Implantación con ejemplos prácticos de HIBERNATE como herramienta de mapeo objeto relacional

Hibernate es un framework de mapeo Objeto Relacional (ORM) open source. Una de las principales características de Hibernate es que permite tener persistencia con los datos transforman modelos de diseño de datos a formato XML de acuerdo a los DTD definidos por Hibernate.

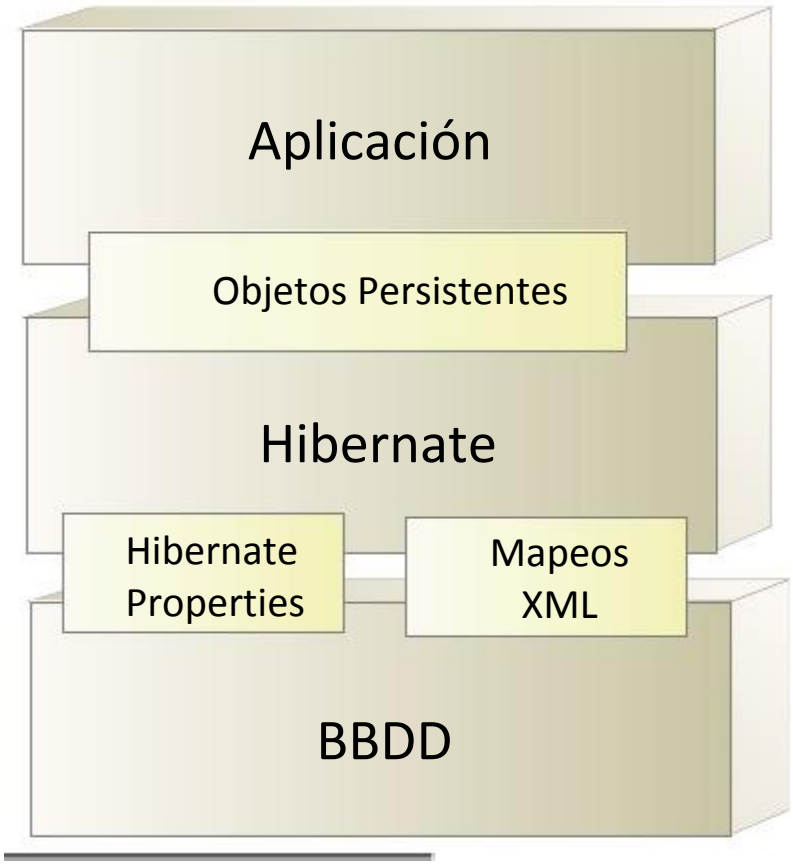


Diagrama de Bloques Hibernate

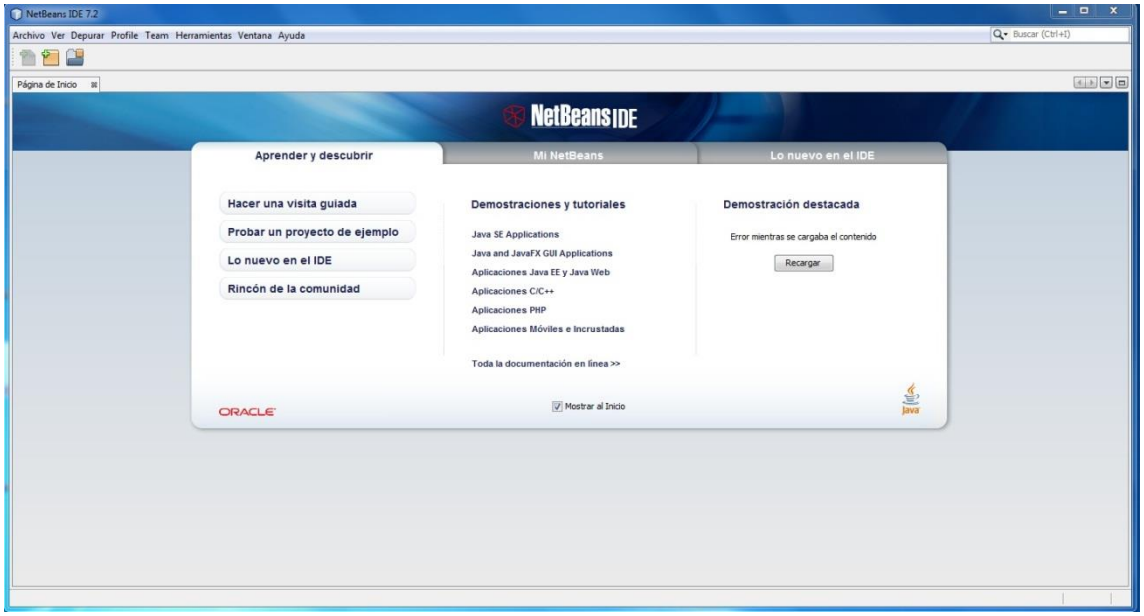
Hibernate es un framework que se ubica en la capa de persistencia objeto/relacional y es un generador de sentencias SQL, permite manejar objetos persistentes, que podrán incluir características tales como: polimorfismo, colecciones, relaciones, y múltiples tipos de datos.

