Tutorial de Hibernate

Elaboró: Ismael Nuñez

Índice de contenido

Introducción

Al trabajar con programación orientada a objetos en muchas ocasiones nos encontramos con el problema de hacer persistir los objetos en una base de datos relacional, esto nos involucra en el problema de que el mundo relacional y el orientado a objetos no son del todo compatibles.

Para evidenciar este problema imaginemos que tenemos que hacer persistir un solo objeto en una tabla relacional, esto evidentemente involucra la creación de un DAO (Data Access Object) que nos permita hacer persistir a este objeto. Este DAO debe tener la implementación de código SQL para tener acceso a la base de datos en la queremos realizar la persistencia y este código debe permitir almacenar en los campos de la tabla todo el estado del objeto

Ejemplo:

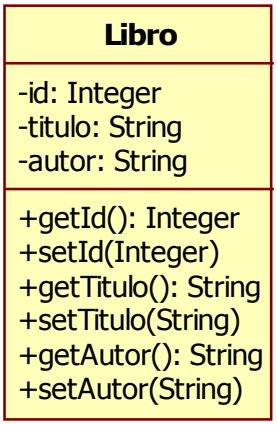

Ilustración 1: Clase Libro

La Ilustración 1 muestra el objeto de dominio `Libro` que deseamos hacer persistir. Para esto en el objeto DAO deberíamos tener implementado el método `create` que deberia ser similar al siguiente código:

```
public boolean create(Libro libro){
      try {
            // Prepara el statement
            PreparedStatement statement =
            DatabaseManager.getConnection().prepareStatement(
            "INSERT INTO Libro VALUES ( ? , ? , ? )");
            statement.setInteger(1,libro.getId());
            statement.setString(2,libro.getTitulo());
            statement.setString(3,libro.getAutor());
            statement.executeUpdate();
```

```
            return true;
      } catch (SQLException e) {
            // Cacha excepcion
            e.printStackTrace();
            return false;
      }
}
```

En realidad hacer persistir un solo objeto en una tabla relacional es bastante sencillo, el problema es que en la vida real no solo se realiza persistencia de un solo objeto y mucho menos de varios objetos sin relación entre ellos, como se puede notar el realizar persistencia en bases de datos relacionales es bastante complejo.

Para solventar estos problemas existen frameworks que se encargan de realizar este mapeo de Objeto a Relacional, a estos frameworks se les denomina ORM (Object-Relational Mapping). En esta tutorial vamos a trabajar con Hibernate, un ORM open source bastante maduro.

Antes de entrar en el detalle de Hibernate vamos a mostrar como un ORM realiza el mapeo de Orientado a Relacional para que se observen las ventajas de utilizar estos frameworks.

- Un buen ORM permite tomar un objeto Java y hacerlo persistir de una forma similar a la siguiente:

  ```
  orm.save(object);
  ```

  la instrucción anterior debe generar todo el código SQL para realizar la persistencia del objeto

- El ORM debe permitir cargar un objeto Java complejo (con relaciones hacia otros objetos) de la base de datos a memoria:

  ```
  myObject = orm.load(MyObject.class,objectId);
  ```

- EL ORM debe permitir hacer consultas a las tablas de la base de datos:

  ```
  List myObjects = orm.find("FROM MyObject object WHERE object.property = 5");
  ```

Como se puede observar en estos ejemplos los métodos `save`, `load` y `find` tienen una correspondencia con los métodos `create` y `retrieve` que se implementan usualmente en un DAO, pero con la ventaja de que son mas sencillos e intuitivos de emplear y reducen fallas al crear todo el código SQL.

Herramientas necesarias

Hibernate

Se puede obtener Hibernate directamente de su pagina en Internet http://hibernate.org/, de aquí se puede obtener el zip de Hibernate 3.0.2 que es la versión que vamos a utilizar.

Después de obtener el zip lo vamos a crear una carpeta llamada `home` en la unidad `C:`, y en esta vamos a descomprimir el zip de Hibernate.

> Nota: El nombre y la localización de la carpeta es una sugerencia propuesta en este tutorial, si se cambia el nombre y la localización se deben ajustar las direcciones en los sitios en donde se solicite

HSQL

HSQL se puede obtener de la pagina http://hsqldb.sourceforge.net/ , de esta vamos a descargar la versión 1.7.3.3 que viene en zip.

Suponiendo que ya tenemos Hibernate vamos a descomprimir el zip de HSQL en la carpeta `home` (o en la misa en donde se descomprimió Hibernate).

Eclipse

Eclipse es la herramienta que vamos a emplear para realizar el desarrollo de nuestras aplicaciones, este se puede obtener de la pagina http://www.eclipse.org/downloads/ . La versión que vamos a utilizar sera la 3.2 que también viene en un zip que vamos a descomprimir en `C:`.

Derby

Derby es una base de datos embebida, open source que utilizaremos en lugar de HSQL en algunos de los ejemplos de este tutorial. Se puede obtener de la pagina http://db.apache.org/derby/releases/release-10.1.2.1.cgi. Para este tutorial utilizaremos la versión 10.1.2.1 que se obtiene en un zip de la pagina antes mencionada.

Aplicación sencilla usando Hibernate

Objetivo de la aplicación:

En esta aplicación vamos a crear una clase Libro que sera nuestro objeto de dominio y crearemos una tabla LIBRO usando Hibernate para realizar 2 operaciones sencillas: insertar un libro y ver la lista de libros en la base de datos.

