1. Mapeo objeto-relacional

1.1. ¿Qué es el mapeo objeto-relacional?

Los lenguajes orientados a objetos son muy populares, aunque es cierto que utilizar un lenguaje orientado a objetos no significa necesariamente utilizar sus características orientadas a objetos. De hecho, muchas aplicaciones no lo hacen. Es decir, no utilizan objetos para representar los datos persistentes con los que trabajan, o bien no sacan partido de la herencia y otras posibilidades de estos lenguajes.

En cualquier caso, cada vez más se planteó la conveniencia o incluso la necesidad de almacenar objetos en bases de datos relacionales, incluyendo objetos complejos con referencias a otros objetos, a colecciones de objetos, etc. La persistencia de objetos en bases de datos relacionales plantea una serie de problemas, conocidos conjuntamente como desfase objeto-relacional. Para resolver o evitar estos problemas surgieron varios planteamientos, siendo uno de ellos el **mapeo objeto-relacional** (en inglés, *object-relational mapping* ó ***ORM***).

El ORM consiste en el establecimiento de una correspondencia entre clases definidas en un lenguaje de programación orientado a objetos, como Java, y tablas de una base de datos relacional, usando mecanismos para que las modificaciones sobre los objetos se registren en la base de datos y (de forma inversa) para que se puedan recuperar en objetos la información almacenada en la base de datos. Esto hace posible la persistencia de objetos en bases de datos puramente relacionales.

Algunas de las herramientas ORM más importantes son:

- En Java: Hibernate.
- En PHP: Doctrine o Propel.
- En Python: Django.

En esta unidad nos centraremos en el uso de la herramienta **Hibernate** (https://hibernate.org/).

1.2. Ventajas y desventajas

Entre las ventajas de utilizar una herramienta ORM encontramos:

- Nos fuerza a desarrollar código siguiendo el patrón MVC, lo que hace que nuestro código sea más limpio.
- No tenemos que escribir sentencias SQL.
- Abstrae nuestra aplicación del sistema de almacenamiento de datos y del modelo, por lo que podríamos cambiarlo cuando quisiésemos.

- Favorece la reutilización y mantenimiento de código.

Algunas desventajas son:

- En proyectos grandes, es menos eficiente que utilizar SQL puro.
- Dado su alto nivel de abstracción, al principio puede ser difícil saber el motivo por el que se produce un error.

