NOMBRE"),
                    rs.getString("APELLIDOS"),
                    rs.getString("FEC_NAC"),
                    rs.getString("CORREO_ELECTRONICO"),
                    rs.getInt("HORAS_FALTAS")));
        }

        rs.close();
        ps.close();

        return result;
    }
```

2. Concepto de mapeo objeto-relacional

- El mapeo objeto-relacional es una 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 base de datos relacional, utilizando un motor de persistencia.
- En la práctica esto crea una base de datos orientada a objetos virtual que actúa de interfaz entre la base de datos relacional y la aplicación orientada a objetos.
  - Esto posibilita el uso de las características propias de la programación orientada a objetos.



3. Herramientas ORM

- Existen multitud de herramientas libres y comerciales e incluso algunos programadores optan por desarrollar sus propias herramientas de mapeo.
- Las herramientas ORM nos permiten crear una capa de **acceso a datos**.
  - Una forma sencilla y válida de hacerlo es crear una clase por cada tabla de la base de datos y mapearlas una a una. (De forma similar a como ya estudiamos en la UT3)
- Estas herramientas aportan un lenguaje de consultas propio y totalmente independiente de la base de datos, lo que nos permitirá migrar de una BBDD a otra sin realizar modificaciones en el código.
  - Tan solo requerirá cambios menores en el fichero de configuración.

3. Herramientas ORM

- Algunas de las ventajas que aportan estas herramientas son:
  - Reducen el tiempo de desarrollo del software.
  - Abstracción de la base de datos.
  - Reutilización.
  - Permiten la producción de mejor código.
  - Son independientes de la BBDD, funcionan en cualquier SGBD.
  - Lenguaje propio para realizar consultas.
  - Incentivan la portabilidad y escalabilidad de los programas de software.

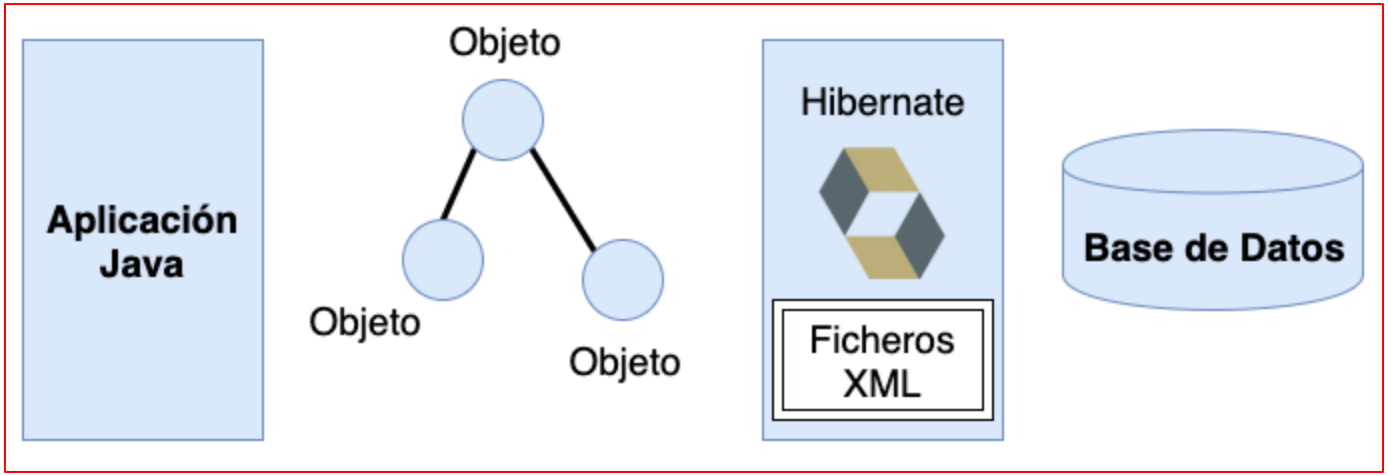
- El mayor inconveniente es que las aplicaciones requieren algo más de potencia computacional debido a que todas las consultas sobre la base de datos deben ser previamente transformadas del lenguaje propio de la herramienta al correspondiente al SGBD.

3. Herramientas ORM

- Existen en el mercado una amplia variedad de herramientas ORM:
  - Doctrine (PHP)
  - Propel (PHP)
  - Nhibernate (C#)
  - Django ORM (Python)
  - Hibernate (Java)
  - EclipseLink(Java)
  - etc.
- De todos ellos, nosotros vamos a tratar específicamente Hibernate ya que está desarrollado con tecnología Java.
- **Hibernate** es software libre bajo licencia GNU LGPL, lo que lo ha convertido en uno de los ORM más populares y extendidos a nivel mundial.

3. Herramientas ORM

- Hibernate es una herramienta de mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación, mediante ficheros declarativos (XML) que permiten establecer relaciones.



3. Herramientas ORM

- Hibernate vs JDBC

Tecnología	Ventajas	Desventajas
JDBC	Ofrece un rendimiento superior. Además, nos permite utilizar al máximo las funcionalidades concretas de nuestro servidor de base de datos	El desarrollo y mantenimiento de nuestras aplicaciones es mucho más costoso. Además, es muy fácil introducir errores durante el desarrollo.
Hibernate	Podemos desarrollar más rápidamente, y haciendo uso de entidades (objetos) en lugar de consultas. Es una gran ayuda en la fase de mantenimiento, y nos permite eliminar al 100% los errores de ejecución debido al acceso a bases de datos	Tiene una curva de aprendizaje mayor, y normalmente no conseguirá el mismo rendimiento que el acceso a base de datos de forma nativa.

3. Herramientas ORM

- Hibernate se está convirtiendo en el estándar de facto para almacenamiento persistente cuando queremos independizar la capa de negocio de la capa de almacenamiento de la información.
- Esta capa de persistencia permite abstraer al programador Java de las particularidades de una determinada base de datos proporcionando clases que encapsularán los datos recuperados de las filas de las tablas.
- Hibernate busca solucionar la diferente entre los modelos de datos usados para organizar y manipular datos:
    - El modelo de objetos proporcionado por el lenguaje de programación y el modelo relacional usado en las bases de datos.

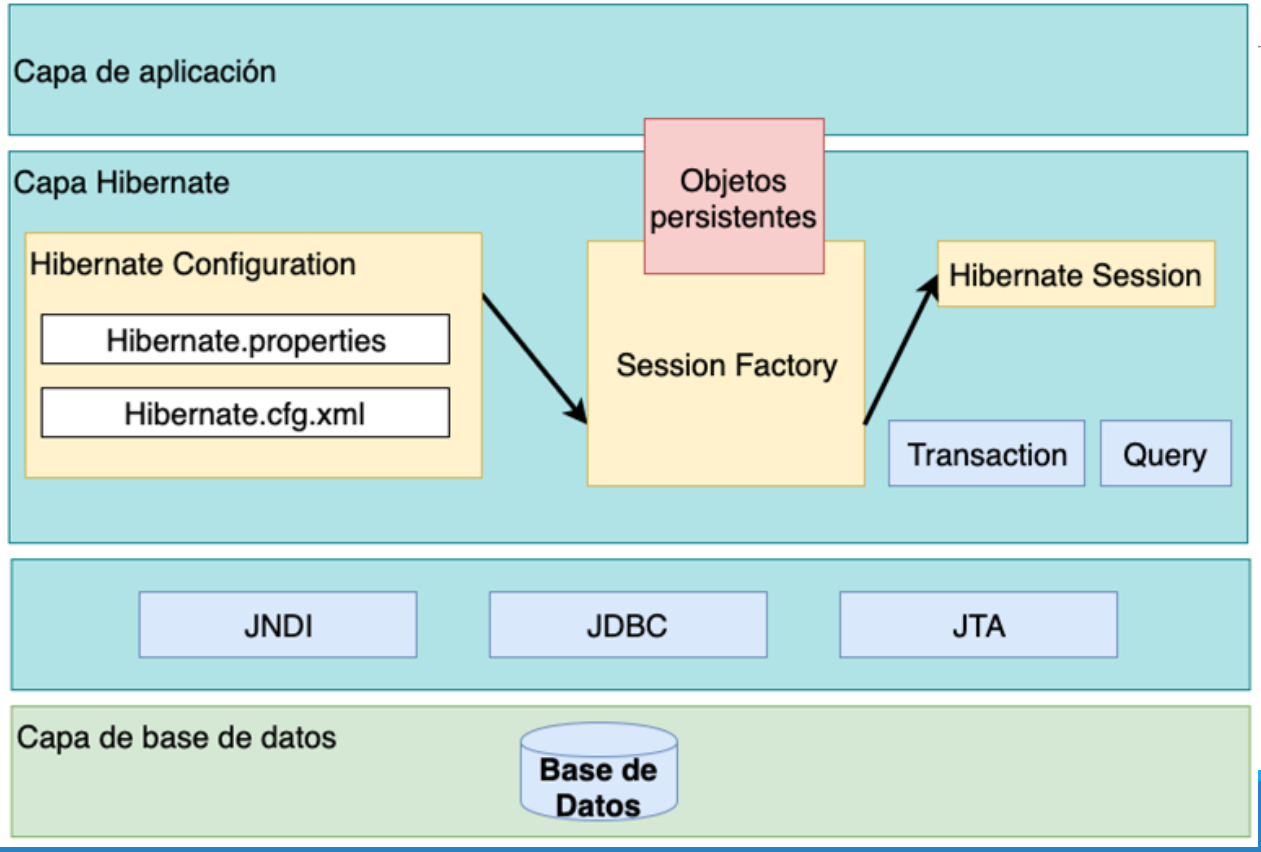