Crear el proyecto Java con Eclipse

Vamos a crear la carpeta Hibernate en la carpeta hsqldb (que se crea al descomprimir HSQL) y vamos a abrir Eclipse y darle como dirección de workspace la dirección de la carpeta que creamos

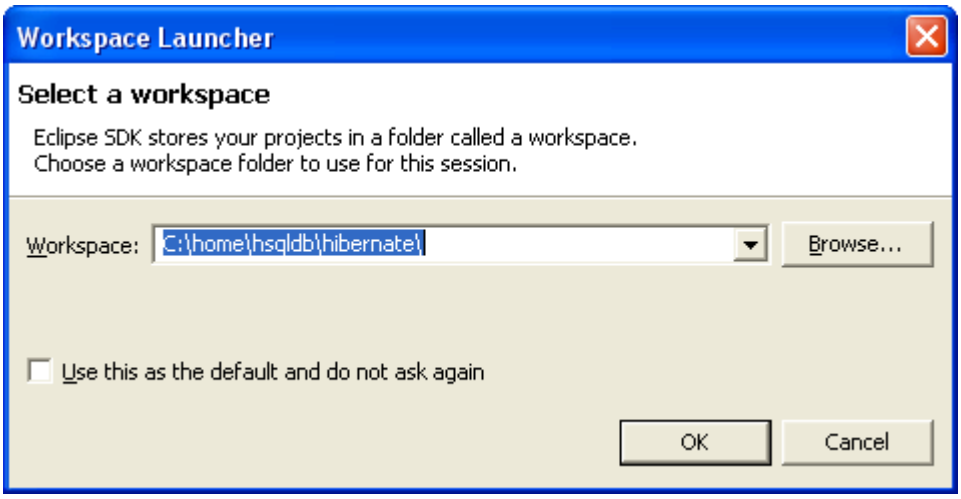

Ilustración 2: Selección de workspace

Una vez en Eclipse vamos a crear un nuevo proyecto al que llamaremos PruebaHibernate. Al crearla le agregamos la carpeta de fuentes src y el package mx.uam.hibernate.

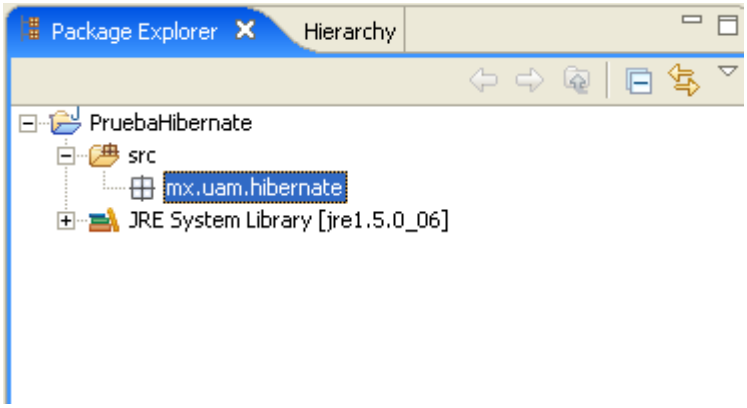

Ilustración 3: árbol del proyecto

Ahora vamos a crear los directorios lib en donde vamos a agregar las librerías que vamos a ocupar y data en donde HSQL genera una bitácora de las operaciones hechas en la base de datos, estos directorios se crean en la raíz del proyecto.

En el directorio de lib vamos a copiar del directorio lib de Hibernate los siguientes jars:

- cglib-2.1
- commons-logging-1.0.4
- hibernate3
- jta
- commons-collections-2.1.1
- dom4j-1.6
- jdbc2_0-stdext
- log4j-1.2.9
- asm
- antlr-2.7.5H3

y del directorio lib de hsql el jar hsql.jar

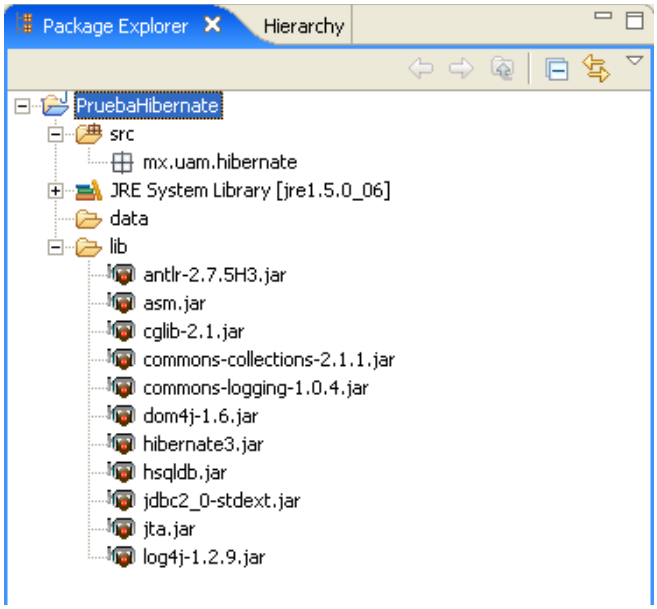

Ilustración 4: El directorio lib con los jars necesarios

Configuración del Build Path

Hasta este momento ya tenemos los jars necesarios para trabajar, ahora hay que decirle a Eclipse que los agregue a su build path para que puede trabajar con ellos. Para hacer esto seleccionamos el proyecto y sobre el damos click con el botón derecho y seleccionamos la opción *Build Path->Configure Build* Path , esto provoca que emerja el dialogo *Properties for PruebaHibernateen* la sección de *Java Build Path*, en esta sección seleccionamos la opción de *add JARs,* y en el dialogo que emerge seleccionamos del directorio lib de PruebaHibernate uno de los jars que ahí aparecen y presionamos *OK* , esto agrega el jar al build path.

