TEMA 3.- HERRAMIENTAS MAPEO OBJETO RELACIONAL.

Objetivos

- Concepto de mapeo objeto relacional.
- Características de las herramientas ORM. Herramientas ORM más utilizadas.
- Instalación de una herramienta ORM.
- Configuración de la herramienta.
- Ficheros de configuración, propiedades configurables,
- Estructura de un fichero de mapeo. Elementos, propiedades.
- Mapeo de colecciones, relaciones y herencia.
- Clases persistentes.
- Sesiones; estados de un objeto.
- Carga, almacenamiento y modificación de objetos.
- Consultas SQL.
- Lenguajes propios de la herramienta ORM.
- Gestión de transacciones.

Contenidos

1.- Introducción.

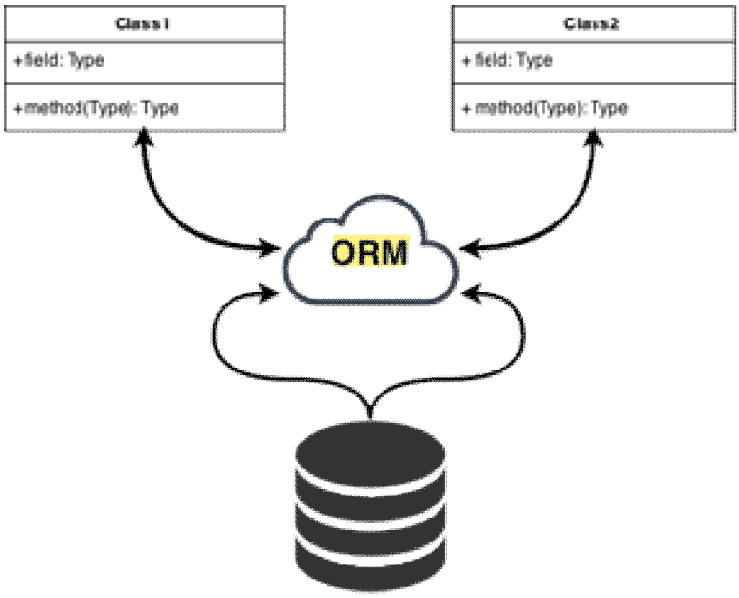
Object-Relational mapping, o lo que es lo mismo, mapeo de objeto-relacional, es un modelo de programación que consiste en la transformación de las tablas de una base de datos, en una serie de entidades que simplifiquen las tareas básicas de acceso a los datos para el programador.

Desde hace muchos años el lenguaje más usado para acceder a las bases de datos relacionales ha sido el SQL. Aunque el lenguaje SQL se usa para acceder a muchas de las bases de datos existentes, existen múltiples varianzas en las funciones que los distintos SGBD han usado. Un ejemplo muy sencillo sería delimitar el número de registros de una consulta:

- **`SELECT TOP 10 * FROM usuarios //SqlServer`**
- **`SELECT * FROM usuarios LIMIT 10 //MySQL`**
- **`SELECT * FROM usuarios WHERE rownum<=10; //Oracle`**

Esto para el programador supone tener que conocer el lenguaje para cada Base de datos, y más importante aún, si en un futuro se desea migrar la aplicación, habría que reescribir gran número de las consultas.

El ORM al tener un capa intermedia, abstrae al programador de la base de datos y le centra en el desarrollo de la aplicación. Otro punto importante es la facilidad de trabajo, un ORM, nos facilita las labores básicas de cualquier acceso a datos , el CRUD (Create, Read, Update y Delete), realizando todas estas labores a través de un lenguaje de alto nivel orientado a objetos.



2.- Características de las herramientas ORM.

Ventajas

- Facilidad y velocidad de uso.
- Abstracción de la base de datos usada.
- Seguridad de la capa de acceso a datos contra ataques.

Desventajas

- En entornos con gran carga poner una capa más en el proceso puede mermar el rendimiento.
- Aprender el nuevo lenguaje del ORM.

Casi todos los lenguajes de alto nivel actualmente disponen de alguna solución de este tipo, una de las más conocidas es Hibernate para JAVA, pero existen muchas más:

- Java => Hibernate, iBatis, Ebean, etc..
- .NET=> Entity Framework, nHibernate, etc..
- PHP=> Doctrine, Propel, ROcks, Torpor, etc..

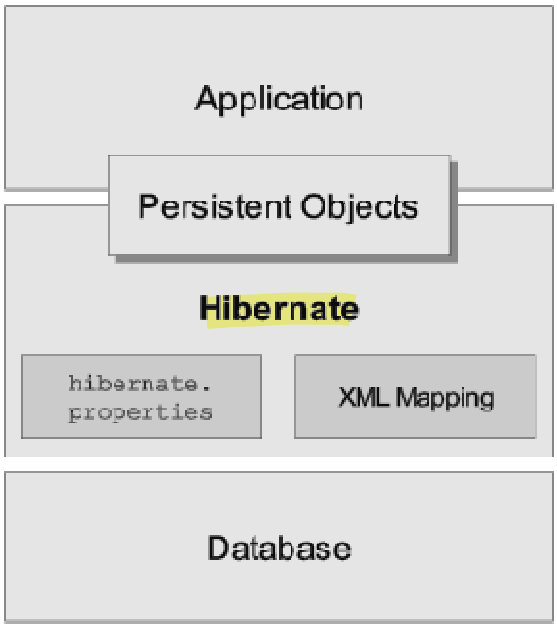
3.- Hibernate.

Hibernate es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

Hibernate es el ORM más potente del mercado y el más usado en Java. Es un software libre y facilita mucho trabajo en desarrollo de aplicaciones OO con BD relacionales.

Se trata de una especie de traductor situado entre nuestra aplicación y la base de datos, que nos abstraerá del diseño de la base de datos, permitiéndonos centrarnos en las

funcionalidades básicas de nuestra aplicación. Es la manera de poder abstraerse de la base de datos mapeando entre las columnas de una tabla de la base de datos y los atributos de una clase mediante archivos de configuración xml.



Debemos realizar es configurar el conector jdbc para que sea empleado por Hibernate y un fichero de configuración base, donde además de los datos de conexión a la BBDD se deben de establecer ciertos parámetros de configuración para Hibernate y las referencias a los recursos de mapeo.

Tras configurar la conexión con la base de datos debemos desarrollar las clase java (modelos) que se asociaran con las tablas de la base de datos. Debe incluir un constructor vació y los getters y setters de todos los atributos, ya que es necesario para Hibernate para el proceso de Mapeo. Las clases implementan la interfaz Serializable, ya que el contenido de los objetos de estas clases se serializarán en la base de datos.

En las relaciones 1 a N requiere un medio de almacenamiento de los datos de la tabla relacionada, para este mapeo se emplea un ArrayList.

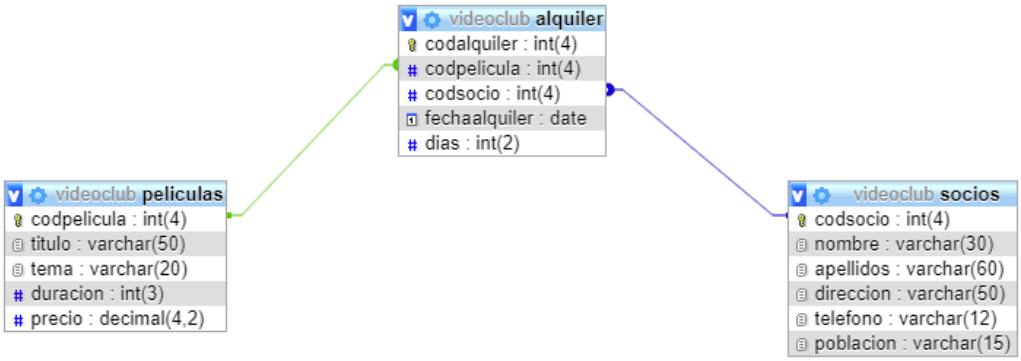
Ejemplo:

```
public class Cliente implements Serializable {
	int id;
	String nombre;
	String apellidos;
	String domicilio;
	List pedidos = new ArrayList();
}
```

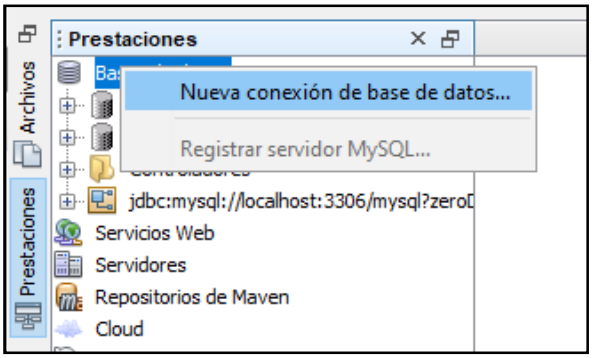
Tras tener configurada la conexión a la base de datos y nuestros modelos (clases) desarrollados, debemos de preparar los ficheros XML de mapeo (Ingeniería Inversa) en el que los atributos de las clases definidas se asocian con las columnas de la base de datos además de definir los tipos de datos que se emplean, la autogeneración de ids, el mapeo de las relaciones con las claves foráneas, etc...

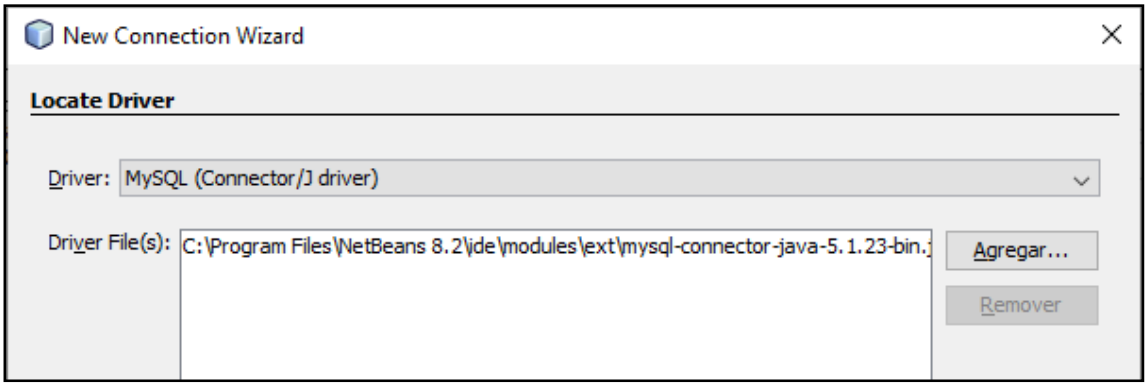
3.- Crear una aplicación en NetBeans con Hibernate.