3. Herramientas ORM

- Con Hibernate no emplearemos habitualmente SQL para acceder a los datos, sino que el propio motor de Hibernate, mediante el uso del factorías (Patrón de diseño Factory) y otros elementos de programación construirá estas consultas para nosotros.
- Hibernate pone a disposición del programador un lenguaje llamado **HQL** (Hibernate Query Language) que permite acceder a los datos mediante POO.

4. Arquitectura de Hibernate

- Hibernate parte de una filosofía de mapear objetos Java normales, más conocidos como "**POJO**s" (Plain Old Java Object).
- Para almacenar y recuperar estos objetos de la base de datos, el desarrollador debe mantener una conversación con el motor de Hibernate mediante un especial que es la **sesión** (clase **Session**) que podríamos equiparar al concepto de conexión (connection) en JDBC.
- Del mismo modo que sucede en las conexiones JDBC hemos de crear y cerrar la sesión cuando ya no sea necesaria.
- En la siguiente figura podemos apreciar la arquitectura de una aplicación Hibernate estándar.

4. Arquitectura de Hibernate



4. Arquitectura de Hibernate

- La clase **Session** (**org.hibernate.Session**) ofrece métodos como **save(Object object)**, **createQuery(String consulta)**, **beginTransaction()**, **close()**, etc. para interactuar con la BBDD de forma similar a como lo haríamos con una conexión JDBC, con la salvedad de que resulta más simple:
  - Ejemplo: Para guardar un objeto simplemente invocaremos al método **session.save(miObjeto)**, sin necesidad de especificar una sentencia SQL.
- Una instancia de session apenas consume memoria y su creación y destrucción son muy baratas (en términos computacionales).
  - Esto es importante ya que, por la propia idiosincrasia de Hibernate, a lo largo de una ejecución se crean y se destruyen infinidad de sesiones.
- Podemos entender una sesión como una caché o una colección de objetos cargados (a o desde la base de datos) relacionados con una única unidad de trabajo.

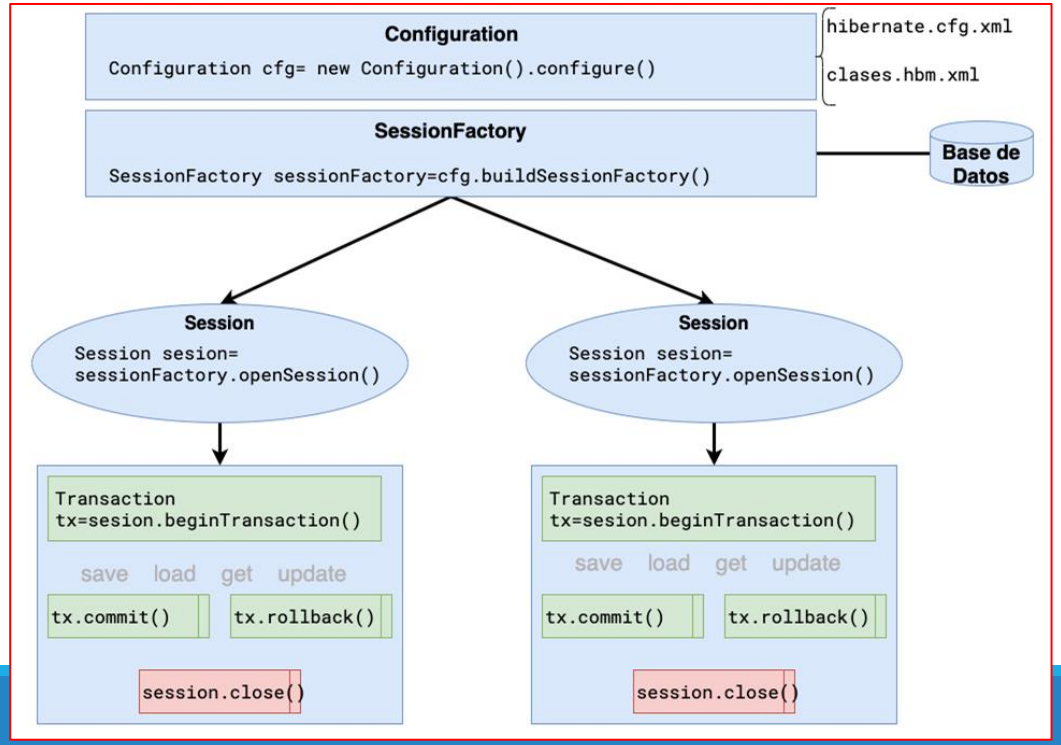
4. Arquitectura de Hibernate

- Las interfaces de hibernate son las siguientes:
  - **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 propiedades específicas de Hibernate, y a continuación se crea la **SessionFactory**.
  - **Query (org.hibernate.Query)** → permite realizar consultas en la base de datos y controla como se ejecutan dichas consultas. Las consultas se escriben en HQL o en el dialecto SQL nativo del SGBD que estemos usando. Una instancia de **Query** se utiliza para enlazar los parámetros de la consulta, limitar el número de resultados devueltos y para ejecutar dicha consulta.
  - **Transaction (org.hibernate.Transaction)** → nos permite asegurar que cualquier error que ocurra entre el inicio y el final de una transacción produzca el fallo de la misma (Rollback)

4. Arquitectura de Hibernate

- Hibernate hace uso de APIs de Java tales como JDBC, JTA (Java Transaction API) y JNDI (Java Naming Directory Interface).

En la figura de la derecha podemos ver la estructura de una aplicación Hibernate junto con las interfaces que emplearemos.



5. Instalación y configuración Hibernate

Si queremos trabajar con Hibernate, tenemos que incorporar a nuestro proyecto Java una serie de librerías, que se pueden descargar fácilmente desde la web http://hibernate.org/orm/downloads. Sin embargo, el mantenimiento de estas librerías puede llevarnos al conocido como infierno de los Jar, ya que podemos tener algunos problemas:

- Incluir dos versiones diferentes de una misma biblioteca.
- Dos bibliotecas que están relacionadas entre sí, y de las que se incluyen versiones incompatibles.
- …

Por todas estas razones, entre otras, es razonable utilizar algún sistema que nos permita gestionar las dependencias de nuestro proyecto con librerías externas. En nuestro caso, usaremos Maven.

5. Instalación y configuración Hibernate

Utilizando Maven, nuestro proyecto tendrá un fichero, llamado **pom.xml** que, entre otros elementos, nos permitirá definir las librerías que necesitamos.

De esa forma, podremos añadir las dependencias de nuestras librerías. Estas pueden ser encontradas en la web del repositorio de Maven (https://mvnrepository.com).

Alguna de las que usaremos durante el curso es la siguiente:

```
<dependency>
	<groupId>org.hibernate</groupId>
	<artifactId>hibernate-core</artifactId>
	<version>5.2.9.Final</version>
</dependency>
```

5. Instalación y configuración Hibernate

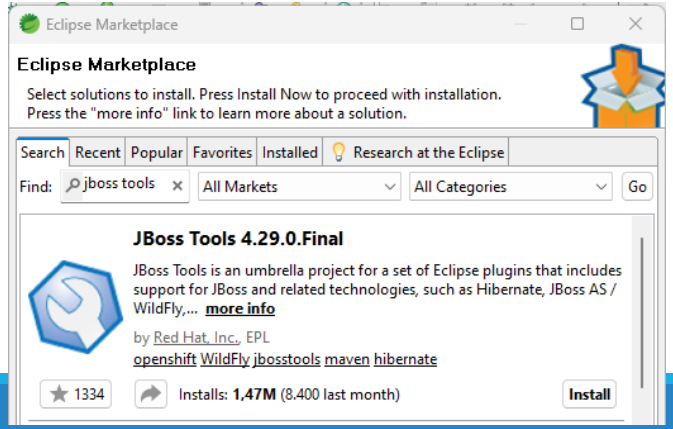
Para poder comenzar con nuestro primer proyecto con Hibernate, debemos tener en cuenta los siguientes prerrequisitos, con respecto al software que vamos a utilizar.

- Como herramienta fundamental de desarrollo vamos a usar una versión *tuneada* de Eclipse, llamada Spring Tool Suite. Hemos escogido esta versión por varias razones, como por ejemplo que nos permite hacer lo mismo que cualquier versión más elemental de eclipse, como la Neon; y que además nos permitirá utilizar más adelante Spring, para implementar proyectos Web MVC con Hibernate y JPA.

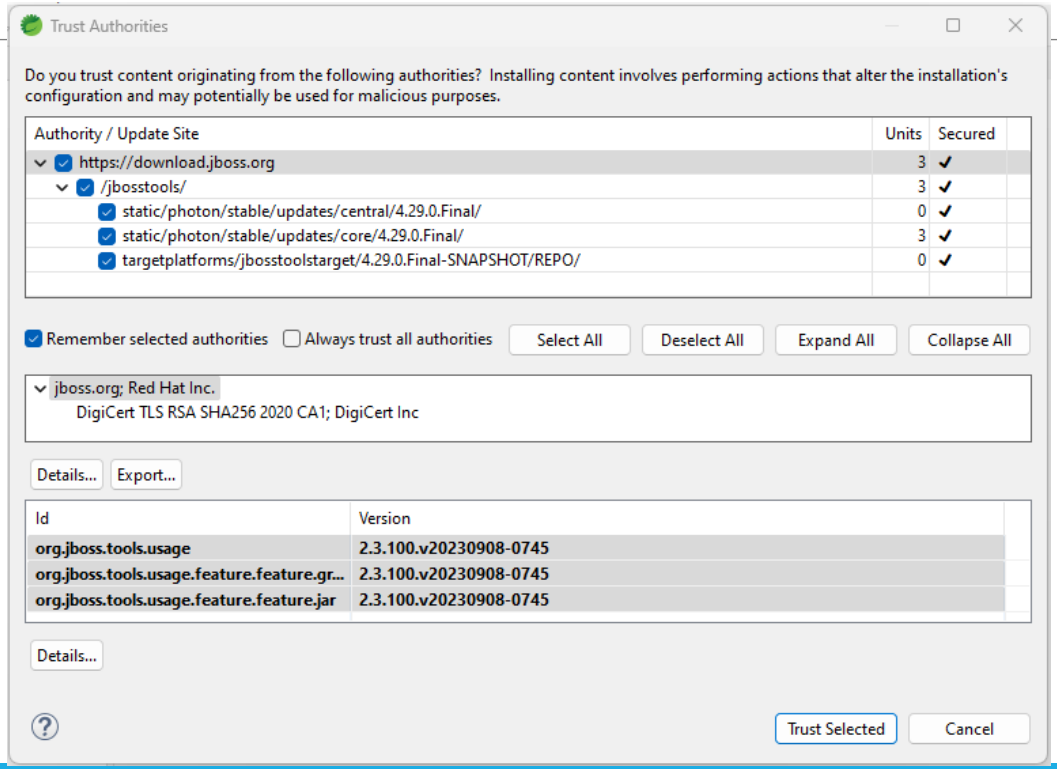
- Instalación de Spring Tool Suite: Spring Tools 4 ready-to-use distribution package

5. Instalación y configuración Hibernate

- Añadiremos a Eclipse un conjunto de herramientas, llamadas **Hibernate Tools**. Se trata de un conjunto de asistentes y vistas que nos permitirán acelerar y facilitar el desarrollo de nuestras aplicaciones con Hibernate. Para instalarlo, una vez abierto STS, nos dirigimos a **Help→Eclipse Marketplace**:
- En el buscador escribimos: JBoss Tools, seleccionamos la opción JBoss Tools, y pulsamos sobre el botón **Install**.



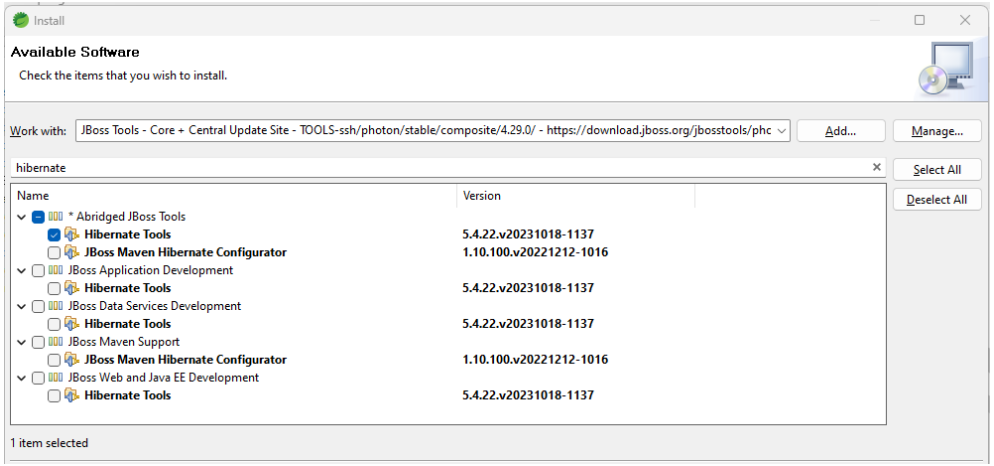
5. Instalación y configuración Hibernate



5. Instalación y configuración Hibernate

Una vez instalado el plugin de JBoss, instalamos la Hibernate Tools.

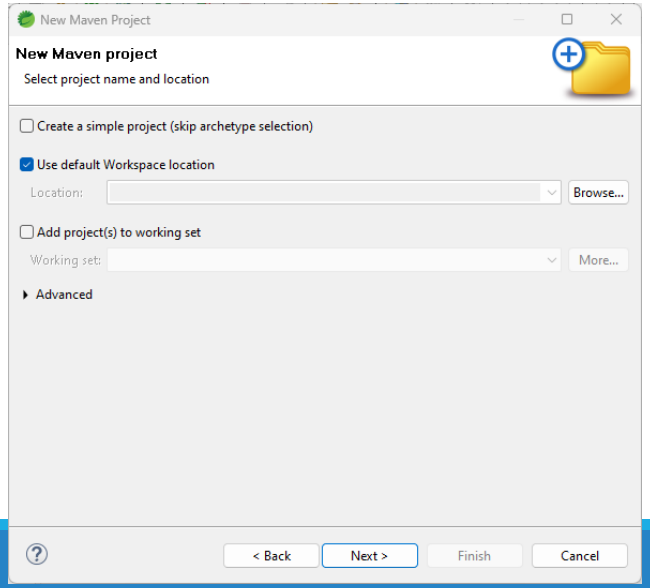