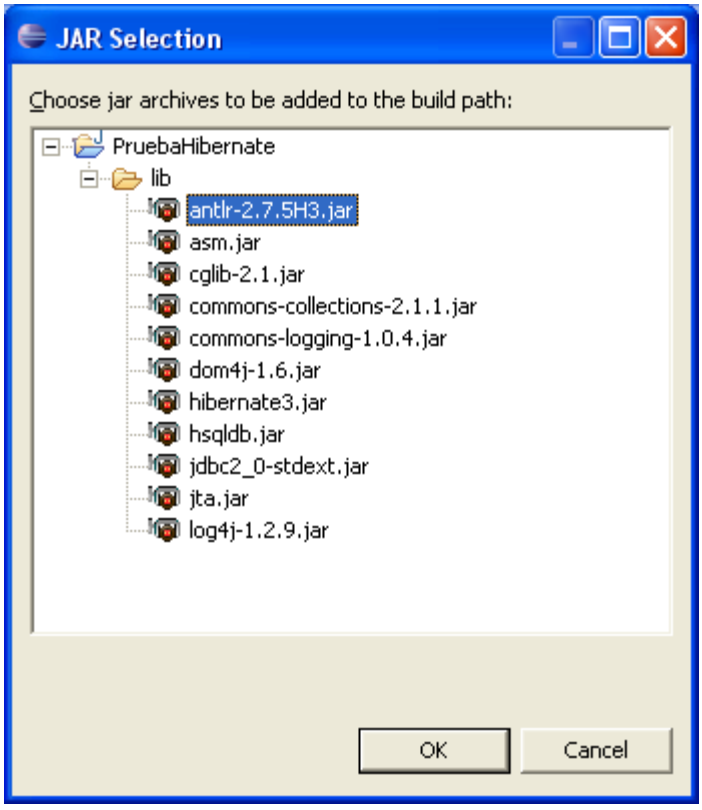

Ilustración 5: Selección de los JARS

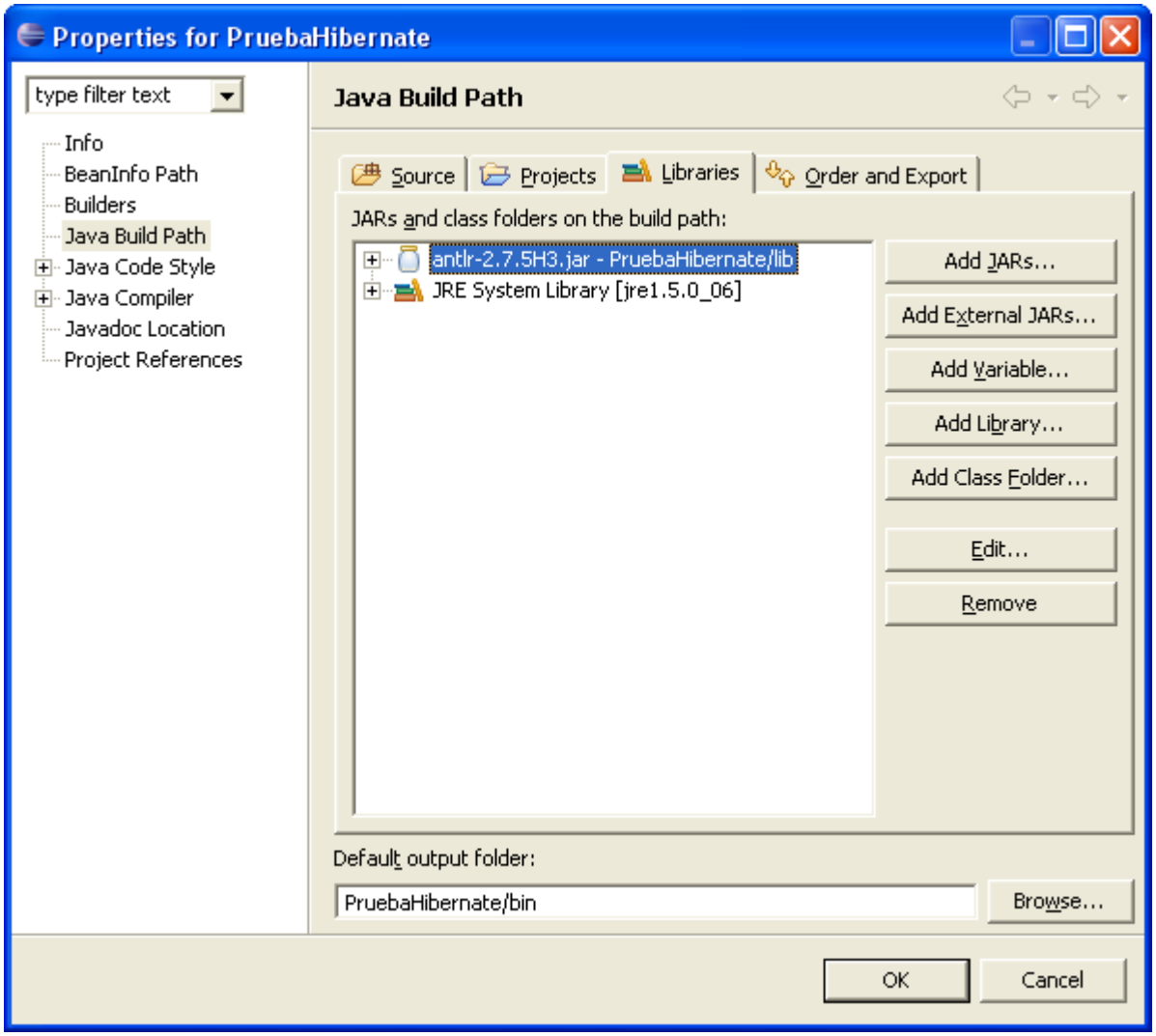

Ilustración 6: JAR agrega al Build Path

Realizamos la misma operación para todos los jar que se encuentren en el directorio lib y una vez agregados presionamos *OK*.