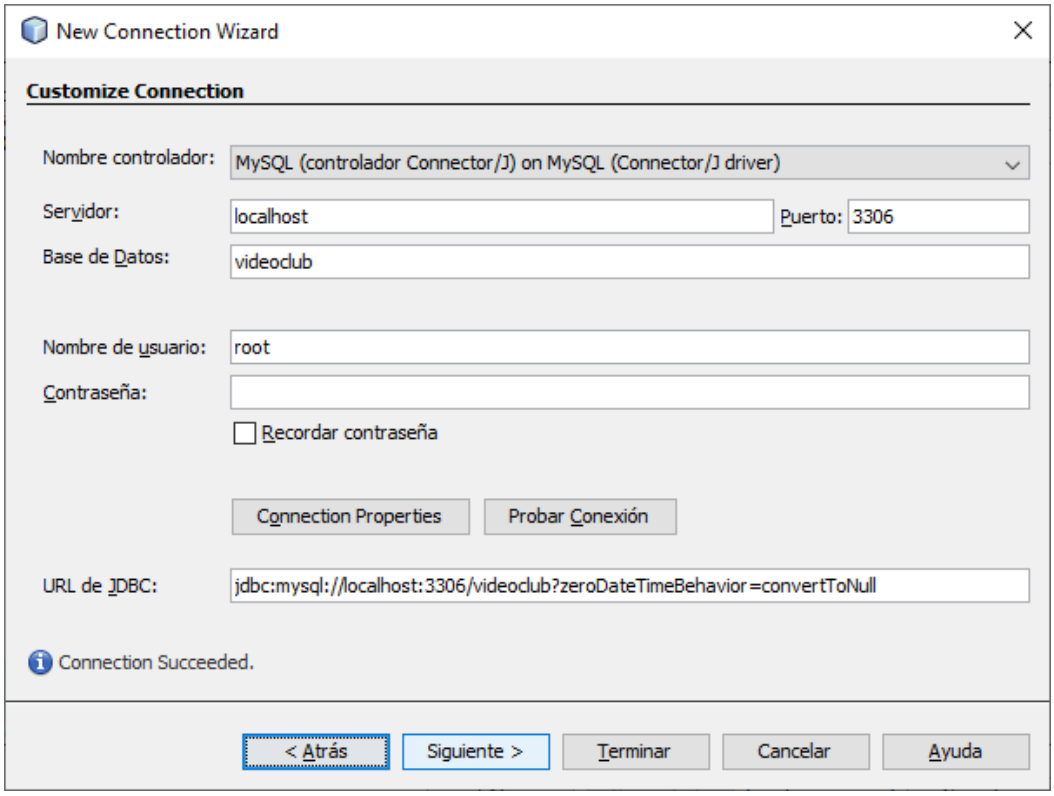
Para realizar un acceso a una base de datos mediante Hibernate con NetBeans partiremos de una Base de Datos llamada Videoclub con el siguiente esquema relacional:

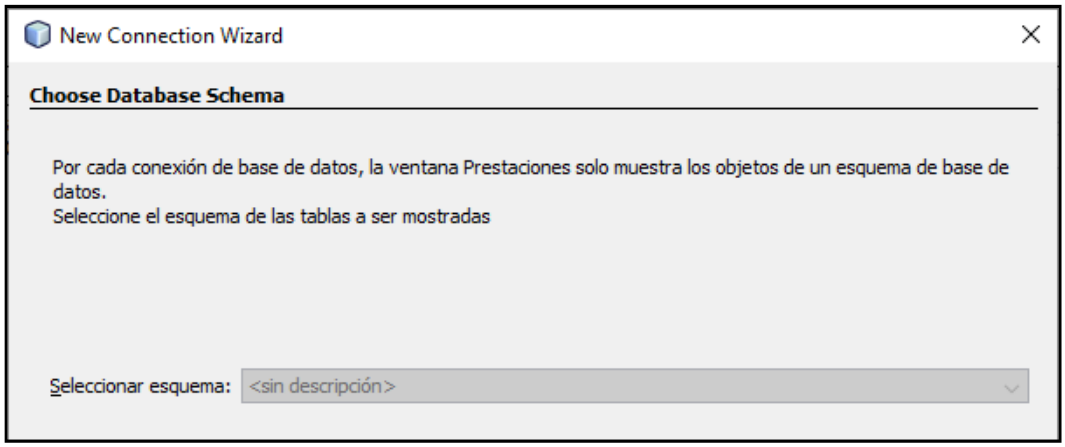


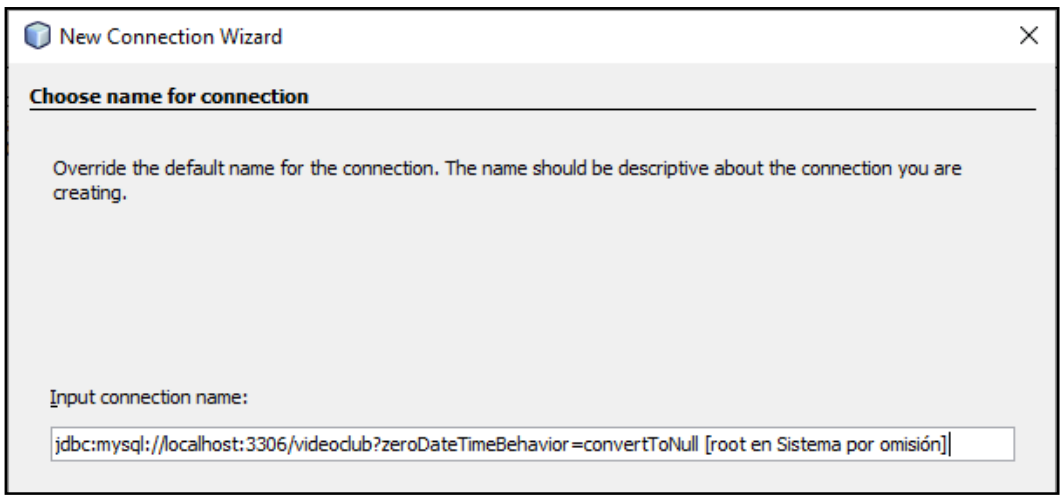
Desde Prestaciones debemos crear una Nueva conexión pulsando con el botón derecho sobre Base de datos. Seleccionamos Nueva Conexión de la Base de datos y seleccionamos el driver e introducimos los datos de la conexión y pulsamos **Probar Conexión** para verificar que todo está correcto:



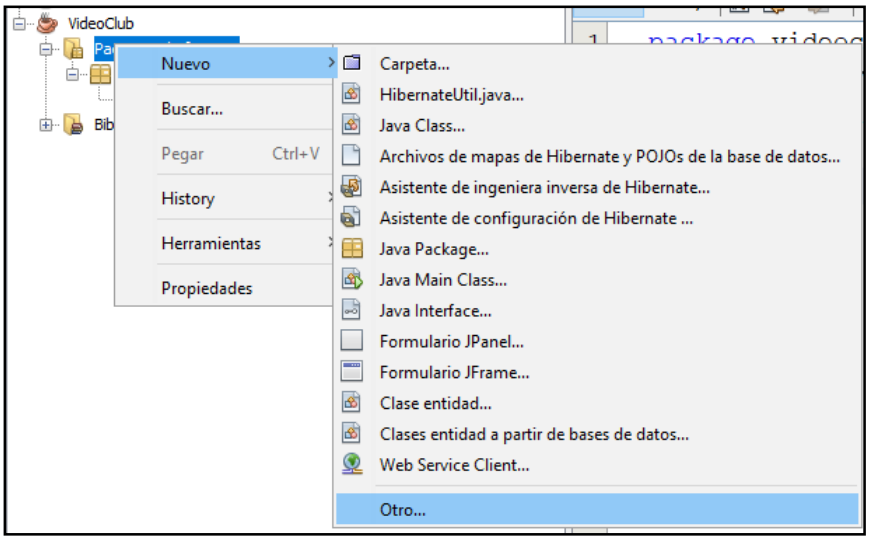



New Connection Wizard
Customize Connection
Nombre controlador:
MySQL (controlador Connector/J) on MySQL (Connector/J driver)
Servidor:
localhost
Puerto:
3306
Base de Datos:
videoclub
Nombre de usuario:
root
Contraseña:
Recordar contraseña
Connection Properties
Probar Conexión
URL de JDBC:
jdbc:mysql://localhost:3306/videoclub?zeroDateTimeBehavior=convertToNull
Connection Succeeded.
< Atrás
Siguiente >
Terminar
Cancelar
Ayuda

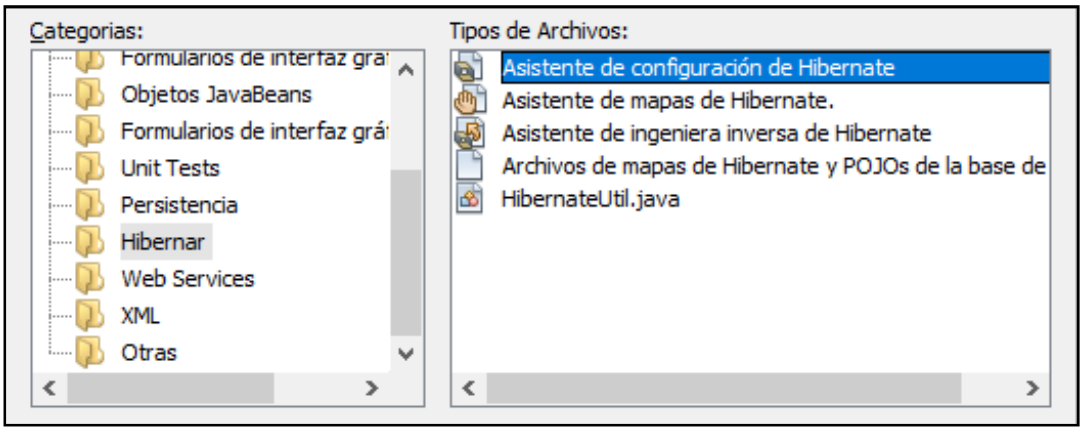

New Connection Wizard
Choose Database Schema
Por cada conexión de base de datos, la ventana Prestaciones solo muestra los objetos de un esquema de base de datos.
Seleccione el esquema de las tablas a ser mostradas
Seleccionar esquema:
<sin descripción>


New Connection Wizard
Choose name for connection
Override the default name for the connection. The name should be descriptive about the connection you are creating.
Input connection name:
jdbc:mysql://localhost:3306/videoclub?zeroDateTimeBehavior=convertToNull [root en Sistema por omisión]

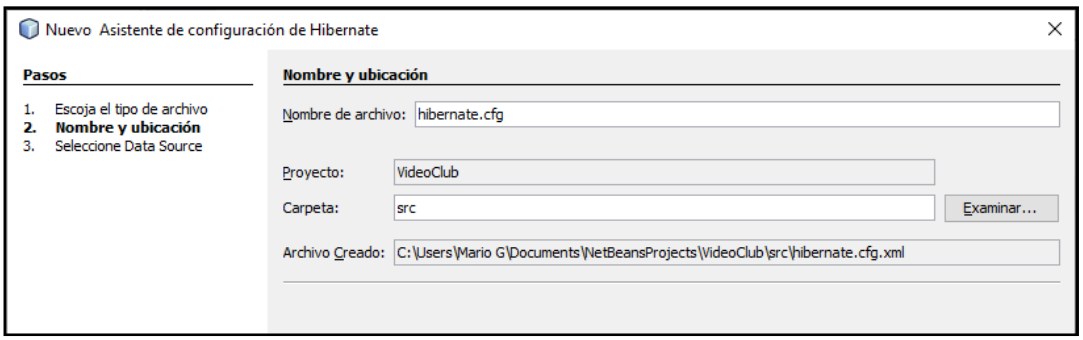
A continuación debemos crear un proyecto. En el Proyecto debemos crear el fichero de configuración de Hibernate XML pulsando botón derecho sobre el proyecto, Nuevo / Otro.