De manera muy optimizada y rápida se pueden generar mapeos Objeto Relacionales en cualquiera de los entornos soportados (tiene soporte para todos los sistemas gestores de bases de datos y se integra de forma elegante y sin restricción alguna con los más conocidos contenedores web).

Hibernate es la solución ORM más conocida en las aplicaciones Java. También tiene una forma muy elegante para acceder a la información de las bases de datos relacionales, esto es gracias a su lenguaje de consultas HQL (Hibernate Query Language), el cual tiene un diseño como una mínima extensión orientada a objetos de SQL. Hibernate también permite realizar consultas basadas en criterios o en SQL puro.

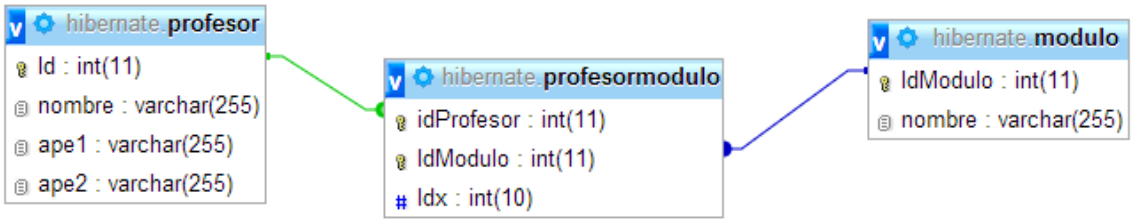
Implantación

Para los ejemplos de los Framework's ORM de Java, emplearemos **NetBeans** como IDE.

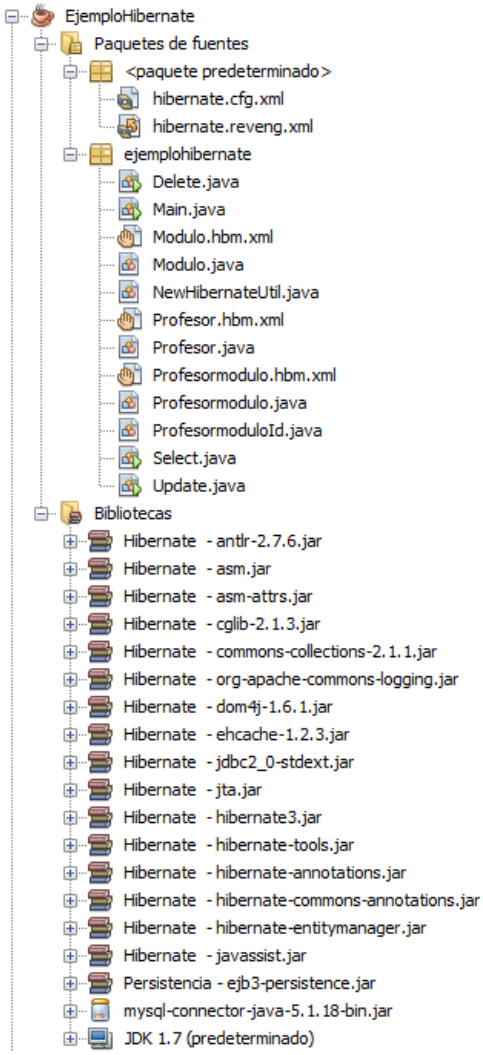


Para conocer más acerca de Hibernate, vamos a ver un pequeño ejemplo.