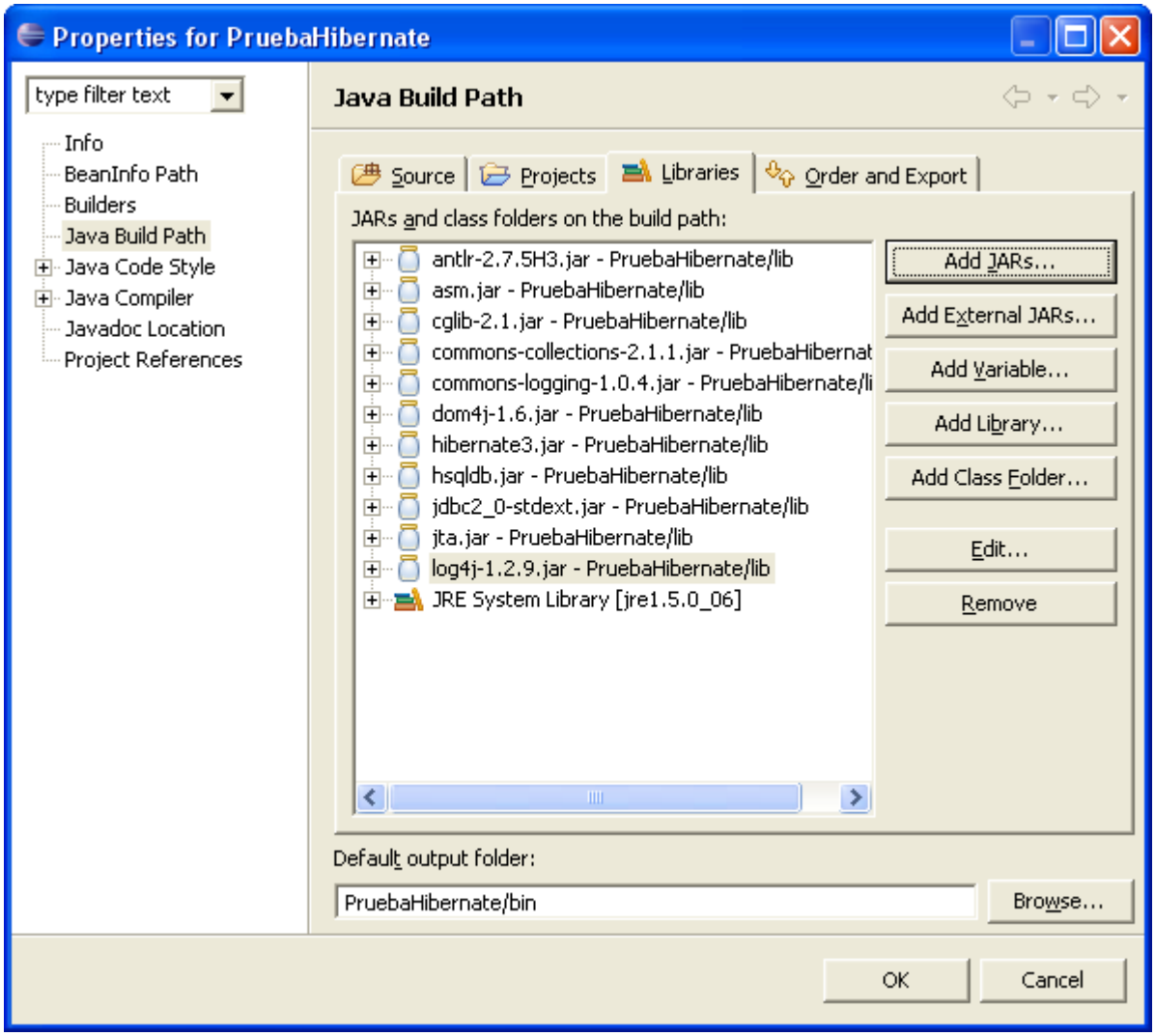

Ilustración 7: JARs agregados al Build Path

Al terminar tendremos una lista de jars debajo del JRE.

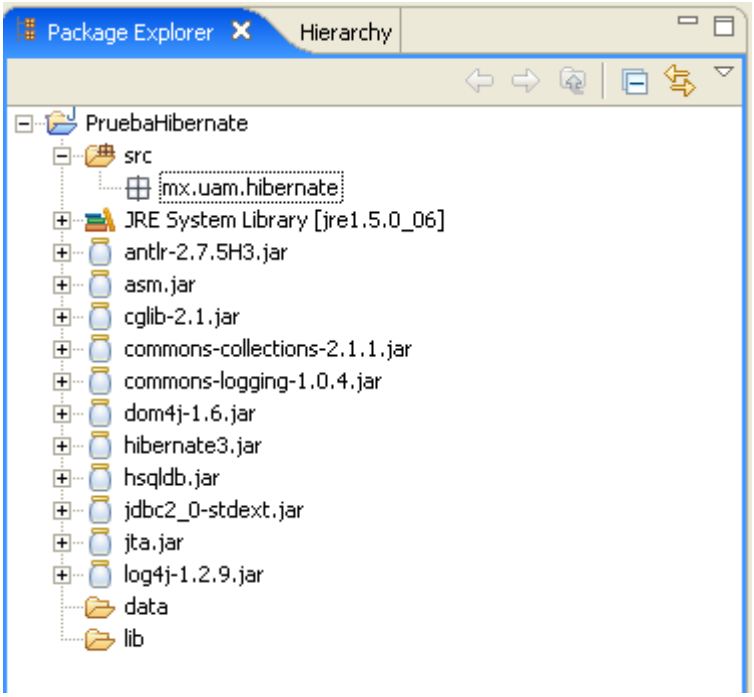

Ilustración 8: JARs agregados al Build Path

Creación de las clases Java

Para probar el funcionamiento de Hibernate vamos a crear 2 clases, 1 sera el objeto de dominio y la otra el objeto encargado de la persistencia usando Hibernate

Clase del objeto de dominio

Esta clase es una clase que modela la clase mostrada en la Ilustración 1, lo primero que hacemos es crear la clase con el wizard de eclipse.