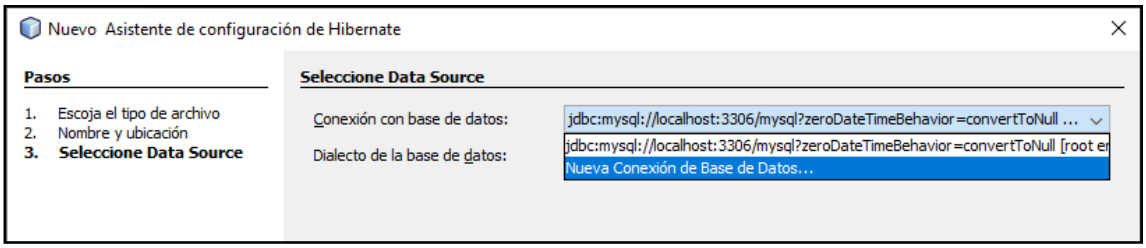
Seleccionaremos Hibernate / Asistente de Configuración de Hibernate y Siguiente.



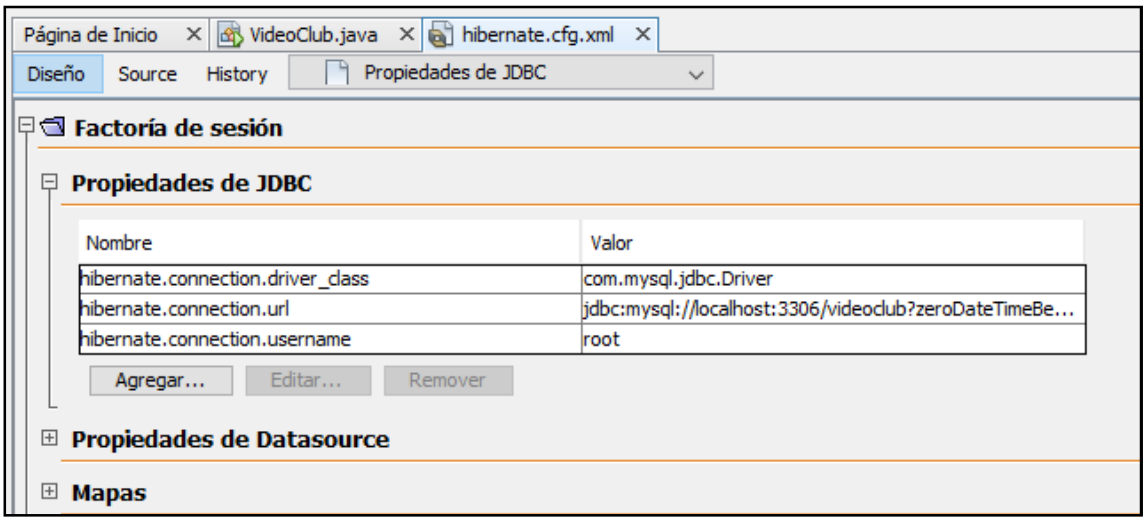
Dejamos el nombre del fichero de configuración y la ubicación y pulsamos Siguiente.



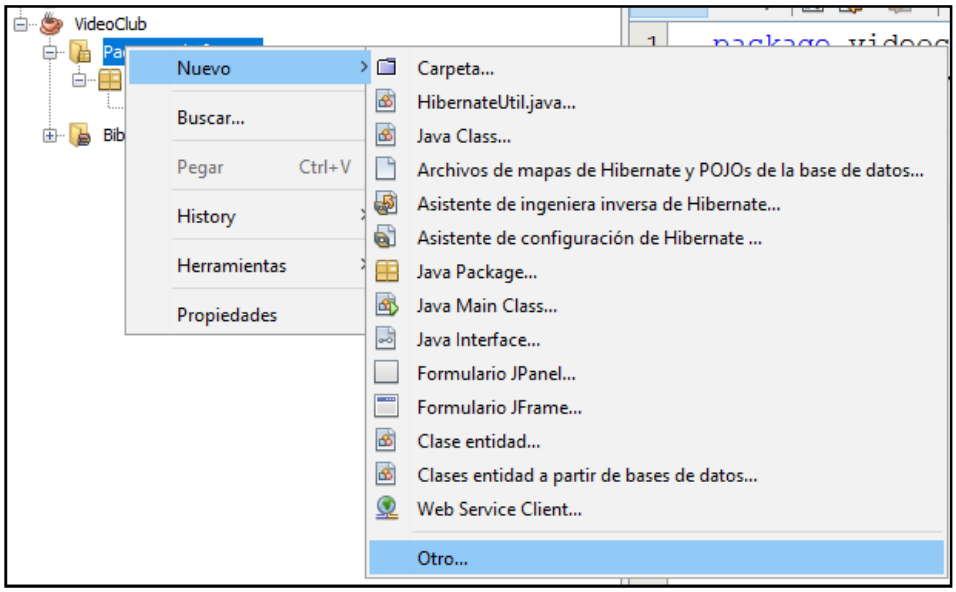
Seleccionamos la conexión establecida con la Base de datos y pulsamos Terminar.



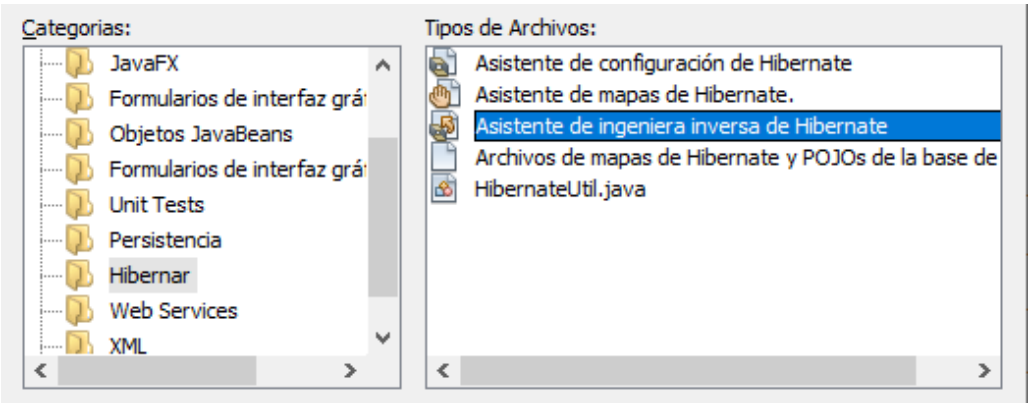
Se nos muestra el fichero XML de conexión con la base de datos:



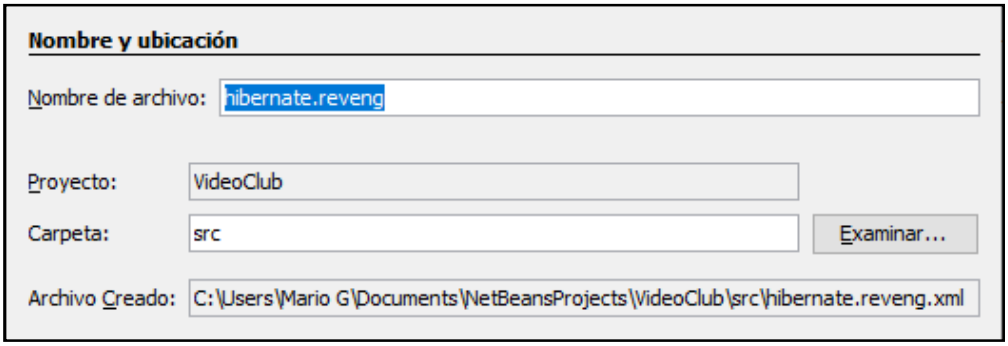
A continuación debemos crear el fichero de Ingeniería Inversa donde se establece la relación entre las tablas y los objetos pulsando botón derecho sobre el proyecto, Nuevo / Otro.



Seleccionaremos Hibernate / Asistente de Ingeniería Inversa de Hibernate y Siguiente.



Dejamos el nombre del fichero de Ingeniería Inversa y la ubicación y pulsamos Siguiente.

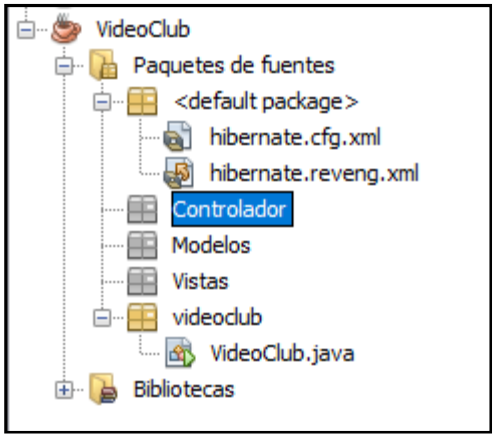


Establecemos la conexión con la Base de datos y seleccionamos las tablas que queremos incluir en el proyecto.

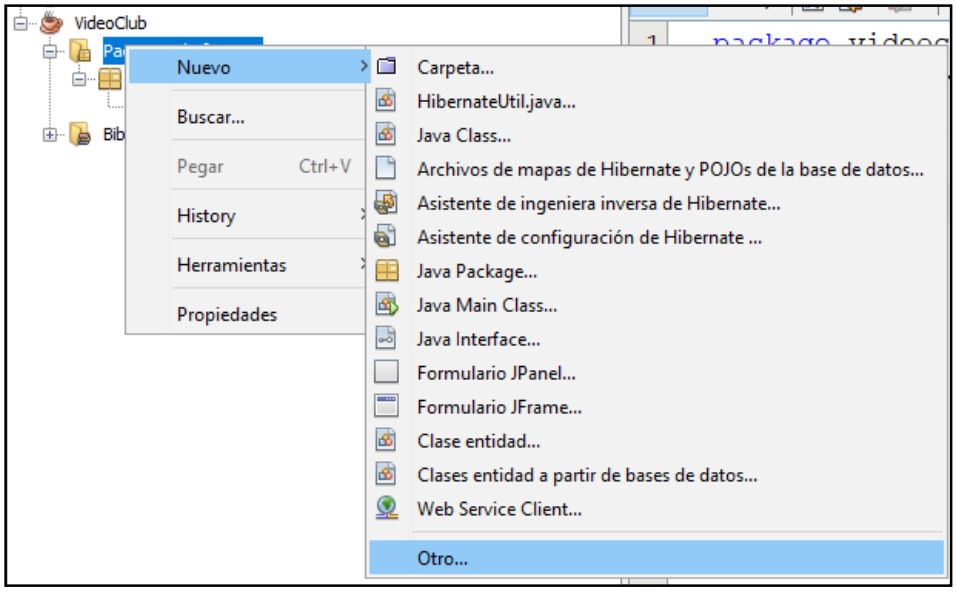
Al pulsar Terminar se muestra el fichero de Ingeniería Inversa XML.

```
<hibernate-reverse-engineering>
    <schema-selection match-catalog="videoclub"/>
    <table-filter match-name="peliculas"/>
    <table-filter match-name="socios"/>
    <table-filter match-name="alquiler"/>
</hibernate-reverse-engineering>
```