Partimos de cualquier estructura de una BD, en nuestro ejemplo usamos XAMPP (MySQL) y la estructura de datos que se muestra.



Vamos a ver la estructura completa del proyecto.



En ella, podemos distinguir tres grupos de archivos:

- Un paquete predeterminado, donde se coloca el fichero XML de configuración de Hibernate.
- Un carpeta con el nombre del proyecto (EjemploHiberntate, en nuestro caso), donde se sitúan los ficheros con los ficheros de mapeo y clases correspondientes a las tablas de la BD
- Por último, una biblioteca, con todas las referencias necesarias para el correcto funcionamiento

A continuación, vamos a conocer como se generan todos estos ficheros.

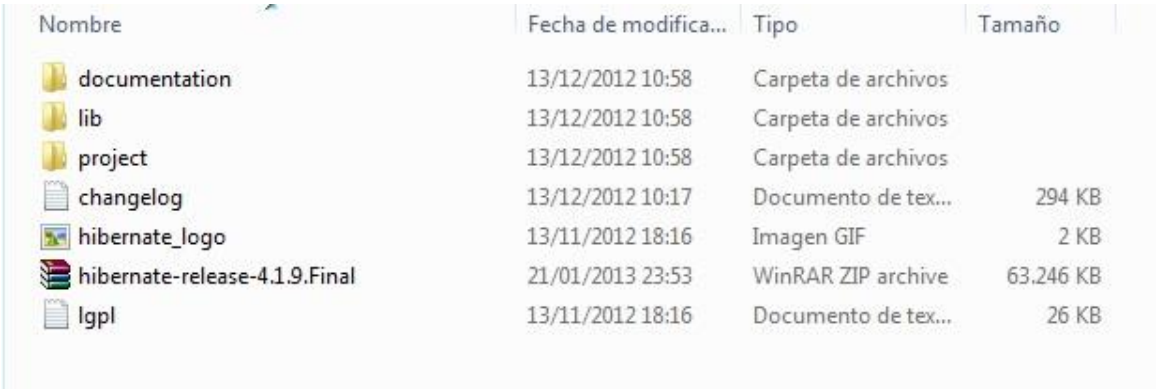
Para instalar Hibernate, lo primero que es necesario hacer es descargarlo y descomprimirlo.

Paso 1: Navegar a Downloads - Hibernate - JBoss Community

Paso 2: Pinchar en el enlace release bundles

Paso 3: Descargar la versión más reciente.

Paso 4: Una vez descargado el fichero, deberemos descomprimirlo. El resultado es el siguiente:

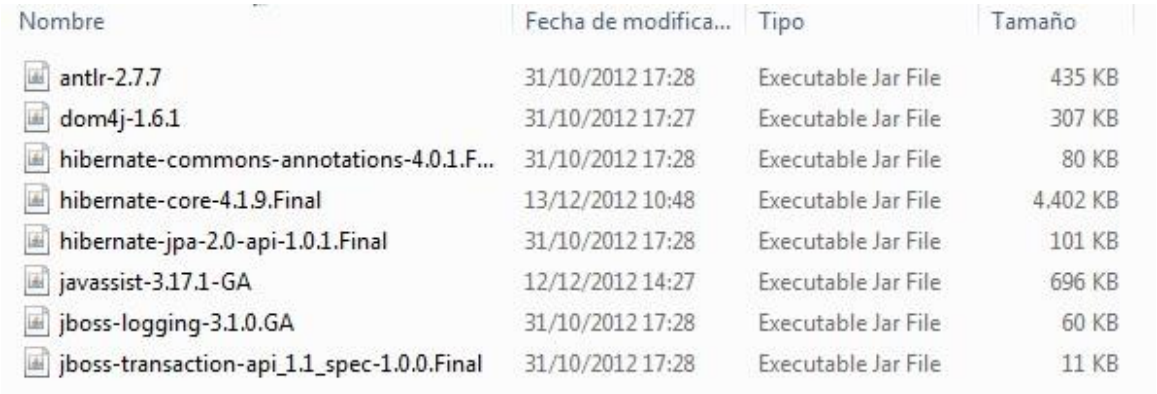


Nombre	Fecha de modifica...	Tipo	Tamaño
documentation	13/12/2012 10:58	Carpeta de archivos	
lib	13/12/2012 10:58	Carpeta de archivos	
project	13/12/2012 10:58	Carpeta de archivos	
changelog	13/12/2012 10:17	Documento de tex...	294 KB
hibernate_logo	13/11/2012 18:16	Imagen GIF	2 KB
hibernate-release-4.1.9.Final	21/01/2013 23:53	WinRAR ZIP archive	63.246 KB
lgpl	13/11/2012 18:16	Documento de tex...	26 KB