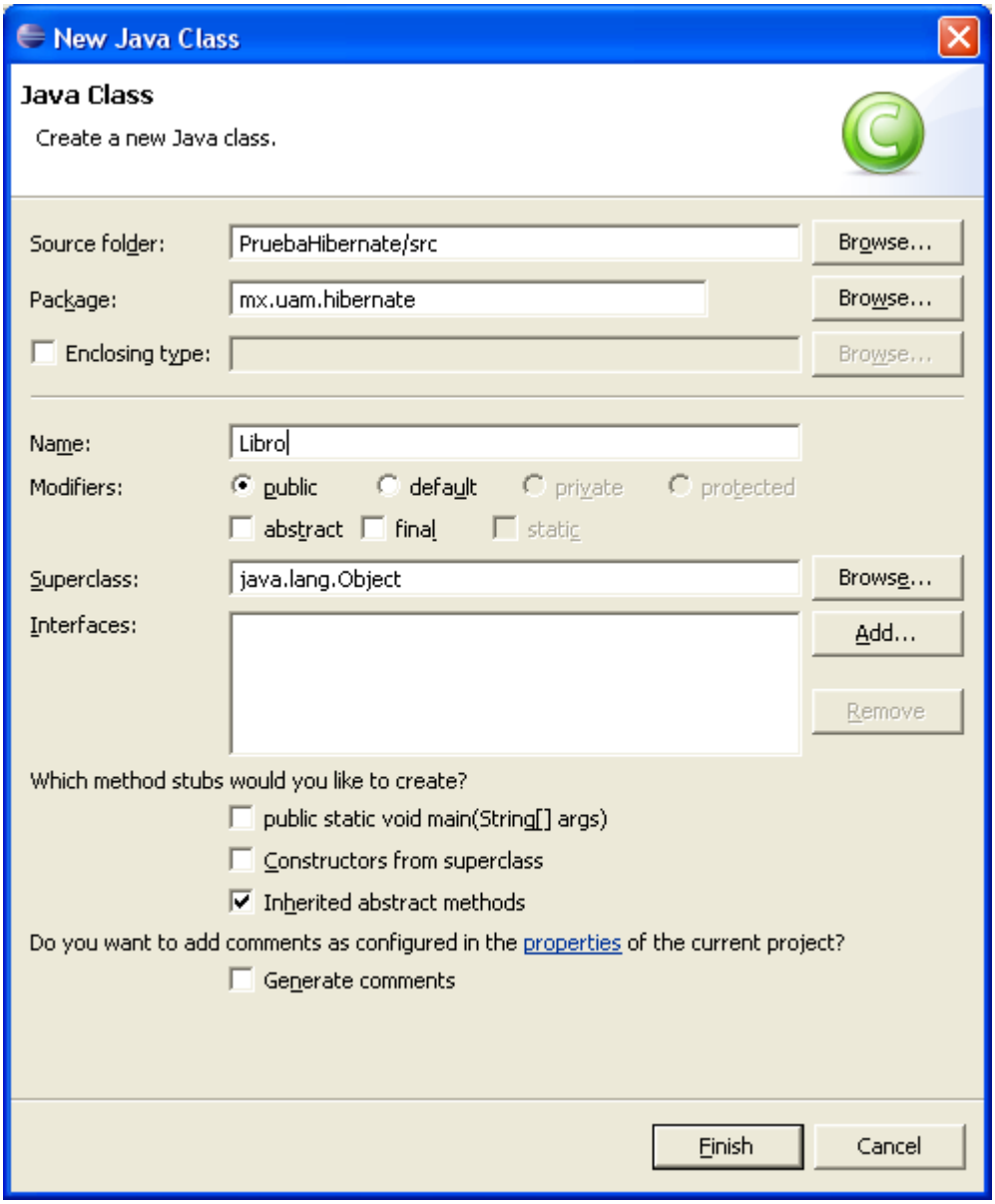

Ilustración 9: Wizard para crear la clase Libro

En esta clase agregamos el siguiente código:

```
package mx.uam.hibernate;

public class Libro {

	private Long id;
	private String titulo;
```

```
	private String autor;

	public String getAutor() {
		return autor;
	}
	public void setAutor(String autor) {
		this.autor = autor;
	}
	public Long getId() {
		return id;
	}
	private void setId(Long id) {
		this.id = id;
	}
	public String getTitulo() {
		return titulo;
	}
	public void setTitulo(String titulo) {
		this.titulo = titulo;
	}

}
```

Cabe hacer notar que el método `setId` es privado, esto es por que solo Hibernate debe encargarse de administrar los id's de los objetos Libro, también se debe notar que este método marcara un warning por que ningún otro método de la clase lo esta utilizando.

Otra cosa que vale la pena mencionar es que no se debe declarar la variable id como un tipo de datos primitivo, esto por facilidad de manejo para Hibernate.

Clase de Manejo del objeto de dominio

Esta clase se encarga de administrar al objeto de dominio, primero con el wizard de eclipse creamos la clase ManejadorLibro