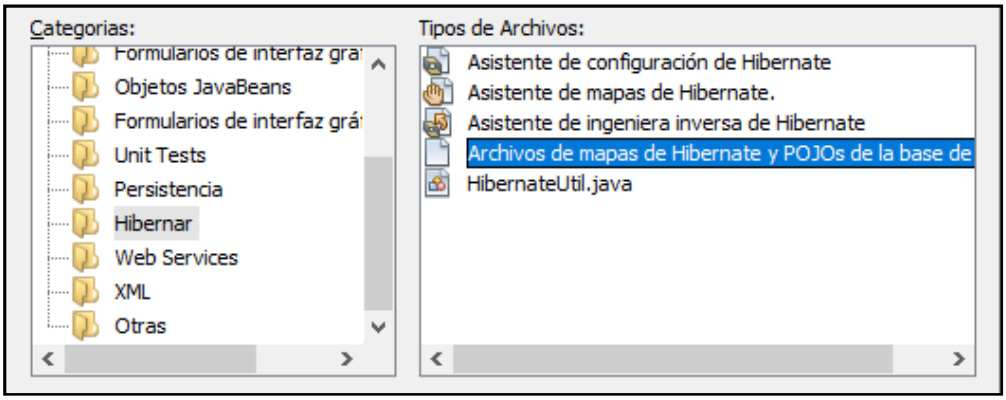
Creamos tres paquetes para establecer el patrón MVC (Modelo – Vista – Controlador).



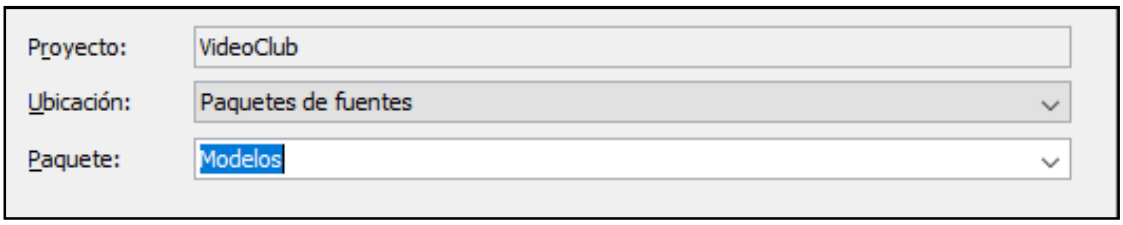
Por último debemos crear las clases que mapean las tablas de las bases de datos pulsando botón derecho sobre el proyecto, Nuevo / Otro.



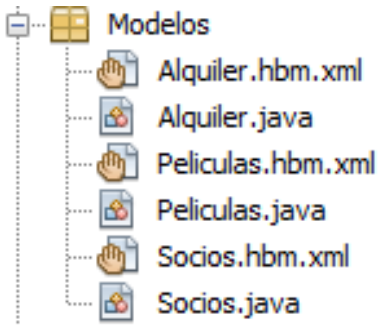
Seleccionaremos Hibernate / Archivos de mapas de Hibernate y POJOs de la base de datos. Pulsamos Siguiente.



Seleccionamos el paquete donde creará las clases POJOs y Terminar.

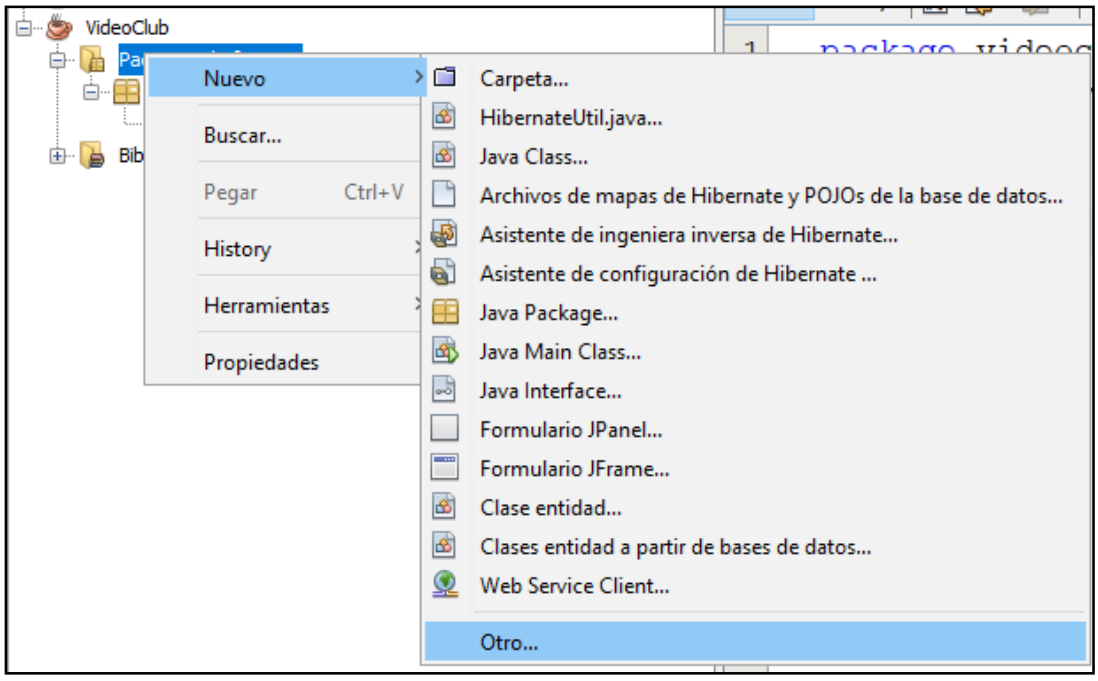


Comprobamos que se han creado las clases en el paquete Modelos.

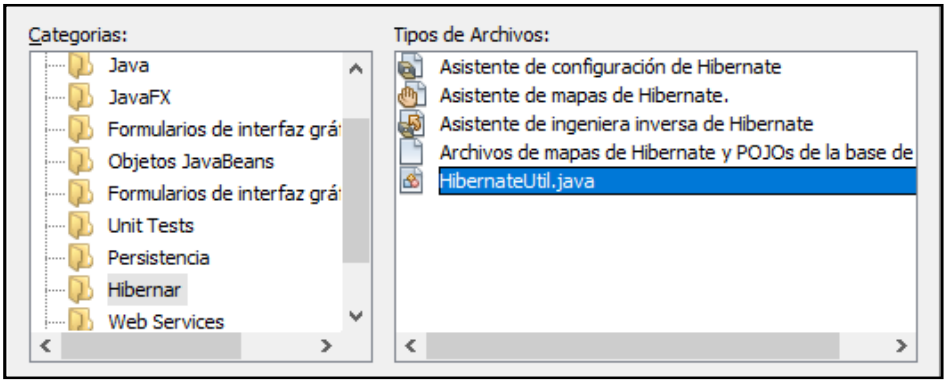


Para poder trabajar con Hibernate debemos tener siempre disponible una referencia al objeto SessionFactory para que cualquier clase pueda tener acceso al objeto Session y por lo tanto a todas las funcionalidades de Hibernate. Esta clase contiene código estático que inicializa Hibernate y crea el objeto SessionFactory. Se incluye además un método estático que da acceso al objeto SessionFactory creado.

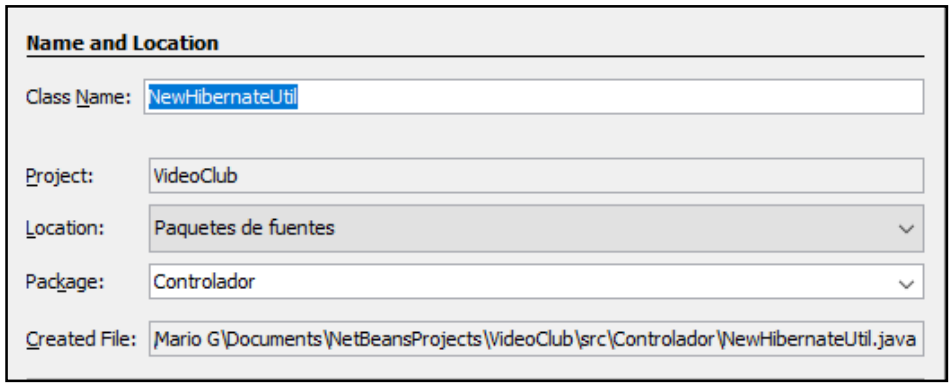
Para crear la clase HibernateUtil pulsamos botón derecho sobre el proyecto, Nuevo / Otro.



Seleccionaremos Hibernate / HibernateUtil.java y pulsamos Siguiente.



Seleccionamos el paquete Controlador para la clase NewHibernateUtil.java y pulsamos Terminar.



Para poder utilizar Hibernate debemos crear el objeto SessionFactory que crea una conexión con la base de datos llamando al método getSessionFactory(). Al finalizar la aplicación deberemos llamar al método close() y cerrar el SessionFactory.

A continuación debemos abrir la sesión con el método openSession() para obtener una sesión de trabajo con Hibernate y una vez finalizado se deberá cerrar la sesión con el método close().

Para trabajar con una base de datos usamos las transacciones. Podemos crear una nueva transacción llamando al método beginTransaction() de la sesión, hacer un commit de la transacción actual llamando al método commit() o hacer un rollback de la transacción actual llamando al método rollback().

Con Hibernate podemos realizar las operaciones fundamentales o CRUD sobre una base de datos:

- **Create**: Guardar un nuevo objeto en la base de datos mediante el método save.
- **Read**:Leer los datos de un objeto de la base de datos mediante el método get, al que le deberemos pasar la clase que queremos leer y su clave primaria. Debemos hacer un casting explicito indicando el tipo de objeto que retorna.
- **Update**:Actualizar los datos de un objeto de la base de datos mediante el método update.
- **Delete**:Borrar los datos de un objeto de la base de datos mediante el método delete.

Muchas veces resulta cómodo al programar no tener que estar pendiente de si un objeto va a insertarse o actualizarse. Para ello Hibernate dispone del método saveOrUpdate que inserta o actualiza en la base de datos en función de si ya existe o no dicha fila.