En el fichero se encuentran 3 carpetas:

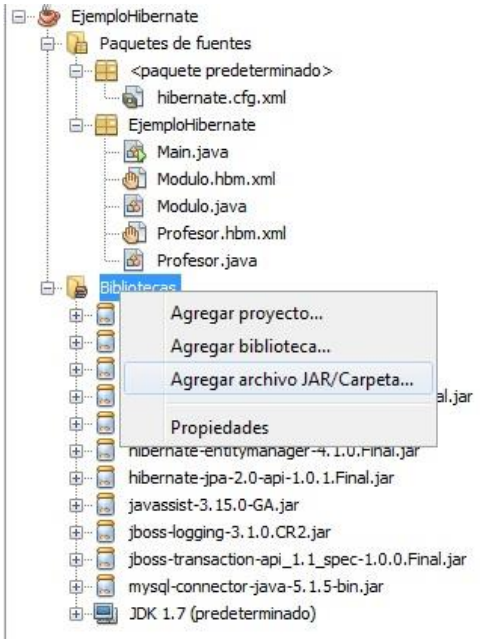
- **lib:** Contiene las librerías (jars) java con el código de Hibernate.
- **documentation:** Documentación sobre Hibernate.
- **project:** Contiene principalmente el código fuente y ficheros de configuración de las distintas bases de datos.

Paso 5: Ahora se deben copiar todos los ficheros jar que se encuentran en la carpeta **lib\required** en la carpeta **lib** de nuestro proyecto java.



Nombre	Fecha de modifica...	Tipo	Tamaño
antlr-2.7.7	31/10/2012 17:28	Executable Jar File	435 KB
dom4j-1.6.1	31/10/2012 17:27	Executable Jar File	307 KB
hibernate-commons-annotations-4.0.1.F...	31/10/2012 17:28	Executable Jar File	80 KB
hibernate-core-4.1.9.Final	13/12/2012 10:48	Executable Jar File	4.402 KB
hibernate-jpa-2.0-api-1.0.1.Final	31/10/2012 17:28	Executable Jar File	101 KB
javassist-3.17.1-GA	12/12/2012 14:27	Executable Jar File	696 KB
jboss-logging-3.1.0.GA	31/10/2012 17:28	Executable Jar File	60 KB
jboss-transaction-api_1.1_spec-1.0.0.Final	31/10/2012 17:28	Executable Jar File	11 KB

Paso 6: Indicar a NetBeans que queremos usar todas esas librerías, para ello con el botón derecho pulsar sobre el árbol en el nodo "**Libraries**" y seleccionar la opción de menú "**Add Jar/Folder...".**

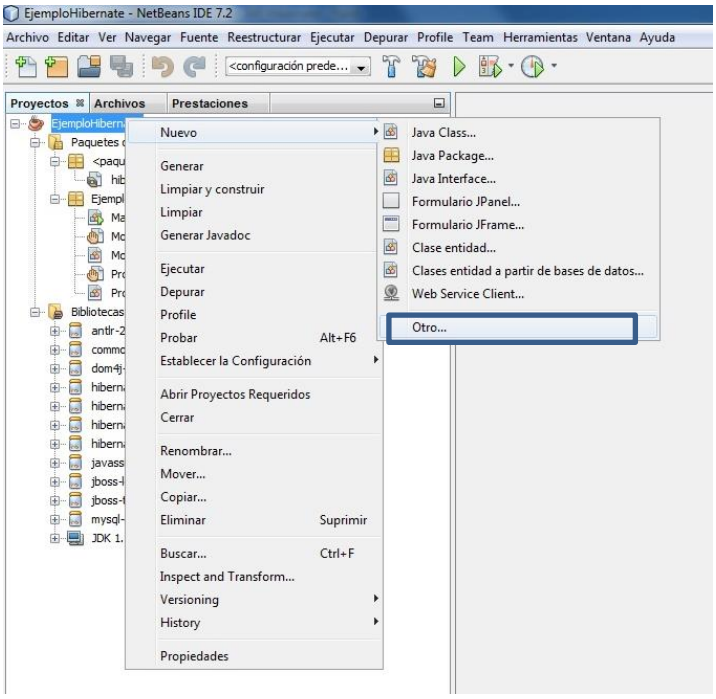


Paso 7: Seleccionar todos los ficheros jar y pulsar el botón "**Abrir**".