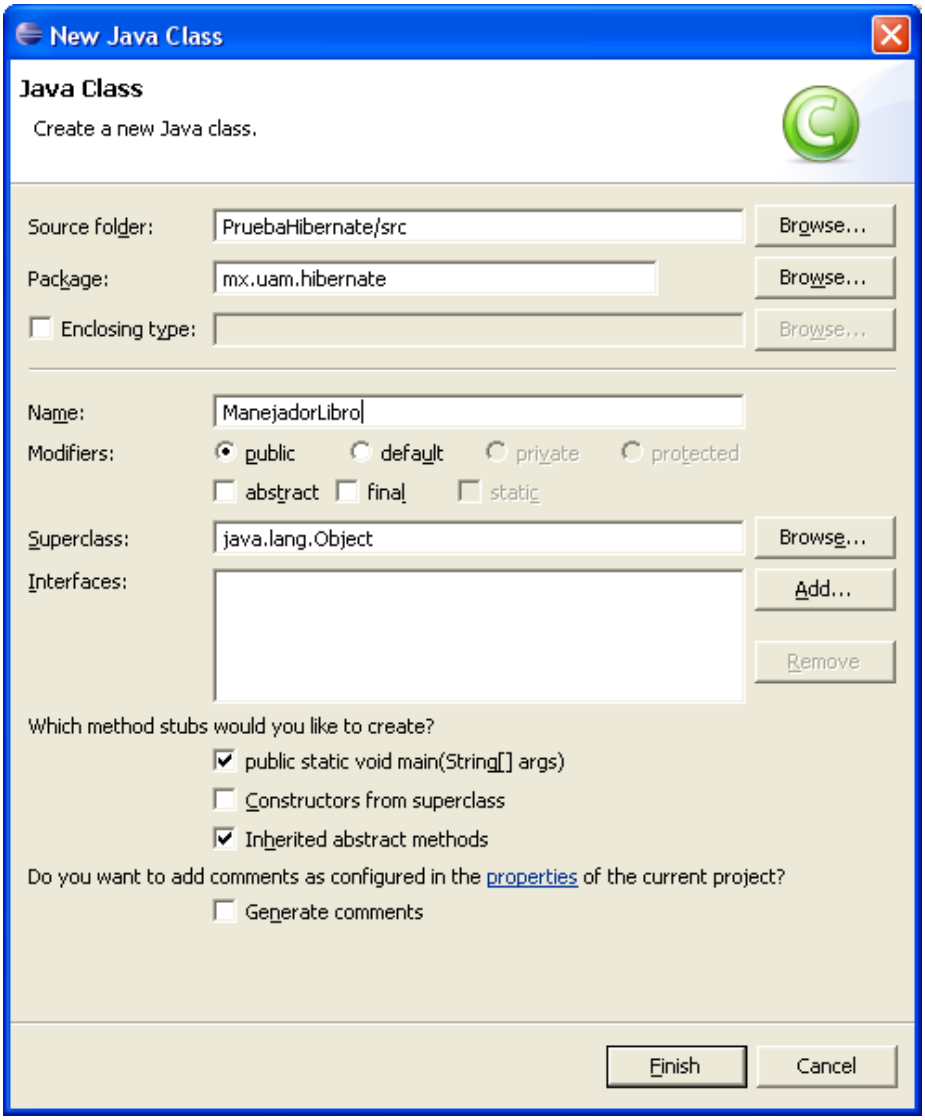

Ilustración 10: wizard para crear la clase ManejadorLibro

En esta clase agregamos el siguiente código:

```
package mx.uam.hibernate;

import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
```

```
import org.hibernate.cfg.Configuration;
import org.hibernate.classic.Session;


public class ManejadorLibro {

      private SessionFactory sessionFactory;

    public ManejadorLibro() {
        try {
            System.out.println("Inicalizando Hibernate");
            sessionFactory = new Configuration().configure().buildSessionFactory();
            System.out.println("terminado la inicializacion de Hibernate");
        } catch (HibernateException e) {
            e.printStackTrace();
        }
    }

    private void agregaLibro(String titulo, String autor) {
        try {
            Session session = sessionFactory.openSession();
            Transaction tx = session.beginTransaction();

            Libro libro = new Libro();
            libro.setTitulo(titulo);
            libro.setAutor(autor);

            session.save(libro);

            tx.commit();
            session.close();
        } catch (HibernateException e) {
            e.printStackTrace();
        }
    }
      private List listaLibros() {
```

```
        try {
            Session session = sessionFactory.openSession();
            Transaction tx = session.beginTransaction();

            List result = session.find("from Libro");

            tx.commit();
            session.close();

            return result;
        } catch (HibernateException e) {
            throw new RuntimeException(e.getMessage());
        }
    }
      public static void main(String[] args) {

            ManejadorLibro manejadorLibro = new ManejadorLibro();
        if (args[0].equals("agrega")) {
            if(args[1] != null && args[2] != null){
                  manejadorLibro.agregaLibro(args[1],args[2]);
            }else{
                  System.out.println("Faltan parametros");
            }
        } else if (args[0].equals("lista")) {
            List libros = manejadorLibro.listaLibros();
            for (int i = 0; i<libros.size(); i++) {
                Libro libro = (Libro) libros.get(i);
                            System.out.println("Libro:  "+libro.getTitulo()+",  Autor:
"+libro.getAutor());
            }
        }
        System.exit(0);

      }
}
```

Esta clase recibe de la linea de comandos 2 tipos de comandos:

- agrega titulo autor
- lista

cabe mencionar que no se han hecho todas las validaciones por la simplicidad de la aplicación, pero esto no evita que se puedan hacer.

Configuración de Hibernate

Hasta ahora solo hemos mencionado algunos aspectos de Hibernate pero no hemos hecho ninguna configuración del servicio, en este momento vamos a configurar a Hibernate para que conozca al objeto que debe hacer persistir y en donde lo hará persistir.

El archivo hibernate.cfg.xml

En directorio src creamos el archivo hibernate.cfg.xml y agregamos el siguiente codigo:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
   PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <property
name="hibernate.connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="hibernate.connection.url">jdbc:hsqldb:data/test</property>
        <property name="hibernate.connection.username">sa</property>
        <property name="hibernate.connection.password"></property>
        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
        <property name="show_sql">true</property>
        <property name="transaction.factory_class">
             org.hibernate.transaction.JDBCTransactionFactory
        </property>
        <property name="hibernate.cache.provider_class">
             org.hibernate.cache.HashtableCacheProvider
        </property>
        <property name="hibernate.hbm2ddl.auto">update</property>

        <mapping resource="mx/uam/hibernate/data/Libro.hbm.xml"/>

    </session-factory>

</hibernate-configuration>
```

Este archivo contiene varios tags, pero lo importante es lo que contiene el tag hibernate-configuration, en este tag aparece otro llamado session-factory que contiene, en orden de aparición:

- La clase del manejador de JDBC
- La URL del la conexión con JDBC

- El nombre de usuario de la base da datos
- El password del usuario de la base de datos
- El dialecto que Hibernate usara para realizar la persistencia del objeto
- Bandera para poder observar el código SQL
- La clase que se empleara para el manejo de las transacciones
- La clase que provee la implementación de la interfaz de CacheProvider
- Automáticamente actualiza los esquemas de la base de datos cuando se inicia y se apaga. Puede tomar 3 valores: create, update y create-drop
- El mapa que describe al objeto y su localización

El archivo Libro.hbm.xml

Este archivo le dice a Hibernate como es el objeto que debe persistir. Este archivo se debe colocar en el directorio que se dijo en el archivo de configuración de Hibernate, para esto debemos crear un directorio en el package mx.uam.hibernate llamado data y crear un archivo llamado Libro.hbm.xml con la siguiente información:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//hibernate/hibernate Mapping DTD 2.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>

        <class name="mx.uam.hibernate.Libro" table="LIBRO">
                <id name="id" column="uid" type="long">
                        <generator class="increment"/>
                </id>
                <property name="titulo" type="string"/>
                <property name="autor" type="string"/>
        </class>

</hibernate-mapping>
```

Este documento contiene el tag hibernate-mapping que contiene el tag class que es el que contiene la información de la estructura del objeto, esta información es:

- El nombre de la clase y la tabla en la que se hará la persistencia
- El tag id que dice a que columna se va a mapear, su tipo y la forma en que se genera el identificador
- Las propiedades de la clase y su tipo

Cabe mencionar que Hibernate puede generar de varias formas un identificador y que en las propiedades se pueden especificar otras cosas tales como: el nombre de la columna, la integridad referencial, etc.