Help → Install New Software → Work with: “jboss” → Filtramos por Hibernate → Seleccionamos Hibernate Tools



6. Primer proyecto con Hibernate

Vamos a comenzar creando nuestro proyecto Maven, a través de la siguiente ruta

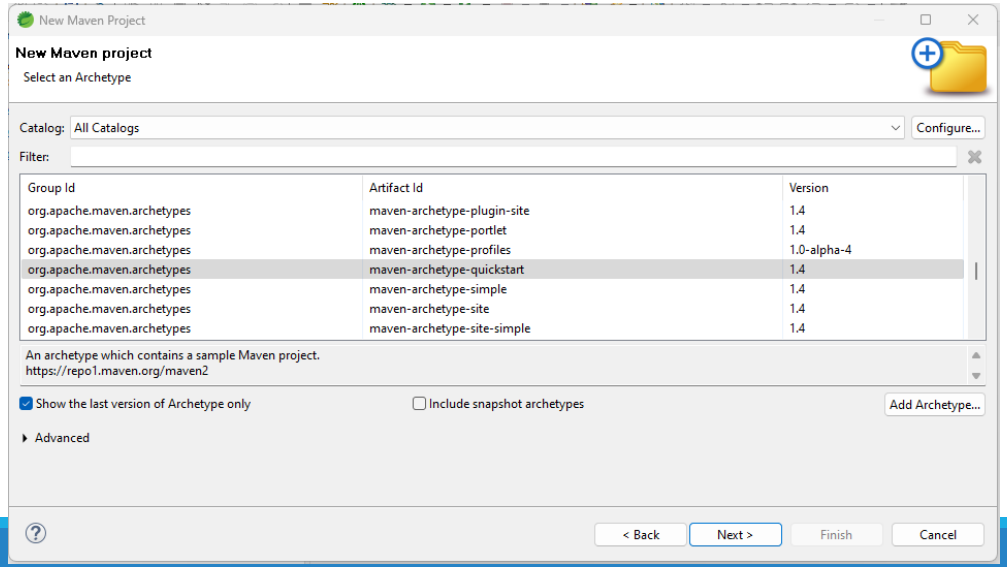
File → New → Other → Maven → Maven Project



6. Primer proyecto con Hibernate

Seleccionamos como arquetipo:

`org.apache.maven.archetypes maven-archetype-quickstart 1.4`



6. Primer proyecto con Hibernate

Se trata de la plantilla de proyecto más elemental.

En la siguiente pantalla, los datos serán:

- GroupId: com.iesvjp.hibernate (o algo similar)
- ArtifacId: PrimerEjemploHibernate (o algo similar)
- Package: com.iesvjp.hibernate.primerejemplohibernate (se trata de la concatenación de groupId y artifactId).

6. Primer proyecto con Hibernate

Ahora vamos a añadir las dependencias necesarias al fichero pom.xml, de forma que el apartado de dependencias queda de la siguiente forma:

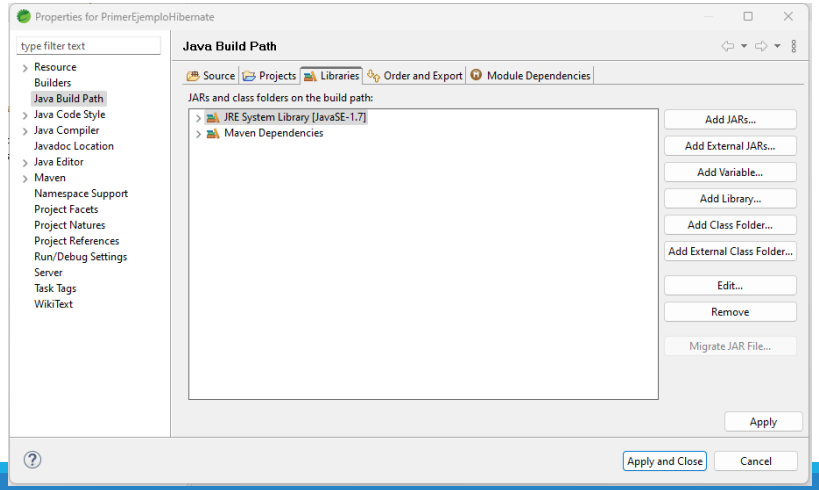
```
<dependencies>
	<!-- Otras dependencias, como la de jUnit... -->
	<dependency>
		<groupId>org.hibernate</groupId>
		<artifactId>hibernate-core</artifactId>
		<version>5.2.9.Final</version>
	</dependency>
	<dependency>
		<groupId>mysql</groupId>
		<artifactId>mysql-connector-java</artifactId>
		<version>8.0.20</version>
	</dependency>
</dependencies>
```

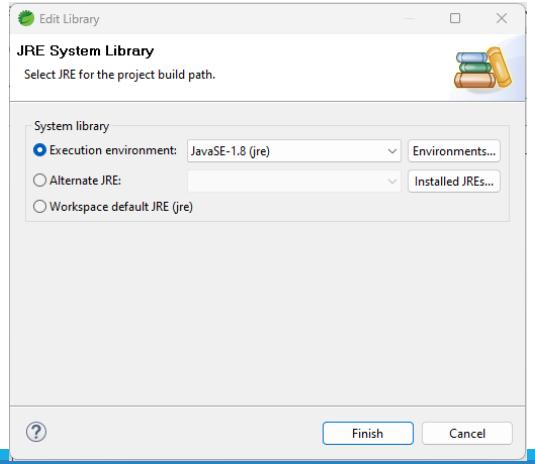
La primera dependencia es la que incluye de manera efectiva a Hibernate, entre otros módulos. La segunda es la que incluye las clases JDBC para conectar a Java con Mysql, y que son usadas por Hibernate.

6. Primer proyecto con Hibernate

Lo siguiente es solucionar el warning, provocado por el arquetipo elegido, cambiando en el Build Path la versión de java, a la 1.8.