Para realizar estas operaciones CRUD sobre la base de datos, en la carpeta Controlador creamos una clase Operaciones donde se crearán los métodos que permiten gestionar la base de datos. En la clase Operaciones importamos las clases necesarias para la gestión de la base de datos a través de Hibernate:

```
import Modelo.*;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
```

Creamos el método que permite **insertar una película** en la tabla Peliculas a través de Hibernate:

```
public static void altaPelicula(SessionFactory conexion, Peliculas
miPelicula){
	// Abrimos la sesión para poder realizar una transacción
	// con la base de datos
	Session sesion = conexion.openSession();
	// Comenzamos la transacción
	Transaction tx = sesion.beginTransaction();
	// Insertamos la instancia de pelicula en la base de datos
	sesion.save(miPelicula);
	// Confirmamos la transacción
	tx.commit();
	// Cerramos la sesión
	sesion.close();
	// Mostramos mensaje indicando que se ha insertado el registro
	System.out.println("Registro Insertado");
}
```

En el método main vamos a utilizar el método que permite insertar una película en la tabla Peliculas. Importamos las clases necesarias para la gestión de la base de datos a través de Hibernate:

```
import org.hibernate.SessionFactory;
import Controlador.*;
import Controlador.Operaciones;
import Modelo.*;
// Importamos para Desactivar mensajes Hibernate de consola
import java.util.logging.*;
```

Introducimos el código para insertar una película:

```
public static void main(String[] args) {
   // Desactivamos los mensajes de hibérnate de la consola
   Logger.getLogger("org.hibernate").setLevel(Level.SEVERE);
   // Creamos la conexión con la base de datos con SessionFactory()
   SessionFactory conexion = NewHibernateUtil.getSessionFactory();
   // Creamos una instancia de Pelicula para insertarla en la tabla Peliculas
   BigDecimal precio = new BigDecimal(5.50);
   Peliculas miPelicula = new Peliculas(7,"It","Terror", 98, precio, null);
   // Llamamos al método altaPelicula pasandole como parámetros
   // la conexión y la instancia de Pelicula
   Controlador.Operaciones.altaPelicula(conexion, miPelicula);
   // Cerramos la conexión del SessionFactory()
   conexion.close();
}
```

Creamos el método que permite **consultar una película** en la tabla Peliculas a través de Hibernate:

```
public static Peliculas consultaPelicula(SessionFactory conexion, int
IdPelicula){
   // Declaramos una instancia de Pelicula
   Peliculas miPelicula;
   // Abrimos la sesión para poder realizar una transacción
   // con la base de datos
   Session sesion = conexion.openSession();
   // Comenzamos la transacción
   Transaction tx = sesion.beginTransaction();
   // Almacenamos en la instancia el registro de la base de datos
   miPelicula = (Peliculas)sesion.get(Peliculas.class, IdPelicula);
   // Confirmamos la transacción
   tx.commit();
   // Cerramos la sesión
   sesion.close();
   return miPelicula;
}
```

En el método main vamos a utilizar el método que permite consultar una película en la tabla Peliculas:

```
public static void main(String[] args) {
   // Desactivamos los mensajes de hibérnate de la consola
   Logger.getLogger("org.hibernate").setLevel(Level.SEVERE);
   // Creamos la conexión con la base de datos con SessionFactory()
   SessionFactory conexion = NewHibernateUtil.getSessionFactory();
   // Llamamos al método consultaPelicula pasandole como parámetros
   // la conexión y el Id de la pelicula que queremos consultar
```

```
   System.out.println(Controlador.Operaciones.consultaPelicula(conexion,
6).toString());
   // Cerramos la conexión del SessionFactory()
   conexion.close();
}
```

Creamos el método que permite **listar las películas** en la tabla Peliculas a través de Hibernate:

```
public static List<Peliculas> listarPeliculas(SessionFactory conexion){
       // Declaramos una instancia de Lista de Pelicula
       List<Peliculas> lista;
       // Abrimos la sesión para poder realizar una transacción
       // con la base de datos
       Session sesion = conexion.openSession();
       // Comenzamos la transacción
       Transaction tx = sesion.beginTransaction();
       // Almacenamos en la instancia de lista de pelicula
       // los registros de la consulta de la base de datos
       Query registros = sesion.createQuery("from Peliculas");
       lista = registros.list();
       // Confirmamos la transacción
       tx.commit();
       // Cerramos la sesión
       sesion.close();
       // Retornamos la Lista de registros
       return lista;
}
```

En el método main vamos a utilizar el método que permite listar las películas en la tabla Peliculas:

```
public static void main(String[] args) {
   // Desactivamos los mensajes de hibérnate de la consola
   Logger.getLogger("org.hibernate").setLevel(Level.SEVERE);
   SessionFactory conexion = NewHibernateUtil.getSessionFactory();
   // Llamamos al método listarPeliculas pasandole como parámetros
   // la conexión y mostramos el resultado con un bucle iterator
   List miLista = Controlador.Operaciones.listarPeliculas(conexion);
   Iterator iter = miLista.iterator();
   while (iter.hasNext()){
       System.out.println(iter.next());
   }
   // Cerramos la conexión del SessionFactory()
   conexion.close();
}
```

Creamos el método que permite **eliminar una película** en la tabla Peliculas a través de Hibernate:

```
public static void eliminarPelicula(SessionFactory conexion, int IdPelicula){
```

```
   // Declaramos una instancia de Pelicula
   Peliculas miPelicula;
   // Abrimos la sesión para poder realizar una transacción
   // con la base de datos
   Session sesion = conexion.openSession();
   // Comenzamos la transacción
   Transaction tx = sesion.beginTransaction();
   // Creamos una instancia con el Id de la Pelicula que queremos eliminar
   // y llamamos al método delete pasandole el objeto que queremos borrar
   miPelicula = new Peliculas();
   miPelicula.setCodpelicula(IdPelicula);
   sesion.delete(miPelicula);
   // Confirmamos la transacción
   tx.commit();
   // Cerramos la sesión
   sesion.close();
   // Mostramos mensaje indicando que se ha eliminado el registro
   System.out.println("Registro Eliminado");
}
```

En el método main vamos a utilizar el método que permite eliminar una película en la tabla Peliculas:

```
public static void main(String[] args) {
        // Desactivamos los mensajes de hibérnate de la consola
        Logger.getLogger("org.hibernate").setLevel(Level.SEVERE);
         // Creamos la conexión con la base de datos con SessionFactory()
         SessionFactory conexion = NewHibernateUtil.getSessionFactory();
         // Llamamos al método eliminarPelicula pasandole como parámetros
         // la conexión y el Id de la Pelicula que queremos eliminar
         int IdPelicula = 7;
         Controlador.Operaciones.eliminarPelicula(conexion, IdPelicula);
         // Cerramos la conexión del SessionFactory()
         conexion.close();
}
```

Creamos el método que permite **actualizar una película** en la tabla Peliculas a través de Hibernate:

```
public static void actualizarPelicula(SessionFactory conexion, Peliculas
miPelicula){
   // Abrimos la sesión para poder realizar una transacción
   // con la base de datos
   Session sesion = conexion.openSession();
   // Comenzamos la transacción
   Transaction tx = sesion.beginTransaction();
   // Llamamos al método update pasandole el objeto que queremos actualizar
   sesion.update(miPelicula);
   // Confirmamos la transacción
   tx.commit();
   // Cerramos la sesión
   sesion.close();
```

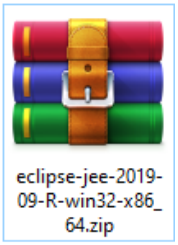
```
   // Mostramos mensaje indicando que se ha actualizado el registro
   System.out.println("Registro Actualizado");
}
```

En el método main vamos a utilizar el método que permite actualizar una película en la tabla Peliculas:

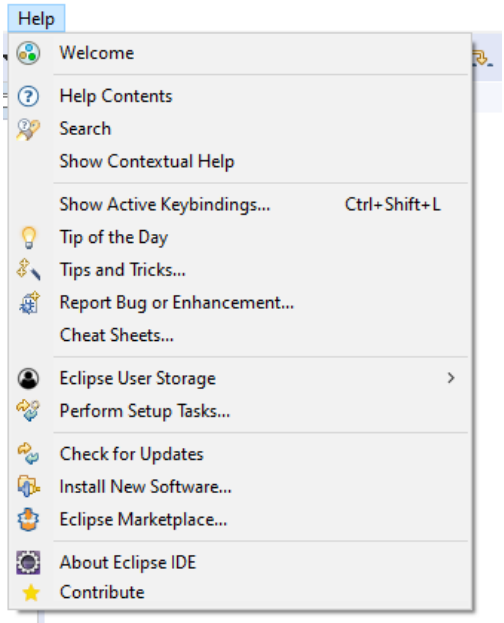
```
public static void main(String[] args) {
   // Desactivamos los mensajes de hibérnate de la consola
   Logger.getLogger("org.hibernate").setLevel(Level.SEVERE);
   // Creamos la conexión con la base de datos con SessionFactory()
   SessionFactory conexion = NewHibernateUtil.getSessionFactory();
   // Creamos una instancia de Pelicula para insertarla en la tabla Peliculas
   BigDecimal precio = new BigDecimal(1.30);
   Peliculas miPelicula = new Peliculas(5,"Thor","Aventuras", 123, precio,
null);
   // Llamamos al método actualizarPelicula pasandole como parámetros
   // la conexión y la instancia de Pelicula
   Controlador.Operaciones.actualizarPelicula(conexion, miPelicula);
   // Cerramos la conexión del SessionFactory()
   conexion.close();
}
```

4.- Crear una aplicación en Eclipse con Hibernate.

Instalamos Eclipse IDE for Java EE Developers Versión 2019-09.

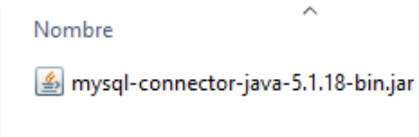


Para instalar y configurar Hibernate en Eclipse debemos pulsar en Help \ Eclipse Marketplace

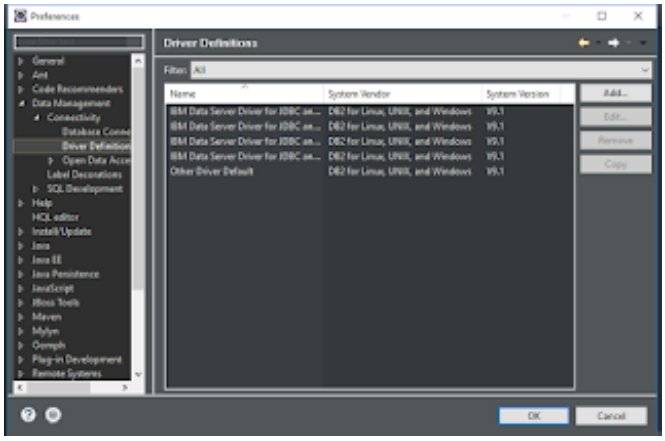


En Find introducimos Jboss y pulsamos Go. Nos muestra JBoss Tools X.XX y pulsamos el botón Install. Pulsamos Confirmar, Aceptar términos de licencia, pulsamos Finish y esperamos a que nos pida reiniciar.