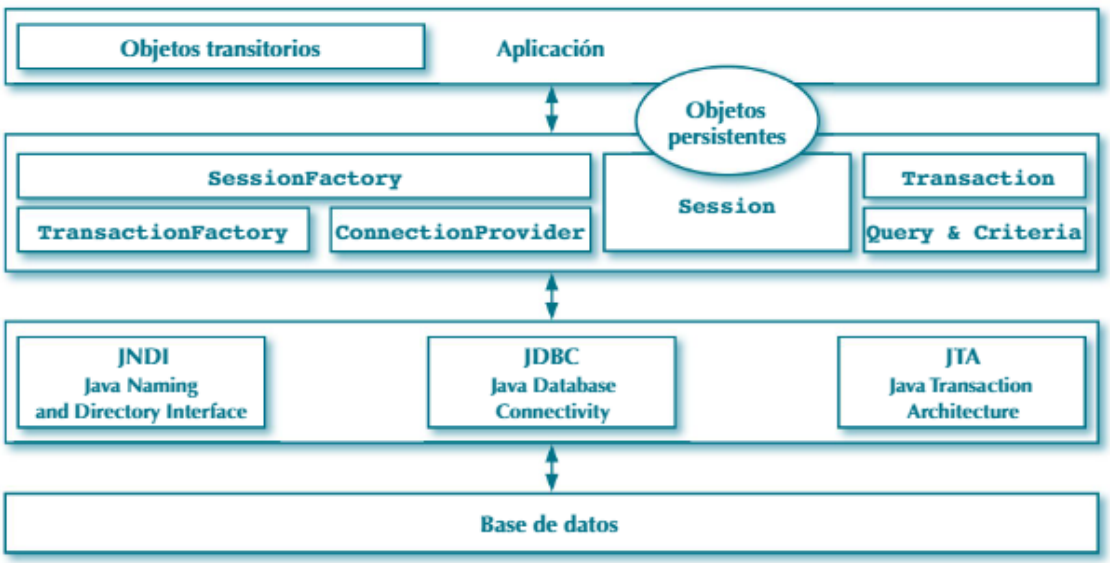
1.3. Hibernate

Hibernate es un framework de ORM para Java, distribuido bajo licencia LGPL 2.1 de GNU. Al estar distribuido bajo licencia GPL, se puede utilizar en aplicaciones comerciales sin necesidad de hacer público su código fuente.

La principal característica de Hibernate es que permite mapear clases de Java con las tablas de la BD y los tipos de datos de Java con los de SQL. Hibernate facilita la realización de consultas, ya que se encarga de hacer las llamadas SQL, evitando que esa tarea la tenga que hacer el desarrollador de forma manual.

Hibernate tiene su propio lenguaje para manejar objetos persistentes, llamado **HQL** (*Hibernate Query Language*), que está inspirado en SQL. La diferencia es que las sentencias SQL operan sobre filas y columnas de tablas, mientras que las de HQL lo hacen sobre conjuntos de objetos persistentes y sus atributos.

La siguiente imagen muestra la arquitectura de Hibernate y sus principales componentes:



Una aplicación que utiliza Hibernate trabaja con objetos tanto transitorios como persistentes. Los objetos persistentes están siempre asociados a una sesión (*Session*) creada por una *SessionFactory*. Una aplicación puede crear objetos transitorios y luego convertirlos en persistentes para que después se almacenen en la base de datos. Puede también recuperar un objeto persistente desde la base de datos y realizar cambios sobre él, para que después se reflejen en la base de datos. Puede agrupar operaciones sobre los objetos persistentes dentro de transacciones (*Transaction*) que están siempre asociadas a una sesión. Puede utilizar también sentencias HQL mediante la clase *Query*.

Hibernate interactúa con la base de datos mediante JDBC y puede utilizar internamente APIs de Java tales como JNDI (*Java Naming and Directory Interface*) para fuentes de datos (*DataSource*) y JTA (*Java Transaction Arquitecture*) para transacciones.

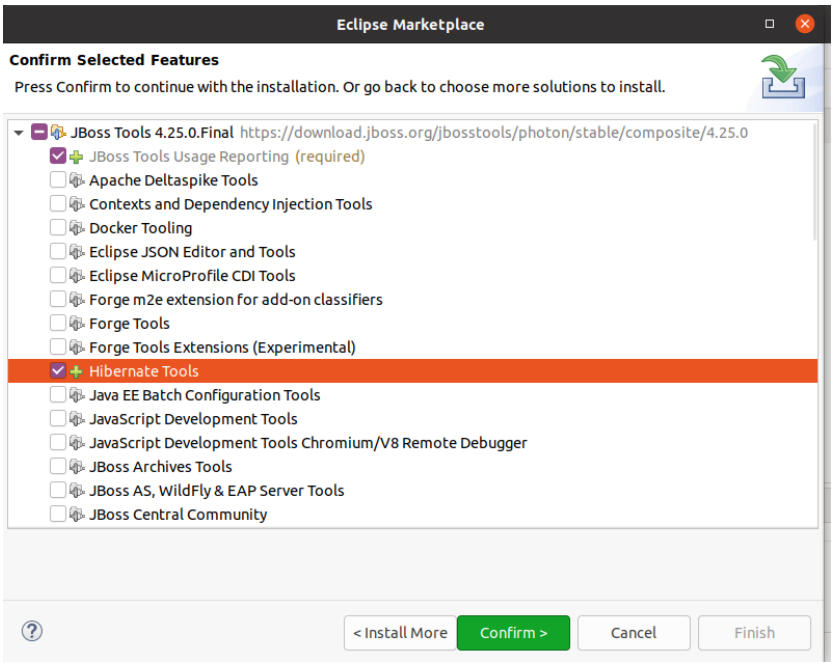
Entre las ventajas del *framework* Hibernate encontramos:

- Es una herramienta *open source*.
- Es una herramienta veloz debido a su uso de memoria caché.
- Al utilizar el lenguaje HQL, las consultas que hagamos son independientes de la base de datos que utilicemos.
- Crea las tablas en la base de datos automáticamente. No es necesario hacerlo de forma manual.
- Simplifica la composición de tablas (JOIN).

1.4. Instalación y configuración de Hibernate

Paso 1: instalación del plugin para Eclipse

Una vez abierto Eclipse, podemos instalar el plugin desde *Help -> Eclipse Marketplace…* y seleccionando el plugin **JBoss Tools**. Una vez seleccionado, marcaremos solo la opción de **Hibernate Tools**. A continuación, comenzará el proceso de descarga.



Reiniciamos Eclipse y comprobamos que esté instalado correctamente. Podemos hacerlo en *Window -> Show View -> Other* y comprobar que aparezcan las opciones de Hibernate.

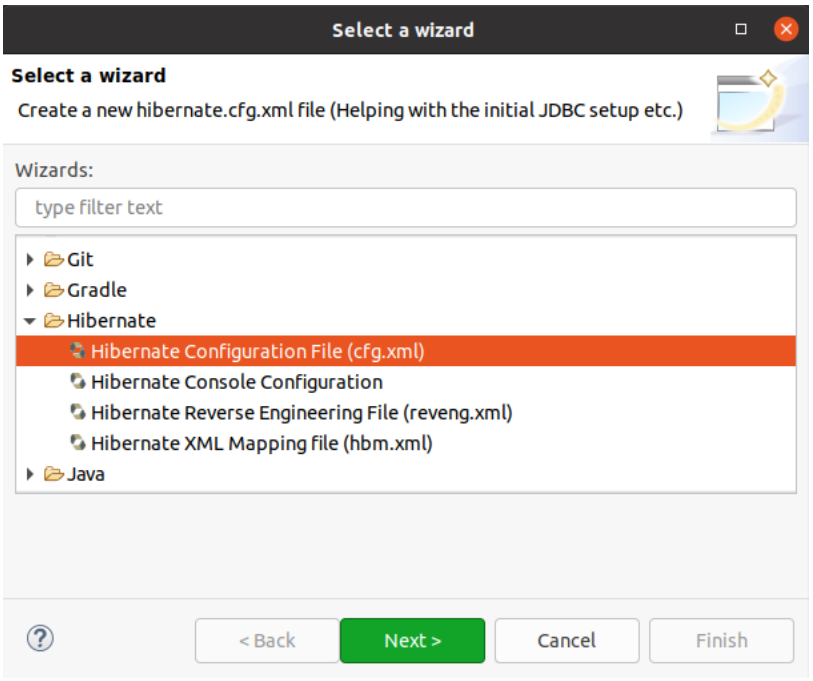
También lo podemos comprobar desde *Window -> Open Perspective -> Other -> Hibernate*.