Click derecho sobre el proyecto → Build path → Configure Build Path

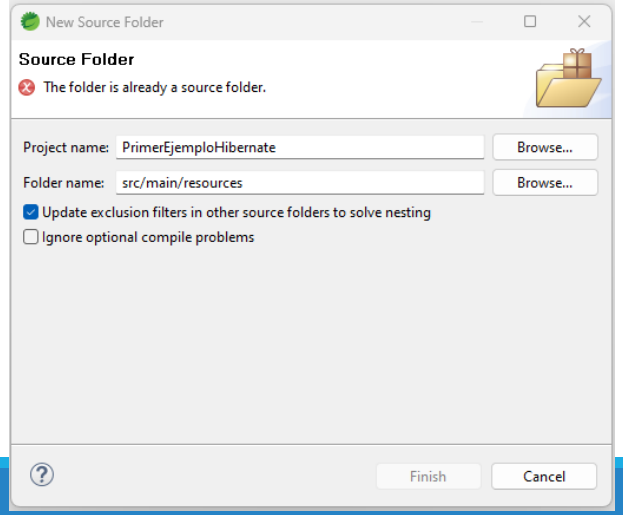




6. Primer proyecto con Hibernate

A continuación, creamos una carpeta de recursos, en la ruta */src/main/resources*. Para ello, pulsamos sobre el proyecto con el botón derecho y *New→Source Folder*. Indicamos la ruta completa, y marcamos la opción de *Update exclusion filters*…

NOTA: Puede que al cambiar la versión de Java se cree automáticamente



6. Primer proyecto con Hibernate

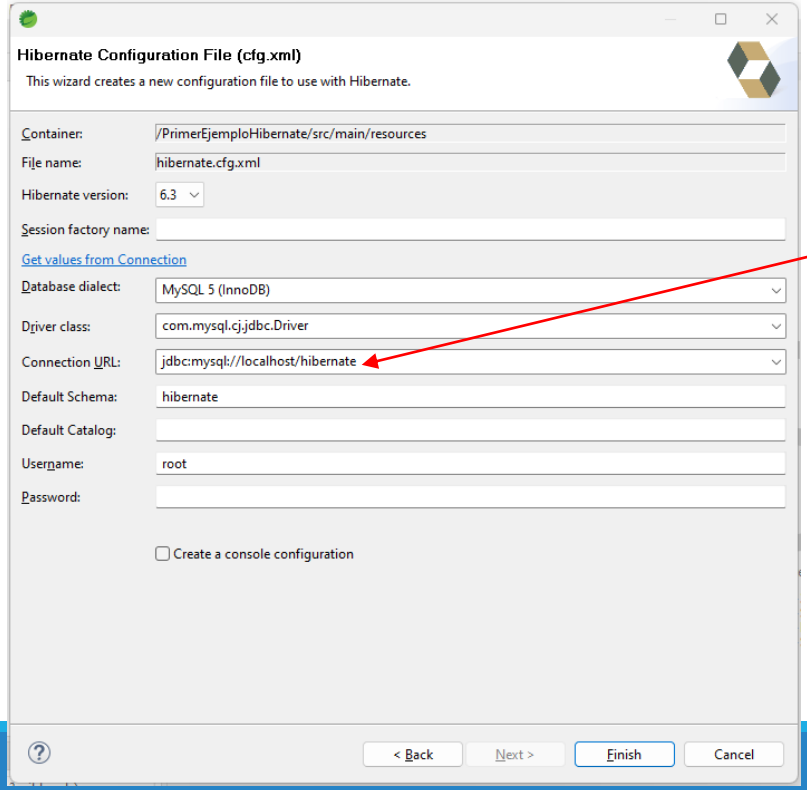
Ahora, sobre la nueva carpeta creada, volvemos a pulsar con el botón derecho y *New* → *Other* …. En el asistente, buscamos la carpeta *Hibernate* y seleccionamos la opción *Hibernate Configuration File (cfg.xml).* Pulsamos siguiente, y nombramos al fichero *hibernate.cfg.xml*.

El asistente nos facilitará mucho el trabajo, aunque es posible que luego tengamos que editar a mano alguna propiedad concreta.

Rellenamos los siguientes datos (o los que correspondan en nuestro caso):

Recuerda crearte la base de datos hibernate en MySQL.

6. Primer proyecto con Hibernate



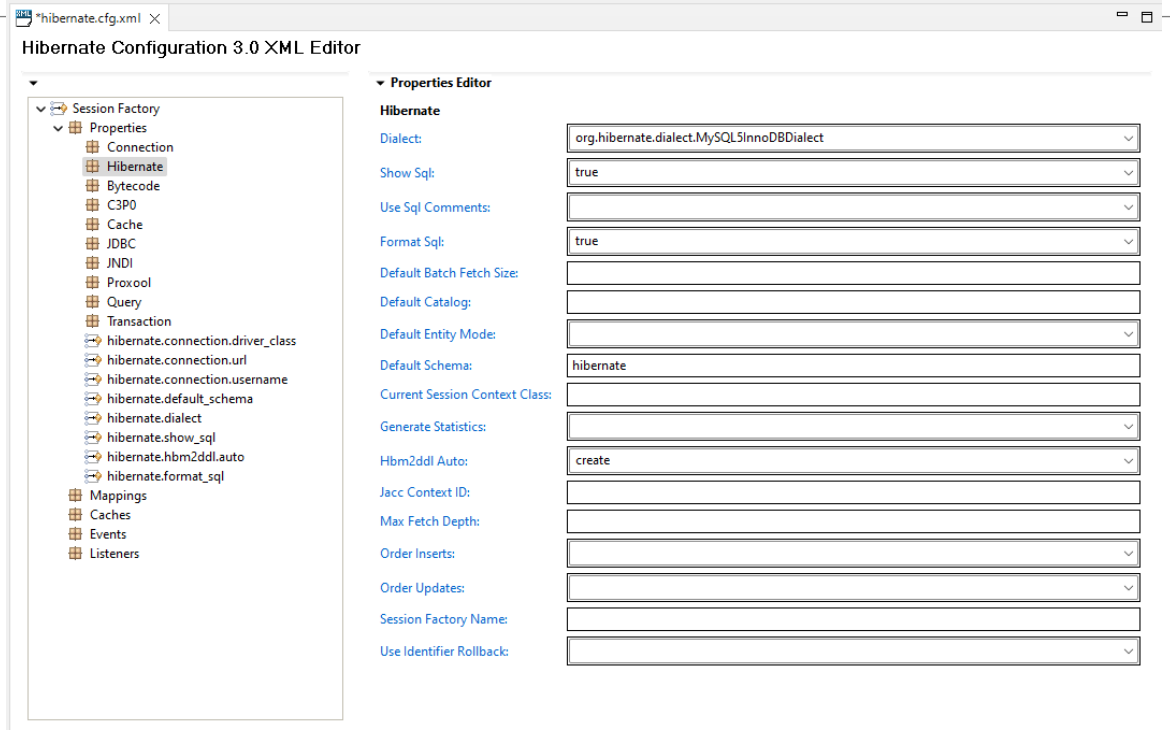
"hibernate":
Base de datos ya creada (recuerda crear una para realizar el ejemplo)

6. Primer proyecto con Hibernate

- Una vez creado el fichero, nos aparece el editor del mismo. Nos situamos en la opción *Properties→Hibernate*, y establecemos las propiedades *Show sql* y *Format sql* a true, así con *Hbm2ddl Auto* a *create*. Esta última propiedad es la encargada de generar las tablas necesarias para albergar nuestro modelo, sin necesidad de que nosotros lo tengamos que hacer manualmente.