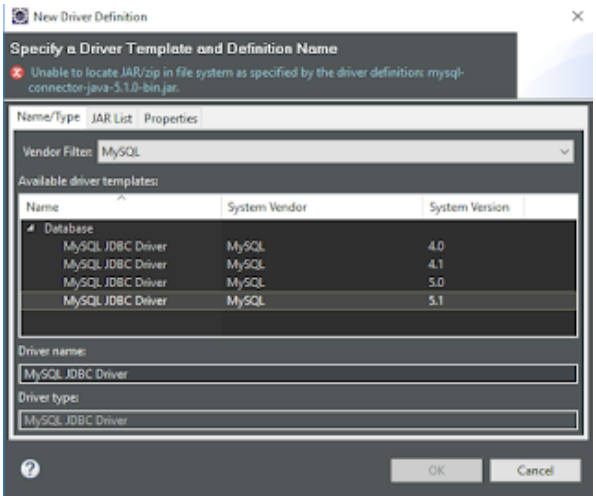
A continuación seleccionaremos el driver para el desarrollo de Hibernate sobre MySQL agregándolo como una librería de Eclipse, para ello creamos la carpeta driverMySql en la carpeta de Eclipse y copiamos el fichero:



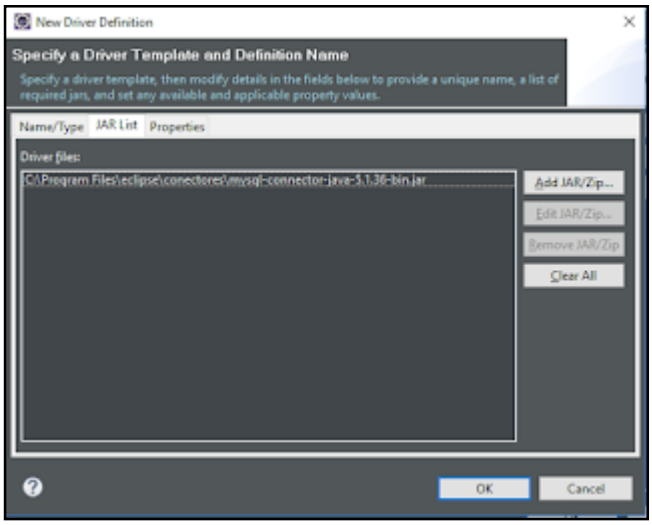
Pulsamos en Window \ Preferences \ Data Management \ Connectivity \ Driver Definition.



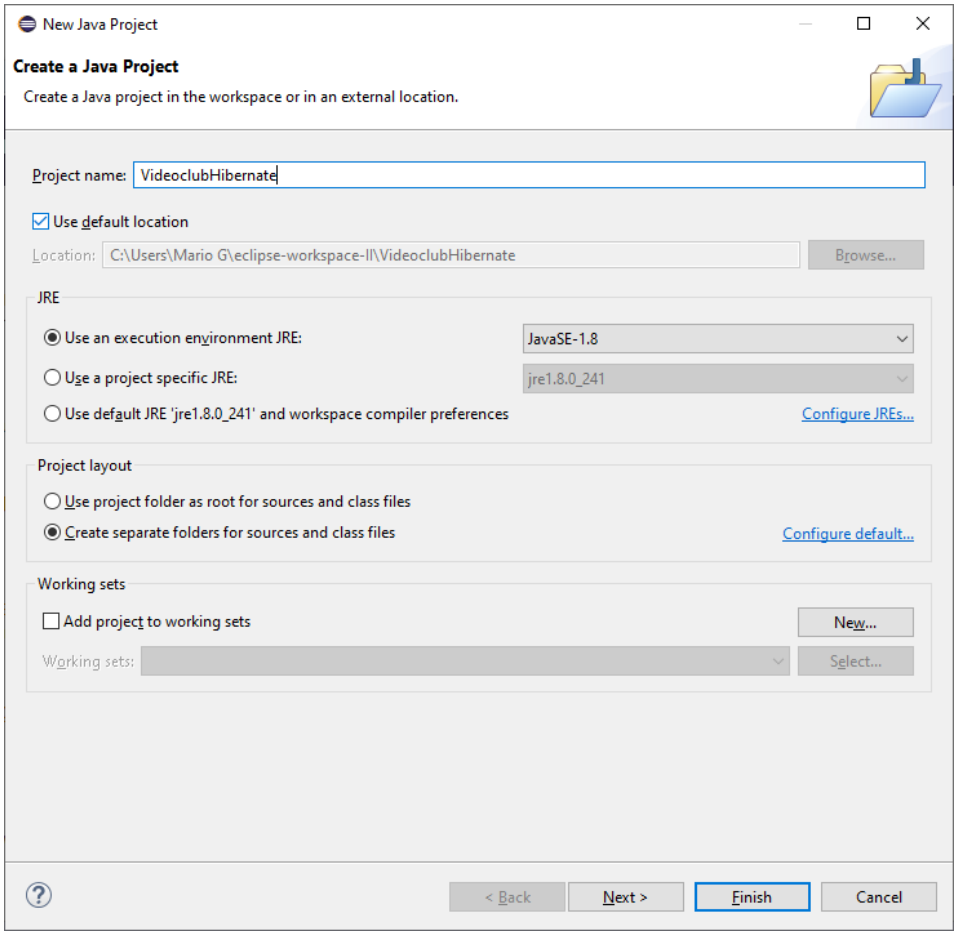
Pulsamos sobre Add y en la pestaña Name/Type en Vendor Filter seleccionamos MySQL, y pulsamos MySQL JDBC Driver la versión 5.1.



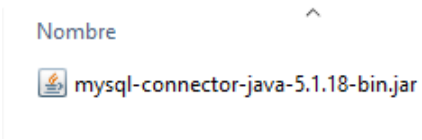
En la pestaña JAR List seleccionamos el driver existente y pulsamos Remove JAR/Zip. A continuación pulsamos sobre Add JAR/Zip y elige el conector guardado en la carpeta de Eclipse y pulsamos el botón OK y Apply an Close.



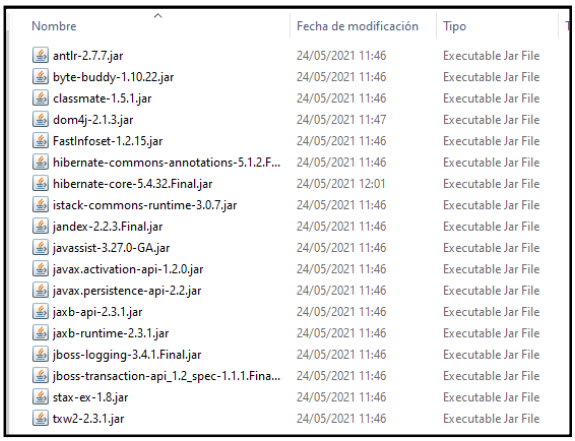
A continuación, creamos un Proyecto Java en eclipse llamada VideoclubHibernate y pulsamos Finish.



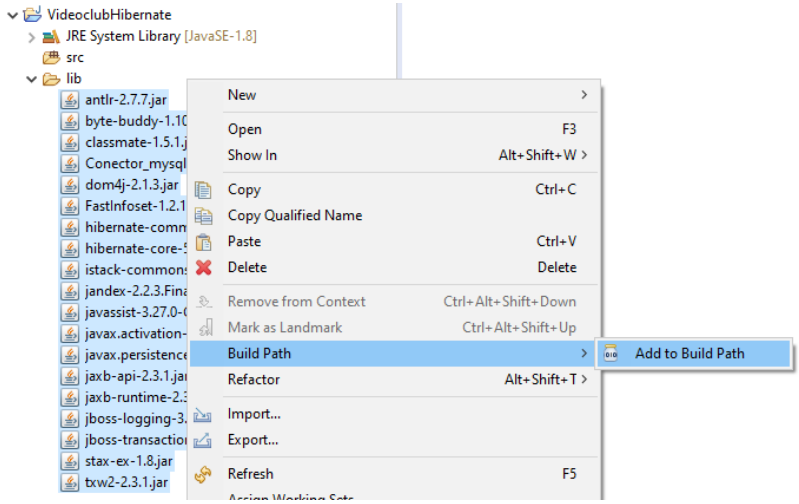
Creamos en la estructura del proyecto una carpeta llamada lib y copiamos el driver de MySQL.



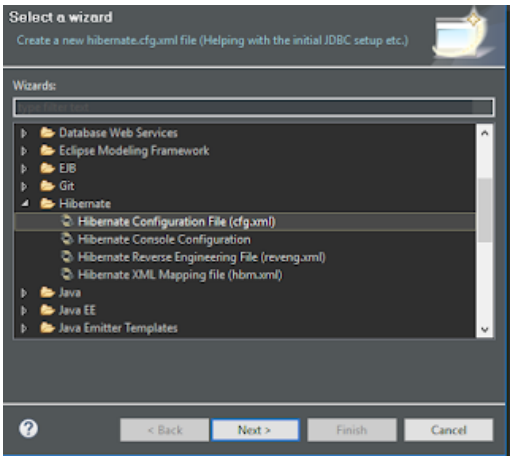
A continuación copiamos a la carpeta lib de la estructura del proyecto los ficheros JAR de Hibernate.

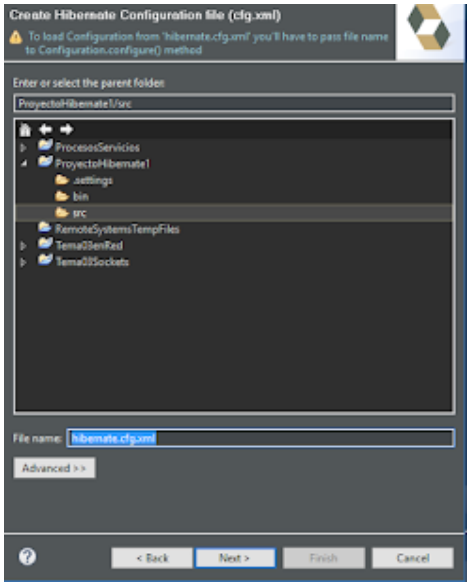


Seleccionamos todos los ficheros de la carpeta lib del proyecto y pulsamos botón derecho Build Path \ Add to Build Path

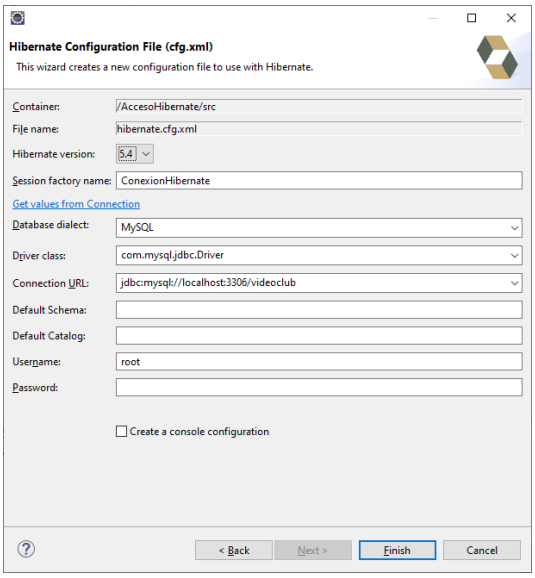


Para crear el fichero de configuración de Hibernate pulsamos botón derecho sobre el proyecto y **New \ Other \ Hibernate Configuration file** y le indicamos que lo almacene en **src** del proyecto.