Paso 2: crear un proyecto Java

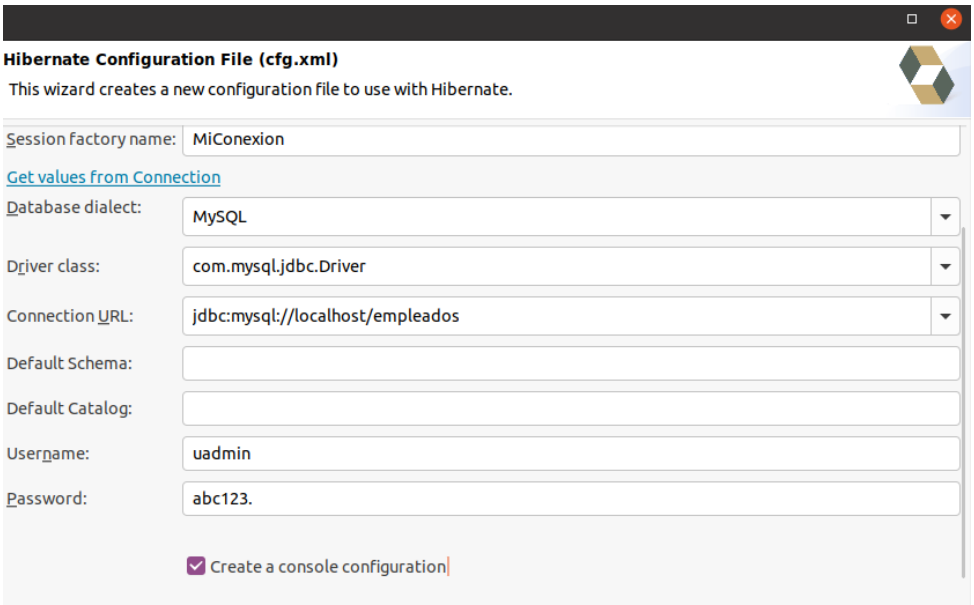
Una vez que ya tenemos instalado Hibernate, vamos a utilizarlo en un proyecto. Para eso, crearemos un nuevo proyecto Java, que en este caso le daremos el nombre *EjemploHibernate*. No debemos crear el fichero *module-info.java*, porque entrará en conflicto con Hibernate. En el proyecto tenemos que añadir el conector JDBC de MySQL para poder acceder a la base de datos, tal y como hacíamos en la unidad anterior.

Paso 3: crear el fichero de configuración de Hibernate

Una vez que tenemos el driver MySQL en nuestro proyecto hemos de crear el fichero de configuración de Hibernate ***hibernate.cfg.xml***. Para eso, haciendo click derecho sobre nuestro proyecto, vamos a *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.



Si vamos a *Next*, se nos pide dónde queremos crear el fichero. En este ejemplo lo crearemos en la carpeta *src* del proyecto. Pulsamos de nuevo *Next* y se nos pedirán los datos para poder realizar la conexión con la BD. En este apartado, seleccionaremos la versión 5.6 de Hibernate para que no haya conflicto entre versiones. Nos conectaremos a la BD *empleados* que utilizamos en la unidad anterior.



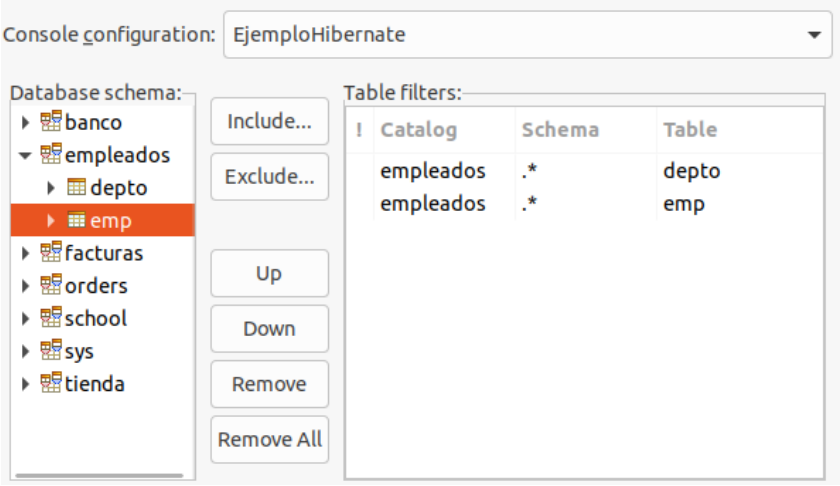
El campo *Session factory name* sirve para escribir el nombre de nuestra conexión MySQL. En *Database dialect* indicamos el SGBD que vamos a utilizar y en *Driver class* seleccionamos el tipo de conector que utilizaremos. Una vez tenemos completados los campos anteriores, le damos a finalizar.