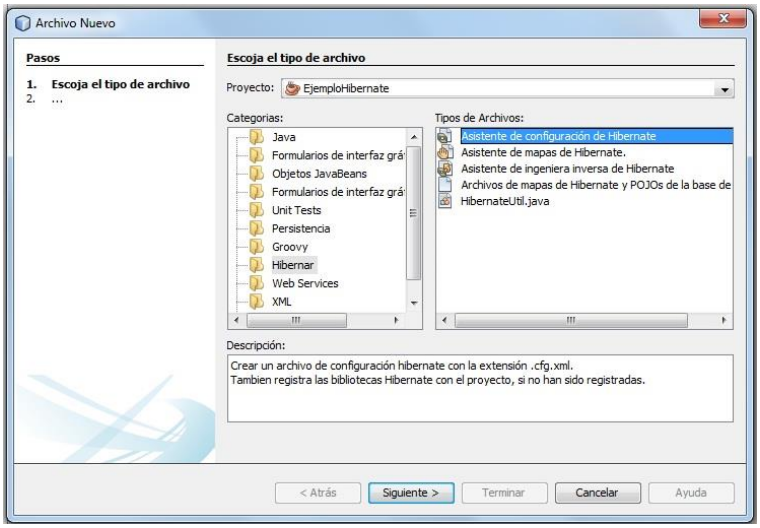
Ahora ya tenemos todas las librerías de Hibernate en nuestro proyecto Java, listas para usarse.

A continuación vamos a persistir la base de Datos. Gracias a Hibernate, podemos generar toda la persistencia automáticamente, mediante un **fichero de configuración de Hibernate.**

Paso1: Hacemos click con el botón derecho sobre el nombre de la solución, seleccionamos **Nuevo** y después en **Otro.**



Paso2: Ahora Seleccionemos **Hibernar,** después **Asistente de configuración de Hibernate.**



Paso3: Seguimos los pasos de asistente, indicando el nombre del fichero de configuración, su ruta.

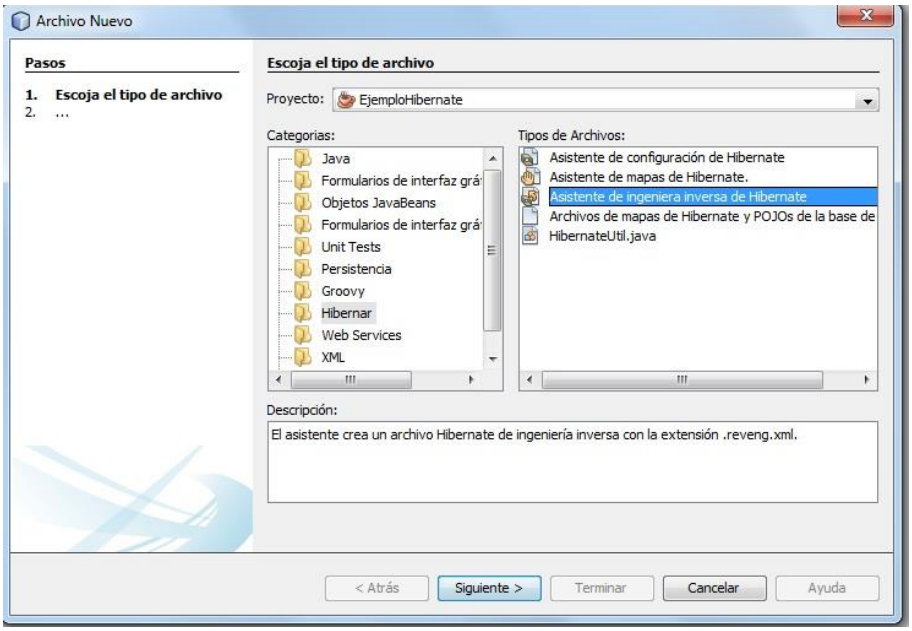
En el último paso, debemos configurar la conexión a BD correspondiente. En este caso, utilizaremos MySQL.