- Aunque la generación automática del esquema es muy útil para el testeo o el prototipado, en un entorno de producción es menos usual, siendo más flexible manejar el esquema a través de scripts sql incrementales.

6. Primer proyecto con Hibernate



6. Primer proyecto con Hibernate

A nivel de configuración, tenemos una propiedad JPA, llamada hibernate.hbm2ddl.auto, que nos permite generar el DDL para un buen número de opciones:

Opción	Propósito
none	Valor por defecto. No realiza ninguna acción
`create-only`	Solamente realiza el proceso de creación de la base de datos.
`drop`	Realiza solamente el borrado de la base de datos.
`create`	Realiza un borrado de la base de datos, y posteriormente su creación.
`create-drop`	Elimina el esquema y lo vuelve a crear, al crear el contexto de persistencia o el SessionFactory. Adicionalmente, también lo elimina cuando uno u otro se cierran.
`validate`	Valida el esquema de la base de datos
`update`	Actualiza la base de datos, con los cambios necesarios.

6. Primer proyecto con Hibernate

Vamos a utilizar la clase User (disponible en el Moodle).

Aunque estamos trabajando con Hibernate nativo, usamos las anotaciones de JPA (entre otras cosas, algunas anotaciones de Hibernate ya están deprecated/obsoletas).

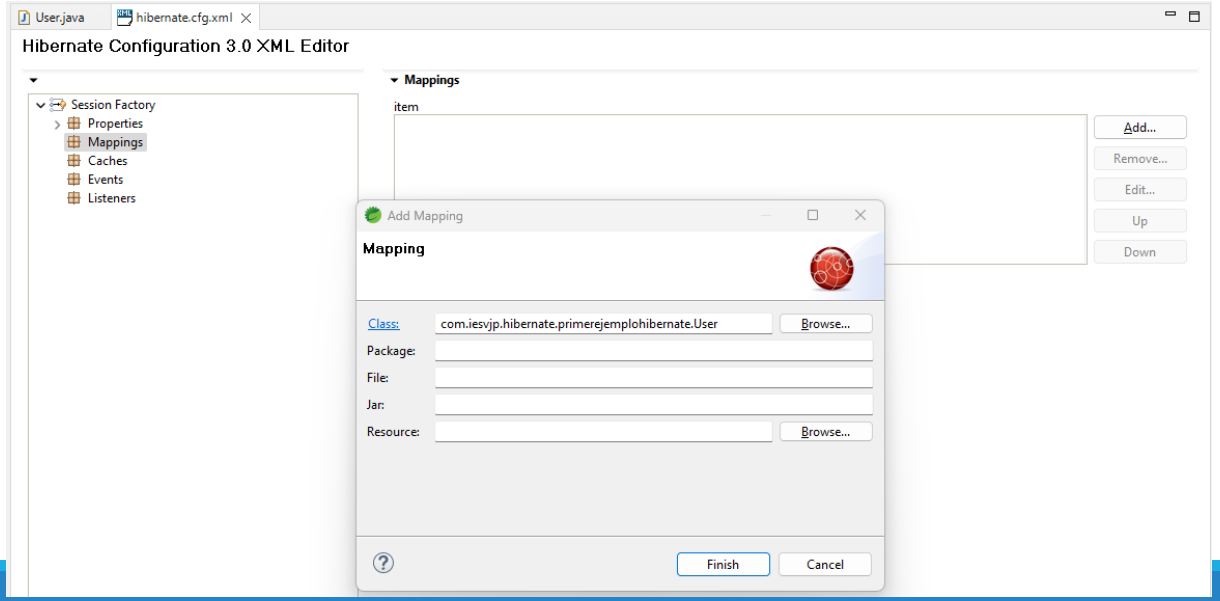
- @Entity indica que esta clase es una entidad que deberá ser gestionada por el motor de persistencia de Hibernate.
- @Id indica que, de todos los atributos, ese será tratado como clave primaria.
- @Column indica que ese atributo es una columna de la tabla resultante.

6. Primer proyecto con Hibernate