Ahora ya tenemos creado el fichero de configuración y podemos visualizarlo en la carpeta *src* del proyecto utilizando la pestaña *Source*.

Paso 4: crear el fichero de ingeniería inversa

Después tenemos que crear el fichero XML *Hibernate Reverse Engineering (reveng.xml)*, que es el encargado de crear las clases de nuestras tablas MySQL. Para ello, hacemos click derecho sobre el proyecto y seleccionamos *New -> Other -> Hibernate -> Hibernate Reverse Engineering File (reveng.xml)*. Lo guardamos en la carpeta *src* del proyecto, al igual que hicimos con el fichero de configuración.

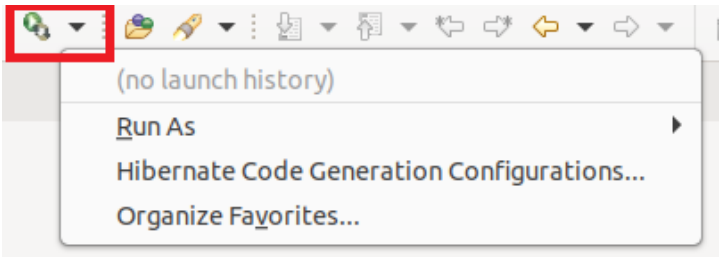
Después, pulsando *Next*, se visualizará una nueva ventana desde donde indicaremos las tablas que queremos mapear. Tenemos que seleccionar la *Console Configuration* e incluir las tablas de la BD *empleados*. Si no se nos muestra ninguna BD, simplemente tenemos que pulsar *Refresh*.



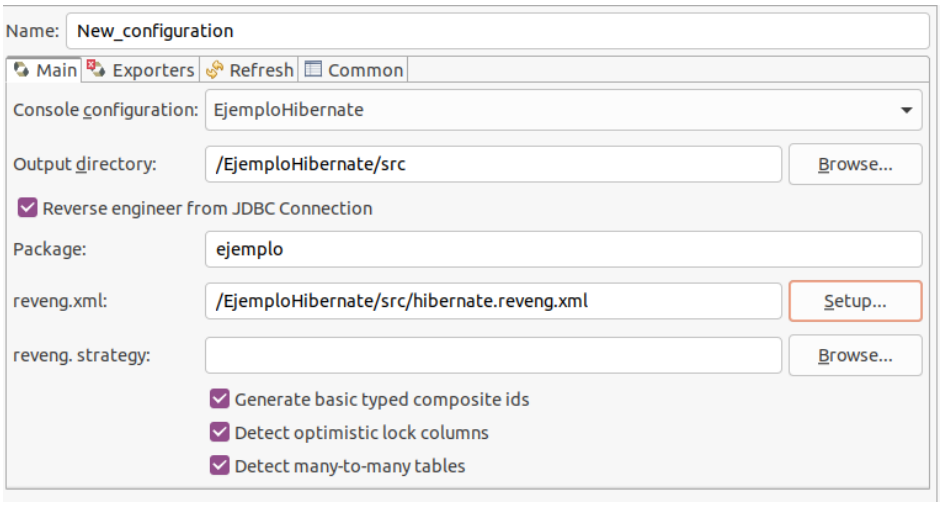
Una vez creado, al igual que con el fichero de configuración, podemos encontrarlo en la carpeta *src* y visualizar su contenido en la pestaña *Source*.