Una vez finalizado el asistente, ya tendríamos creado el **fichero de configuración de Hibernate**.

Este es su resultado: (**Ver anexo Hibernate Fichero de configuración de Hibernate**)

A continuación, vamos a generar un archivo **Hibernate de ingeniería inversa**, es decir, a partir del fichero generado anteriormente, crearemos las relaciones entre las tablas.

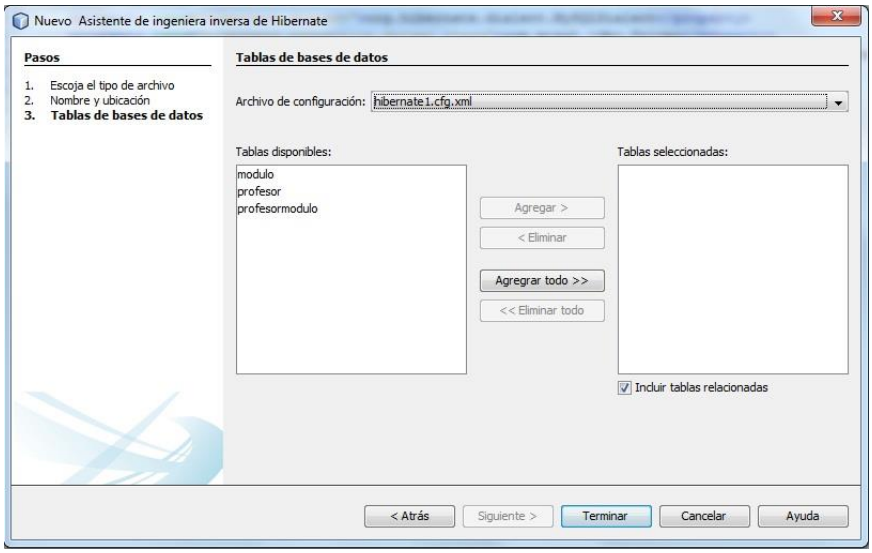
Para ello, repetimos el **Paso1** anterior y en el **Paso2**, seleccionemos **Hibernar,** después **Asistente de ingeniería inversa de Hibernate.**



Paso3: Indicamos el nombre del fichero y su ruta.

Paso4: En este paso, vemos como en la parte superior, aparece el fichero de configuración de Hibernate creado anteriormente. Si todo es correcto, nos aparecen las tablas que forman la base de datos.

.