```java
package com.iesvjp.hibernate.primerejemplohibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class User {
    @Id
    private int id;
    @Column
    private String userName;
    @Column
    private String userMessage;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getUserMessage() {
        return userMessage;
    }

    public void setUserMessage(String userMessage) {
        this.userMessage = userMessage;
    }

}
```

6. Primer proyecto con Hibernate

Una vez creada nuestra entidad User deberemos incluirla en nuestro fichero de configuración de hibernate. Seleccionamos la opción de **Mappings** y pulsamos **Add**…



6. Primer proyecto con Hibernate

Si nos vamos a “Source” del archivo de configuración hibernate.cfg.xml, deberíamos encontrar algo así:

User.java | hibernate.cfg.xml

```
-//Hibernate/Hibernate Configuration DTD 3.0//EN (doctype with catalog)
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
3                                           "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
4  <hibernate-configuration>
5   <session-factory name="">
6    <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
7    <property name="hibernate.connection.url">jdbc:mysql://localhost/hibernate</property>
8    <property name="hibernate.connection.username">root</property>
9    <property name="hibernate.default_schema">hibernate</property>
10   <property name="hibernate.dialect">org.hibernate.dialect.MySQL5InnoDBDialect</property>
11   <property name="hibernate.show_sql">true</property>
12   <property name="hibernate.hbm2ddl.auto">create</property>
13   <property name="hibernate.format_sql">true</property>
14   <mapping class="com.iesvjp.hibernate.primerejemplohibernate.User"/>
15  </session-factory>
16 </hibernate-configuration>
17
```

6. Primer proyecto con Hibernate

Por último, nos falta implementar el código necesario para cargar la configuración del fichero *hibernate.cfg.xml* y realizar las operaciones necesarias. En nuestro caso, insertar dos nuevas entidades en la base de datos.

También tenéis la clase App.java disponible en el Moodle.