Paso 5: generar las clases de la base de datos

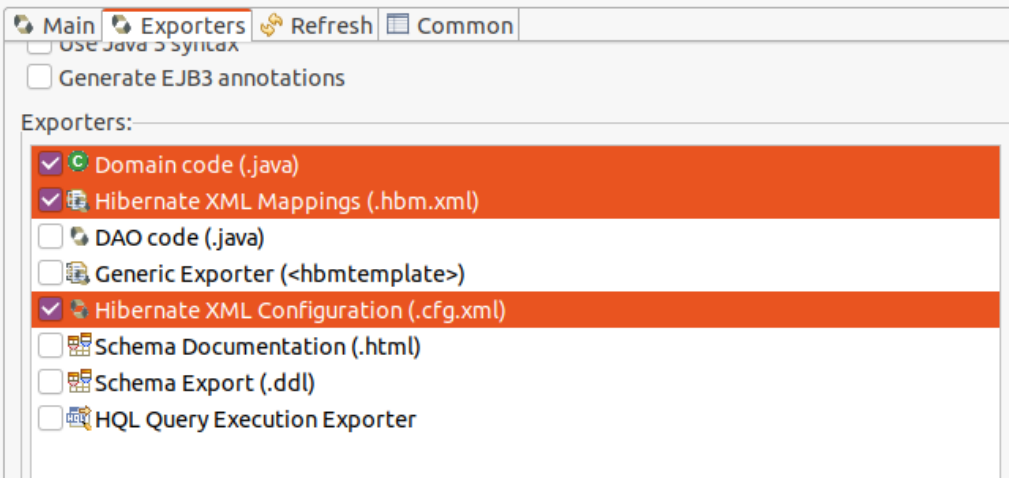
El siguiente paso es generar las clases de nuestra base de datos *empleados*. Para ello pulsamos la flecha situada a la derecha del botón *Run As* y seleccionamos *Hibernate Code Generation Configurations*:



Desde la ventana que aparece hacemos doble click sobre *Hibernate Code Generation Configurations* que aparece en el marco de la izquierda. Se abrirá una nueva ventana y la configuramos de la siguiente manera:



Después seleccionamos la pestaña *Exporters* y marcamos las siguientes casillas:



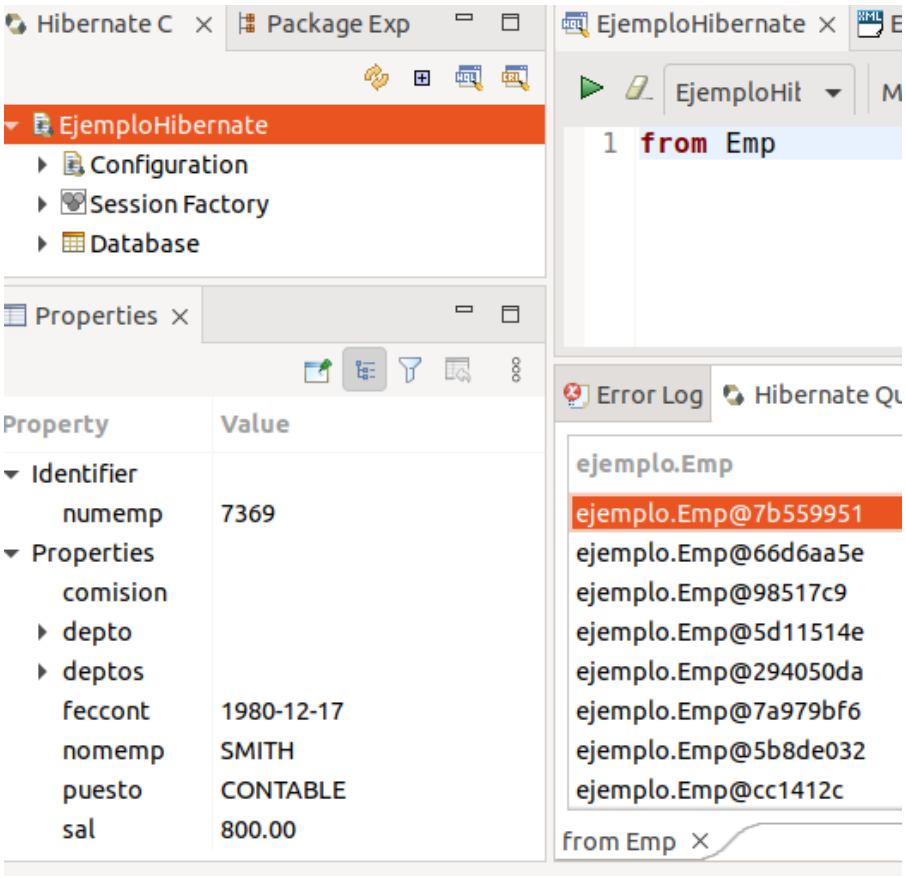
Aplicamos los cambios y pulsamos en *Run*. Veremos que se nos crea un paquete *ejemplo* con las clases Java para los empleados y departamentos. Estas clases contienen los constructores, *getters* y *setters* de cada campo de la tabla. También se generan unos ficheros XML que contienen información del mapeo de su respectiva tabla.