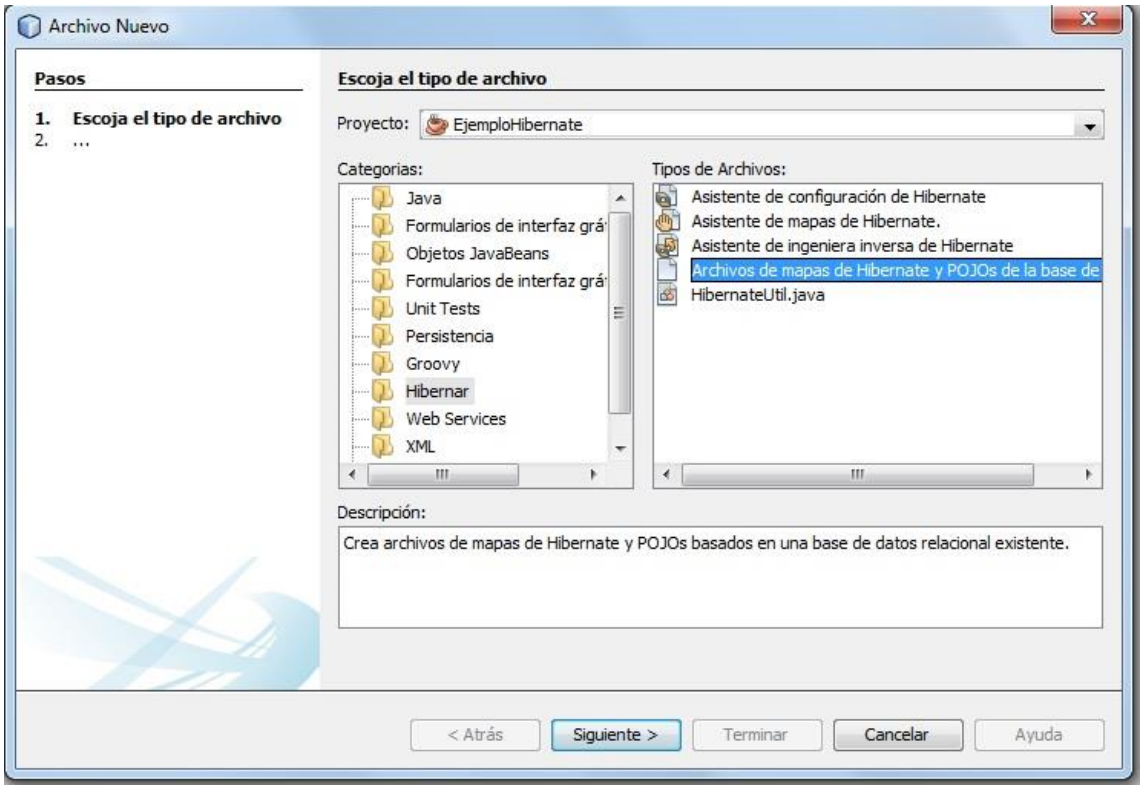


Una vez finalizado el asistente, ya tendríamos creado el archivo **Hibernate de ingeniería inversa** de Hibernate.

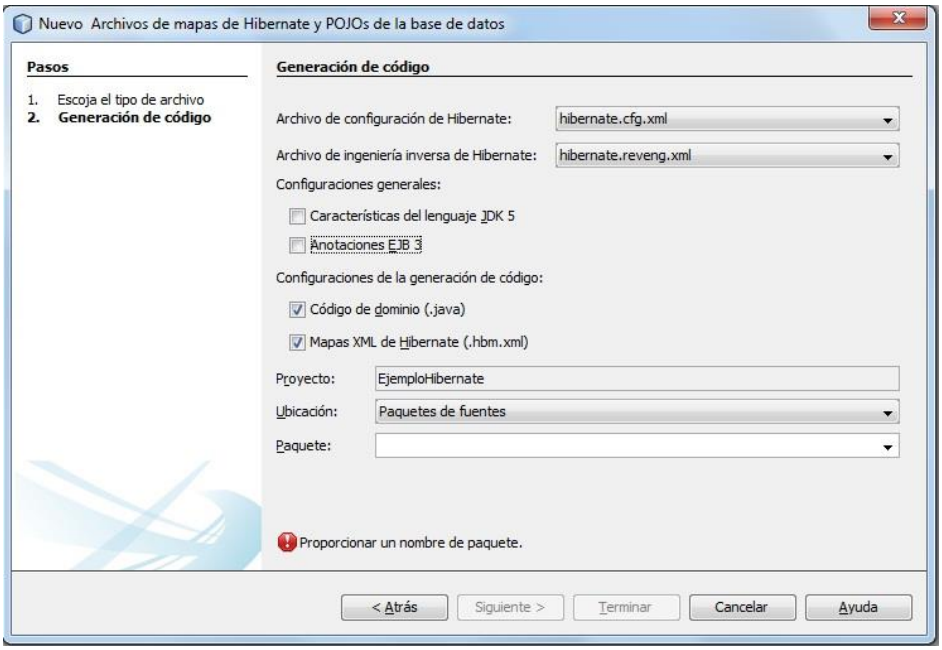
Este es su resultado: (**Ver anexo Hibernate Hibernate de Ingeniería Inversa**)

Por último, crearemos un **archivo de mapas de Hibernate y POJOs basados en una base de datos relacional existente,** es decir, generaremos las clases java, correspondientes a las tablas de la base de datos.

Para ello, repetimos el **Paso1** anterior y en el **Paso2**, seleccionemos **Hibernar,** después **archivo de mapas de Hibernate y POJOs basados en una base de datos relacional existente.**

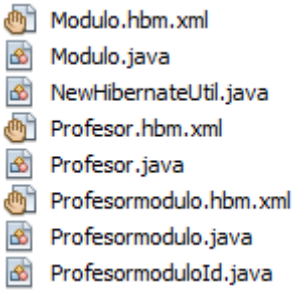


Paso3: En este paso, vemos como en la parte superior, aparece tanto el **fichero de configuración** como **el archivo de ingeniería inversa**, creados anteriormente. Únicamente informamos, el paquete donde crearemos, los ficheros.