Creación del archivo log2j.properties

La librería log2j sirve para realizar trazas en una aplicación, estas trazas las puede enviar directamente a una bitácora en disco o en consola, para realizar cualquiera de estas operaciones es necesario configurar un archivo llamado log2j.properties, para el objetivo de esta aplicación solo es necesario se requiere que se cree este archivo en el directorio de src y se agregue el siguiente texto:

```
log4j.rootCategory=INFO, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-5p - %m%n
```

Hasta este momento deberíamos tener un proyecto similar al siguiente:

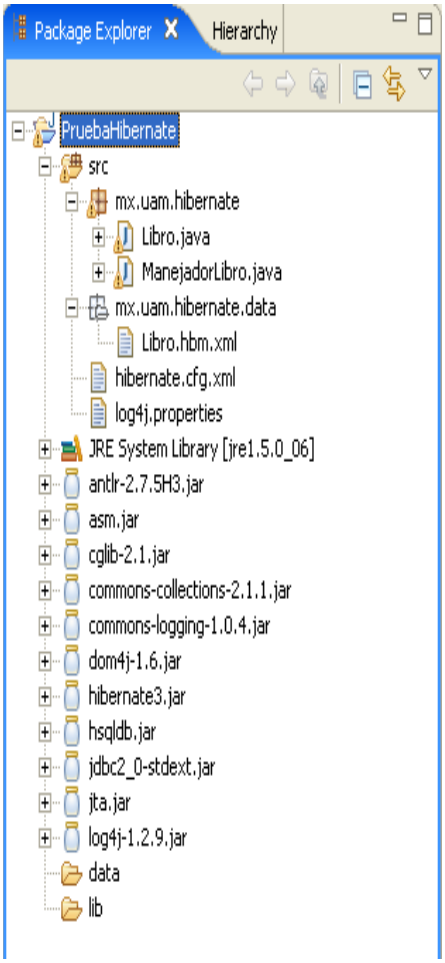

Ilustración 11: Vista del proyecto

Automatización del proceso de construcción con ANT

En el directorio raíz de nuestra aplicación vamos a crear un archivo build.xml para que ANT automatice el proceso de construcción de la aplicación.

En este archivo vamos a requerir de las siguientes tareas:

- init: Declaración de las propiedades que empleara el sistema
- prepare: Creación de los directorios
- copy-resources: Copia los archivos de configuración de Hibernate y log2j al directorio bin del proyecto
- compile: Compila el proyecto
- run: Corre la aplicación y agrega los parámetros necesarios

El archivo build.xml debe verse de la siguiente manera:

```
<project name="Prueba de hibernate" default="run">
      <target name="init">
            <property name="src.dir" value="src"/>
          <property name="classes.dir" value="bin"/>
          <property name="lib.dir" value="lib"/>
            <property name="mainclass" value="mx.uam.hibernate.ManejadorLibro"/>
      <path id="classpath">
            <pathelement location="${classes.dir}" />
            <fileset dir="${lib.dir}">
                  <include name="*.jar" />
            </fileset>
       </path>
      </target>
      <target name="prepare" depends="init">
            <mkdir dir="${classes.dir}"/>
            <mkdir dir="${lib.dir}"/>
      </target>
    <target name="compile" depends="copy-resources">
       <javac srcdir="${src.dir}" destdir="${classes.dir}">
               <classpath refid="classpath" />
       </javac>
    </target>
    <target name="copy-resources" depends="prepare">
        <copy todir="${classes.dir}">
            <fileset dir="${src.dir}">
                <exclude name="**/*.java"/>
```

```
            </fileset>
        </copy>
    </target>
      <target name="run" depends="compile">
            <java classname="${mainclass}" fork="true">
                  <arg value="agrega"/>
                  <arg value="Tutorial"/>
                  <arg value="IngSW"/>
                  <classpath refid="classpath" />
            </java>
      </target>
</project>
```

Correr la aplicación

Para poder correr la aplicación vamos a abrir la vista de ANT, para esto vamos a al menú principal y seleccionamos Window->Show View->Ant, después seleccionamos el archivo build.xml y lo arrastramos hasta colocarlo en la vista de ANT.