1.5. Consultas con HQL

Como se comentó anteriormente, HQL es el lenguaje que utiliza Hibernate para realizar consultas. Se puede consultar la documentación del lenguaje en el siguiente enlace:

https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/queryhql.html

Si queremos realizar consultas, podemos hacerlo desde *Window -> Perspective -> Open Perspective -> Other -> Hibernate*. Sobre nuestra configuración hacemos click derecho y seleccionamos *HQL Editor*. Un ejemplo de consulta sería:



Entre las principales características de HQL, podemos destacar:

- No podemos utilizar el *.
- Las consultas se hacen sobre los nombres de las clases, no de las tablas.
- Es *case sensitive* con el nombre de las clases.

Si seleccionamos *Mapping Diagram* podemos ver un diagrama de mapeo entre la base de datos y las clases de Java.

1.6. Desarrollo de una aplicación con Hibernate

Paso 1: descargar e importar las librerías de Hibernate

Descargamos la distribución de Hibernate en el siguiente enlace:

https://sourceforge.net/projects/hibernate/files/hibernate-orm/5.6.5.Final/

Después, descomprimimos el archivo y añadimos al proyecto los *.jar* del directorio */lib/required*.

Paso 2: creación del archivo HibernateUtil.java

El siguiente paso es crear una instancia de la base de datos para poder trabajar con ella. Esta instancia se puede utilizar a lo largo de toda la aplicación.

Para ello, crearemos una clase llamada *HibernateUtil* que seguirá el patrón *Singleton* (https://es.wikipedia.org/wiki/Singleton).

La clase será como la siguiente:

```
package ejemplo;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            return new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

La documentación de la API de Hibernate la podemos encontrar en:

https://docs.jboss.org/hibernate/orm/5.6/javadocs/

También podemos encontrar otros recursos en:

https://hibernate.org/orm/documentation/5.6/

Paso 3: desarrollo del programa principal

Podemos crear un nuevo paquete (que se llame “app”, por ejemplo) y que contenga el programa principal de nuestra aplicación. Para este ejemplo, crearemos una clase *AñadirDepartamento*, en la que insertaremos un nuevo departamento en nuestra base de datos.

```
3  import org.hibernate.Session;
9
10 public class AñadirDepartamento {
11
12     public static void main(String[] args) {
13         SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
14         Session session = sessionFactory.openSession();
15         Transaction tx = session.beginTransaction();
16
17         System.out.println("Insert new DEPTO...");
18
19         Depto depto = new Depto();
20         depto.setNumdep(10);
21         depto.setNomdep("I+D");
22         depto.setLocalidad("LUGO");
23
24         session.save(depto);
25
26         tx.commit();
27         System.out.println("DEPTO INSERTED!");
28         session.close();
29     }
30 }
```

Si ejecutamos el programa podremos comprobar cómo se insertó el nuevo departamento en la base de datos.