Una vez finalizado el asistente, ya tendríamos creado tanto los **ficheros java (POJO`s), como los archivos de mapas.**

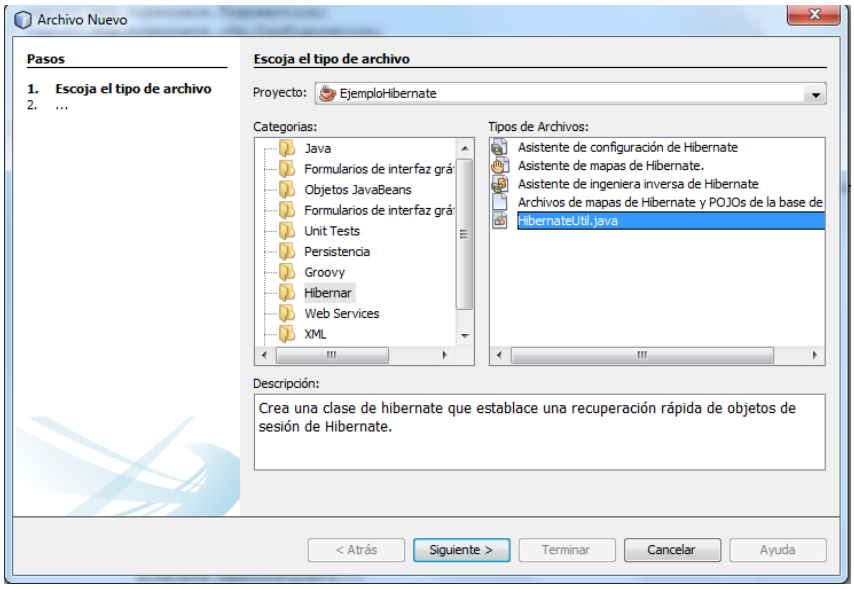
Este es su resultado:



El contenido de ellos, es el siguiente: (**Ver anexo Hibernate Ficheros POJO´s y Archivos Mapas)**

Por último, debemos crear un HibernateUtil, para poder establecer la conexión.

Para ello, repetimos el **Paso1** anterior y en el **Paso2**, seleccionemos **Hibernar,** después **HibernateUtil.java.**



Paso3: En este paso, únicamente informamos, el nombre del fichero y el paquete donde crearemos, los ficheros.

Una vez, finalizado el asistente, ya tenemos creado nuestro **HibernateUtil.**

Este es el resultado **(Ver Anexo Hibernate HibernateUtil)**

Una vez creados todos los ficheros necesarios, nos disponemos a crear la lógica de la aplicación.

Para implantar este ejemplo, se ha realizado una aplicación de consola, en la que se visualizaran los resultados de las operaciones que se van realizando.

Se han creado 3 clases **Select**, **Update** y **Delete**. En ellas se realizan las operaciones de visualización, actualización y borrado, respectivamente. La operación de inserción se realiza en la clase **Main**

La clase Main insertara una nuevo modulo llamado DAM

1. En ella, primero creamos un SessionFactory para establecer la conexión
2. Seguidamente abrimos sesión e iniciamos la transacción.
3. Inicializamos un objeto Modulo e incluimos los datos a insertar.
4. Posteriormente, guardamos el objeto en la sesión y realizamos commit.

La clase Select, visualizará los nombres de los profesores.

En ella, primero creamos un SessionFactory para establecer la conexión.

1. Seguidamente abrimos sesión y creamos la query.
2. Guardamos la información en una lista, gracias al método **.list()**
3. Posteriormente, recorremos la lista y visualizamos los registros.
4. Cerramos la sesión.

La clase Update, actualizará el nombre de un módulo

1. En ella, primero creamos un SessionFactory para establecer la conexión
2. Seguidamente abrimos sesión e iniciamos la transacción y creamos la query.
3. Recorremos la información en una lista, gracias al método **.list()**
4. Inicializamos un objeto Modulo e incluimos los datos a modificar.
5. Posteriormente, guardamos el objeto en la sesión y realizamos commit.

La clase Delete, eliminará un módulo

1. En ella, primero creamos un SessionFactory para establecer la conexión.
2. Seguidamente abrimos sesión y creamos las querys.
3. Ejecutamos las querys
4. Realizamos commit y cerramos la sesión.