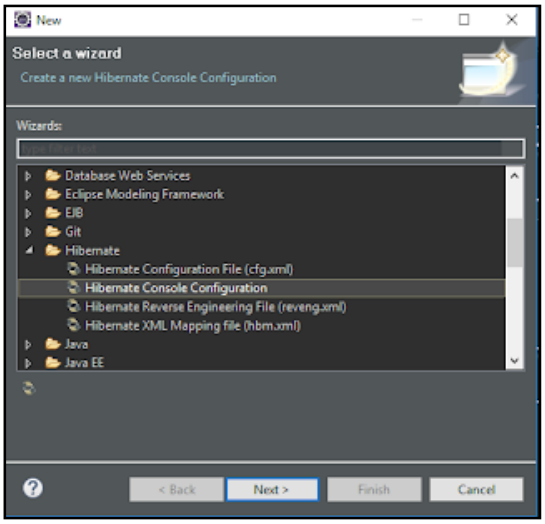


Pulsamos Next y procederemos a su configuración. En Session Factory name le indicaremos el nombre de nuestra sesión, por ejemplo **ConexionHibernate** e introducimos los datos de la conexión.



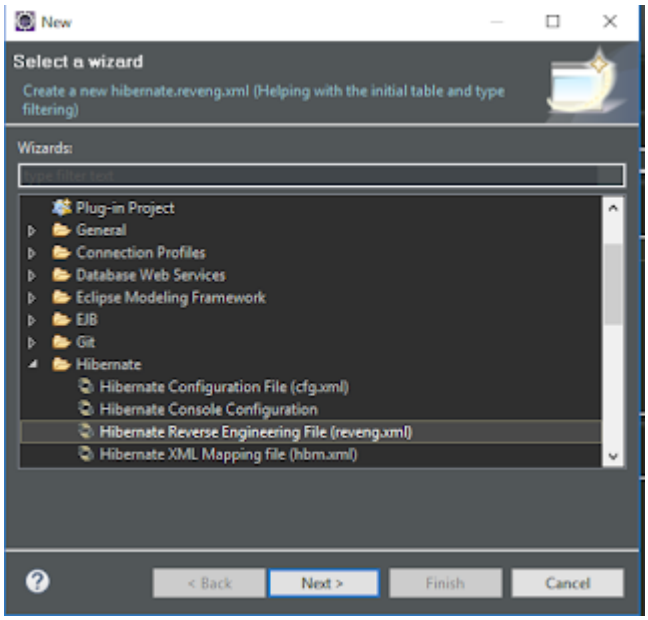
Database dialect seleccionamos **MySQL** y como Driver class elegimos **com.mysql.jdbc.Driver**. La **Connection URL** sería **jdbc:mysql://localhost:3306/basedatos**, siendo basedatos el nombre de la base de datos a la que queremos conectar e indicamos el usuario **root** y la contraseña vacía y pulsamos **Finish**.

A continuación debemos crear un archivo de configuración de consola. Nos vamos a New \ Other \ Hibernate \ Hibernate Console Configuration.

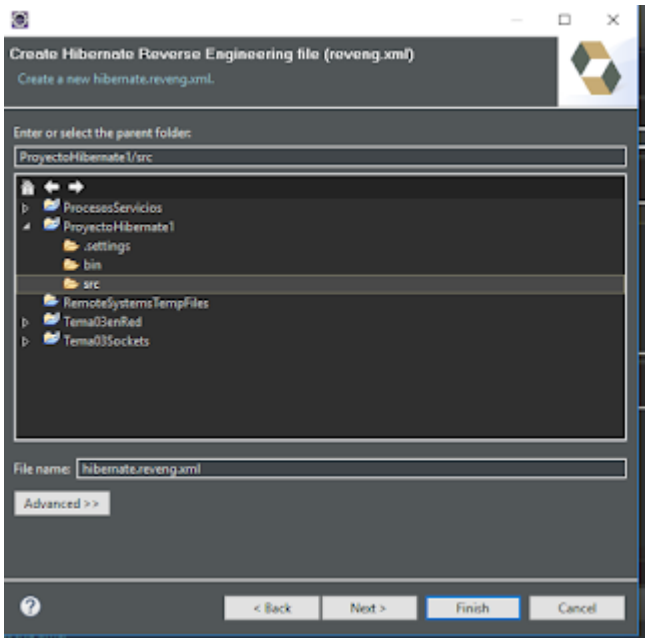


Pulsamos **Finish**.

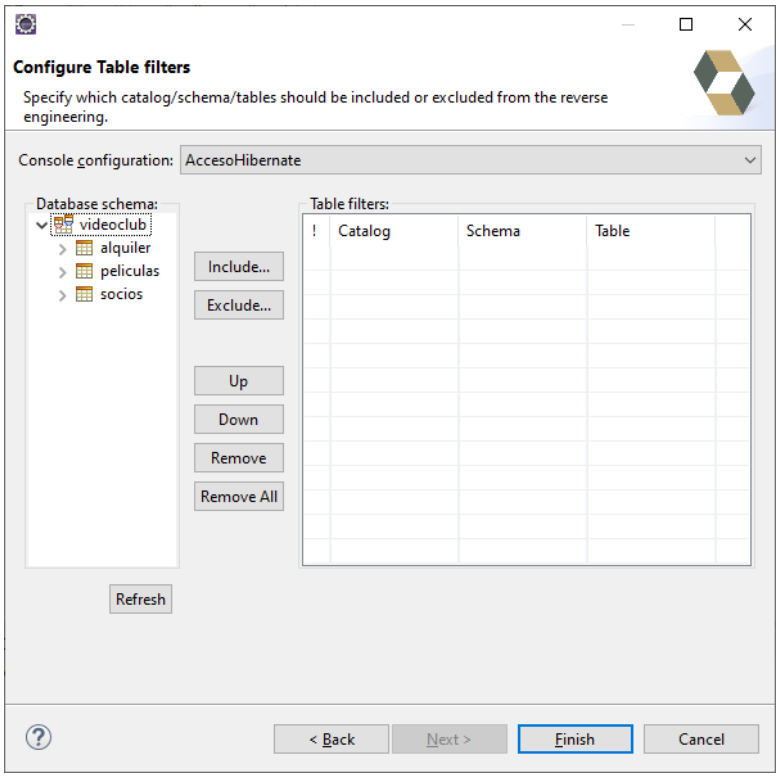
También debemos de crear un archivo para la ingeniería inversa que transforma los datos de la base de datos relacional en objetos. Pulsaremos botón derecho sobre el proyecto y New \ Other \ Hibernate \ Hibernate Reverse Engineering File.



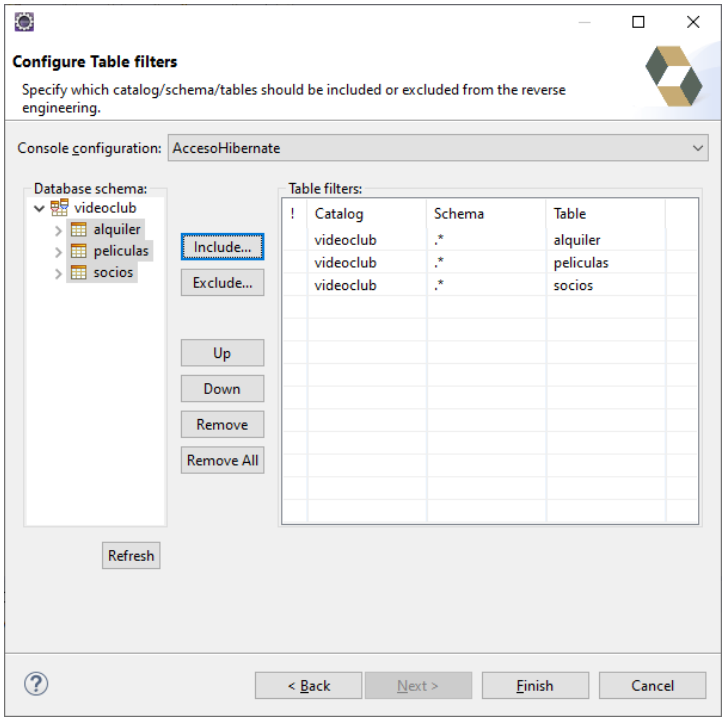
Seleccionamos la carpeta **src** de nuestro proyecto y dejamos el nombre por defecto.



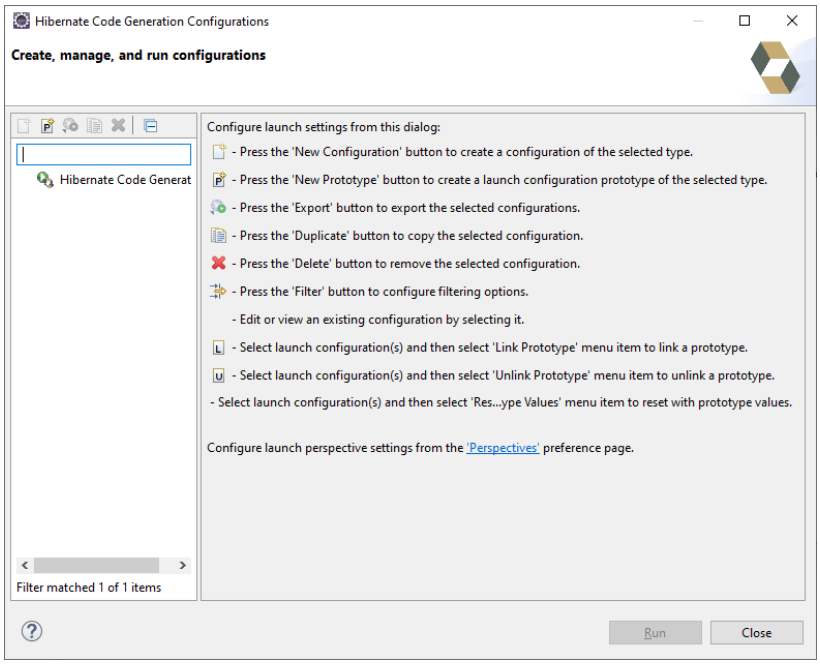
Pulsamos Next y seleccionamos en Console Configuration la configuración que creamos anteriormente en nuestro proyecto y pulsamos **Refresh**.



Nos mostrará la base de datos con sus respectivas tablas. Seleccionamos las tablas y pulsamos **Include**. A continuación pulsamos **Finish**.