```java
package com.iesvjp.hibernate.primerejemplohibernate;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class App {
    public static void main(String[] args) {
        // Inicialización del SessionFactory
        StandardServiceRegistry sr = new StandardServiceRegistryBuilder().configure().build();
        SessionFactory sf = new MetadataSources(sr).buildMetadata().buildSessionFactory();

        /*
         * //Codigo "legacy" SessionFactory sf = new Configuration() .configure()
         * .buildSessionFactory();
         */
        // Apertura de una sesión (e inicio de una transacción)
        Session session = sf.openSession();

        User user1 = new User();
        user1.setId(1);
        user1.setUserName("Pepe");
        user1.setUserMessage("Hello world from Pepe");

        User user2 = new User();
        user2.setId(2);
        user2.setUserName("Juan");
        user2.setUserMessage("Hello world from Juan");
        session.beginTransaction();

        // Almacenamos los objetos
        session.save(user1);
        session.save(user2);

        // Commit de la transacción
        session.getTransaction().commit();

        // Cierre de la sesión
        session.close();
        sf.close();
    }
}
```

6. Primer proyecto con Hibernate

Antes de ejecutar el proyecto tenemos que instalar Maven, descargarnos todas las dependencias y actualizar Maven **(recordad que añadimos unas dependencias al fichero pom.xml**):

- Botón derecho encima del proyecto: Run As→Maven clean
- Run As→Maven install
- Maven→Update Project

Ten a mano antes de ejecutar el fichero de errores comunes (es probable que nos salga alguno la primera vez que ejecutamos).

6. Primer proyecto con Hibernate

Cuando todo funcione correctamente **se creará la tabla en la base de datos** y se insertarán los datos:

6. Primer proyecto con Hibernate

Como habéis podido comprobar al ejecutar el proyecto nos aparece mucha información de logs de Hibernate en color rojo. Si queremos que no se muestre esta información tendremos que añadir en el main() de la aplicación la siguiente línea:

```
java.util.logging.Logger.getLogger("org.hibernate").setLevel(Level.OFF);
```

Dudas y preguntas