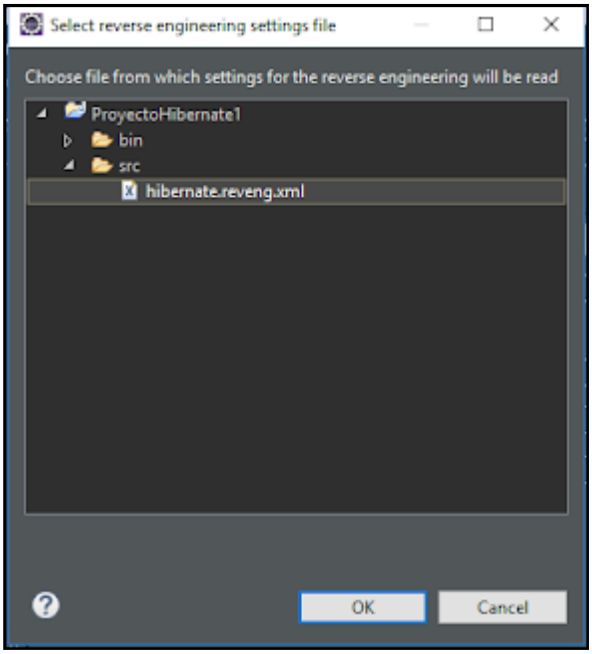
Por último indicaremos a Hibernate que genere el código en Java pulsando Run \ Hibernate Code Generation \ Hibernate Code Generation Configurations.

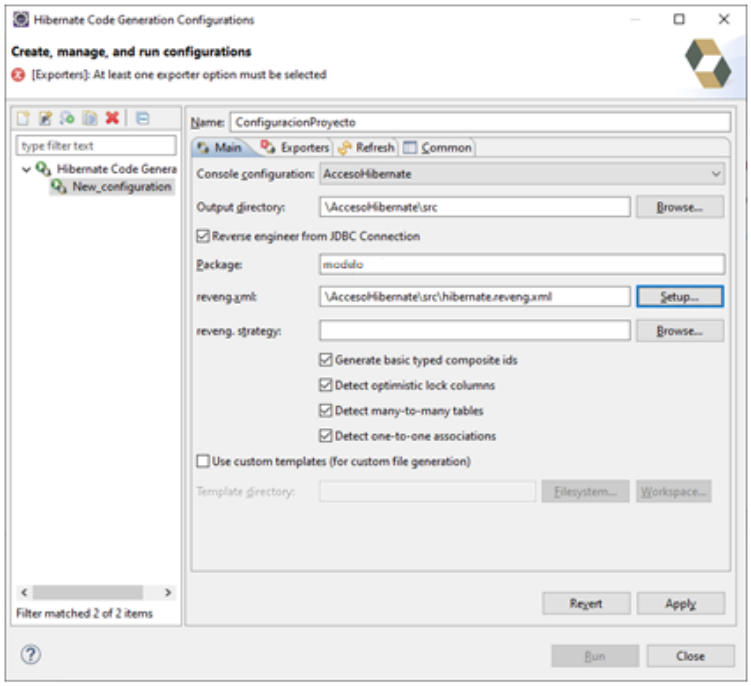


Pulsamos doble clic sobre **Hibernate Code Generation** e indicamos en **Name** el nombre de la configuración, por ejemplo **ConfiguracionProyecto**.

En la pestaña **Main** seleccionamos del desplegable la configuración de consola (Console configuration) que se creó anteriormente y marcamos la opción **Reverse engineer from JDBC Connection**. Indicamos la ruta de salida de los ficheros (Output directory) que será la carpeta src del proyecto. En Package introducimos el paquete donde se almacenará, por ejemplo **modelo**.

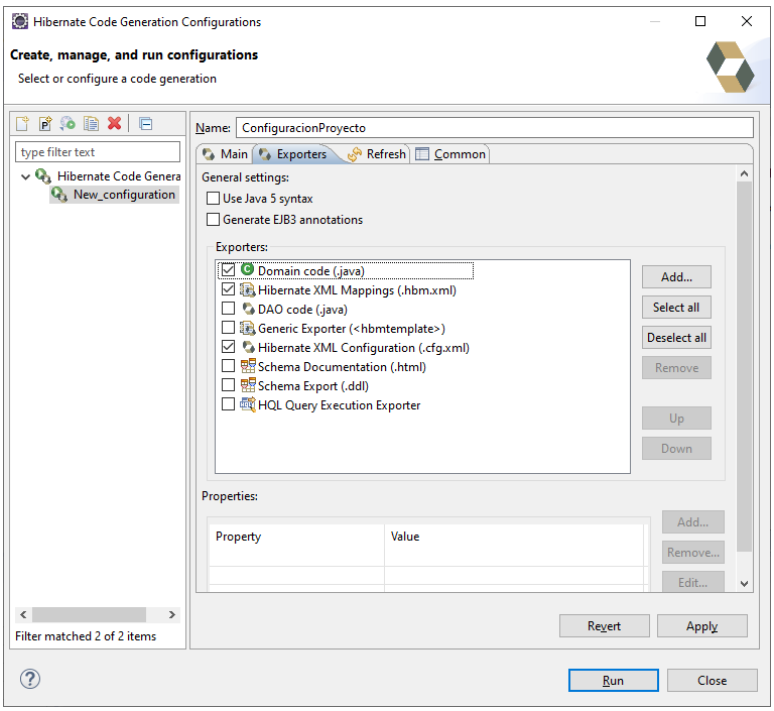
Pulsamos sobre el botón **Setup** de **reveng.xml** e indicamos que queremos usar el fichero **hibernate.reveng.xml** creado anteriormente y almacenado en **src**.





En la pestaña **Exporters**, marcaremos las opciones:

- Domain code (.java)
- Hibernate XML Mappings (.hbm.xml)
- Hibernate XML Configuration (.cfg.xml)



Pulsamos el botón **Apply** para aplicar los cambios y pulsamos el botón **Run**.

Por último debemos crear la Clase que nos permite crear la conexión a la base de datos pulsando botón derecho en el proyecto y New / Other / Class y le damos el nombre **HibernateUtil**. Esta clase Netbeans la creaba automáticamente.

Introducimos en la clase el siguiente código y guardamos el proyecto.

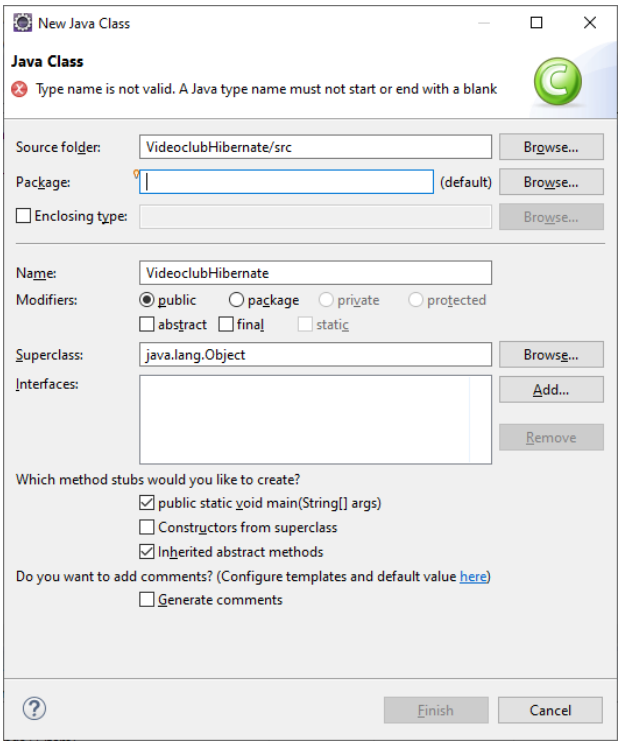
```
package modelo;

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

public class HibernateUtil {
	private static final SessionFactory sessionFactory;
	static {
		try {
			sessionFactory = 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;
	}
}
```

Por último creamos una clase con el método main() pulsando botón derecho en el proyecto y New / Other / Class y le damos el nombre **VideoclubHibernate** y en Package lo dejamos vacio.



Escribimos el siguiente código en la clase:

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import modelo.*;
// Importamos para Desactivar mensajes Hibernate de consola
import java.math.BigDecimal;
import java.util.logging.*;

public class VideoclubHibernate{
	public static void main(String[] args) {
		// Desactivamos los mensajes de hibérnate de la consola
		LogManager.getLogManager().reset();
		Logger globalLogger =
		    Logger.getLogger(java.util.logging.Logger.GLOBAL_LOGGER_NAME);
		globalLogger.setLevel(java.util.logging.Level.OFF);
		// Creamos la conexión con la base de datos con SessionFactory()
		SessionFactory conexion = HibernateUtil.getSessionFactory();
		// Creamos instancia de Pelicula para insertarla en la tabla Peliculas
		BigDecimal precio = new BigDecimal(5.50);
		Peliculas miPelicula = new Peliculas(9999,"It","Terror", 98, precio);
		// Llamamos al método altaPelicula pasandole como parámetros
		// la conexión y la instancia de Pelicula
		// Abrimos la sesión para poder realizar una transacción
		// con la base de datos
		Session sesion = conexion.openSession();
		// Comenzamos la transacción
		Transaction tx = sesion.beginTransaction();
		// Insertamos la instancia de pelicula en la base de datos
		sesion.save(miPelicula);
		// Confirmamos la transacción
		tx.commit();
		// Cerramos la sesión
		sesion.close();
		// Mostramos mensaje indicando que se ha insertado el registro
		System.out.println("Registro Insertado");
		// Cerramos la conexión del SessionFactory()
		conexion.close();
	}
}
```

5.- Ejecutando HQL desde Java.

La clase Query permite realizar consultas HQL, devolviendo el resultado como objetos Java. La consultas son creadas mediante el método createQuery() de la clase Session. El método list() devuelve una colección con los objetos devueltos por la consulta HQL.

Ejemplo:

```
package consultahql;
import java.util.Iterator;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import modelo.*;
import controlador.*;
import java.util.logging.*;
```

```
public class ConsultaHQL {
    public static void main(String[] args) {
         // Desactivamos los mensajes de hibérnate de la consola
        Logger.getLogger("org.hibernate").setLevel(Level.SEVERE);
        // Creamos una cadena con la sentencia HQL que queremos ejecutar
        String sentencia = "select p from Peliculas p where titulo like
'%de%'";
        // Declaramos una variable para almacenar un instancia de Pelicula
        Peliculas miPelicula;
        // Creamos la conexión con la base de datos
        SessionFactory conexion = NewHibernateUtil.getSessionFactory();
        // Abrimos la sesión para poder realizar una transacción
        // con la base de datos
        Session sesion = conexion.openSession();
        // Comenzamos la transacción
        Transaction tx = sesion.beginTransaction();
        // Ejecutamos la consulta HQL
        Query consulta = sesion.createQuery(sentencia);
        // Volcamos los registros de la consulta en una Lista
        List registros = consulta.list();
        // Confirmamos la transacción
        tx.commit();
        // Cerramos la sesión
        sesion.close();
        // cerramos la conexión
        conexion.close();
        // Volcamos los registros para recorrerlos con un Iterator
        Iterator resultado = registros.iterator();
        // Mientras haya registros ejecuta el bucle
        while (resultado.hasNext()){
            // Guarda el registro en la instancia miPelicula
            miPelicula = (Peliculas)resultado.next();
            // Muestra el título de la película
            System.out.println(miPelicula.getTitulo());
        }
    }
}
```