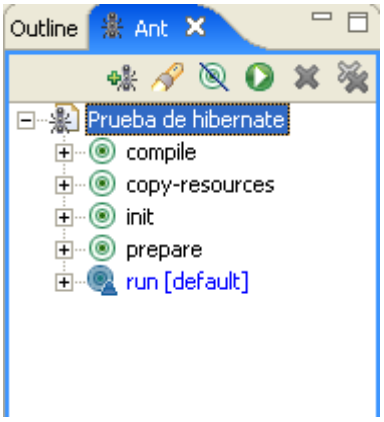

Ilustración 12: Vista de ANT con las tareas declaradas en el archivo build.xml

El siguiente para es dar doble click cobre la tarea “run” y en la consola debe desplegarse algo similar a:

```
Buildfile: C:\home\hsqldb\hibernate\PruebaHibernate\build.xml
init:
prepare:
copy-resources:
compile:
run:
     [java] Inicalizando Hibernate
     [java] INFO  - Hibernate 3.0.2
```

```
     [java] INFO  - hibernate.properties not found
     [java] INFO  - using CGLIB reflection optimizer
     [java] INFO  - using JDK 1.4 java.sql.Timestamp handling
     [java] INFO  - configuring from resource: /hibernate.cfg.xml
     [java] INFO  - Configuration resource: /hibernate.cfg.xml
     [java] INFO  - Mapping resource: mx/uam/hibernate/data/Libro.hbm.xml
     [java] INFO  - Mapping class: mx.uam.hibernate.Libro -> LIBRO
     [java] INFO  - Configured SessionFactory: null
     [java] INFO  - processing extends queue
     [java] INFO  - processing collection mappings
     [java] INFO  - processing association property references
     [java] INFO  - processing foreign key constraints
     [java] INFO  - Using Hibernate built-in connection pool (not for production
use!)
     [java] INFO  - Hibernate connection pool size: 20
     [java] INFO  - autocommit mode: false
     [java] INFO  - using driver: org.hsqldb.jdbcDriver at URL:
jdbc:hsqldb:data/test
     [java] INFO  - connection properties: {user=sa, password=****}
     [java] INFO  - RDBMS: HSQL Database Engine, version: 1.7.3
     [java] INFO  - JDBC driver: HSQL Database Engine Driver, version: 1.7.3
     [java] INFO  - Using dialect: org.hibernate.dialect.HSQLDialect
     [java] INFO  - JDBC batch size: 15
     [java] INFO  - JDBC batch updates for versioned data: disabled
     [java] INFO  - Scrollable result sets: enabled
     [java] INFO  - JDBC3 getGeneratedKeys(): disabled
     [java] INFO  - Aggressive release : disabled
     [java] INFO  - Default batch fetch size: 1
     [java] INFO  - Generate SQL with comments: disabled
     [java] INFO  - Order SQL updates by primary key: disabled
     [java] INFO  - Query translator:
org.hibernate.hql.ast.ASTQueryTranslatorFactory
     [java] INFO  - Using ASTQueryTranslatorFactory
     [java] INFO  - Query language substitutions: {}
     [java] INFO  - Transaction strategy:
org.hibernate.transaction.JDBCTransactionFactory
     [java] INFO  - No TransactionManagerLookup configured (in JTA environment, use
of read-write or transactional second-level cache is not recommended)
     [java] INFO  - Automatic flush during beforeCompletion(): disabled
     [java] INFO  - Automatic session close at end of transaction: disabled
     [java] INFO  - Second-level cache: enabled
     [java] INFO  - Query cache: disabled
     [java] INFO  - Cache provider: org.hibernate.cache.HashtableCacheProvider
     [java] INFO  - Optimize cache for minimal puts: disabled
     [java] INFO  - Structured second-level cache entries: enabled
     [java] INFO  - Echoing all SQL to stdout
     [java] INFO  - Statistics: disabled
     [java] INFO  - Deleted entity synthetic identifier rollback: disabled
     [java] INFO  - Default entity-mode: pojo
     [java] INFO  - building session factory
     [java] INFO  - Not binding factory to JNDI, no JNDI name configured
     [java] INFO  - Using dialect: org.hibernate.dialect.HSQLDialect
     [java] INFO  - Using Hibernate built-in connection pool (not for production
use!)
     [java] INFO  - Hibernate connection pool size: 20
```

```
     [java] INFO  - autocommit mode: false
     [java] INFO  - using driver: org.hsqldb.jdbcDriver at URL:
jdbc:hsqldb:data/test
     [java] INFO  - connection properties: {user=sa, password=****}
     [java] INFO  - Running hbm2ddl schema update
     [java] INFO  - fetching database metadata
     [java] INFO  - updating schema
     [java] INFO  - processing extends queue
     [java] INFO  - processing collection mappings
     [java] INFO  - processing association property references
     [java] INFO  - processing foreign key constraints
     [java] INFO  - table found: LIBRO
     [java] INFO  - columns: [titulo, uid, autor]
     [java] INFO  - foreign keys: []
     [java] INFO  - indexes: [sys_pk_libro]
     [java] INFO  - schema update complete
     [java] INFO  - cleaning up connection pool: jdbc:hsqldb:data/test
     [java] INFO  - Checking 0 named queries
     [java] terminado la inicializacion de Hibernate
     [java] Hibernate: insert into LIBRO (titulo, autor, uid) values (?, ?, ?)
BUILD SUCCESSFUL
Total time: 3 seconds
```

Con esto hemos agregado una entrada a la tabla LIBRO con la informacion:

Titulo: Tutorial

Autor: IngSW

Esta es la información que le pasamos al programa en las lineas de ANT de la tarea run:

```
<arg value="agrega"/>
<arg value="Tutorial"/>
<arg value="IngSW"/>
```

Para observar que esta entrada en la tabla ha sido persistida por Hibernate cambiaremos estas lineas por:

```
<arg value="lista"/>
```

Ahora damos doble click en la tarea run y debe aparecer algo similar a:

```
Buildfile: C:\home\hsqldb\hibernate\PruebaHibernate\build.xml
init:
prepare:
copy-resources:
compile:
run:
     [java] Inicalizando Hibernate
     [java] INFO  - Hibernate 3.0.2
     [java] INFO  - hibernate.properties not found
     [java] INFO  - using CGLIB reflection optimizer
     [java] INFO  - using JDK 1.4 java.sql.Timestamp handling
     [java] INFO  - configuring from resource: /hibernate.cfg.xml
     [java] INFO  - Configuration resource: /hibernate.cfg.xml
     [java] INFO  - Mapping resource: mx/uam/hibernate/data/Libro.hbm.xml
     [java] INFO  - Mapping class: mx.uam.hibernate.Libro -> LIBRO
```

```
     [java] INFO  - Configured SessionFactory: null
     [java] INFO  - processing extends queue
     [java] INFO  - processing collection mappings
     [java] INFO  - processing association property references
     [java] INFO  - processing foreign key constraints
     [java] INFO  - Using Hibernate built-in connection pool (not for production
use!)
     [java] INFO  - Hibernate connection pool size: 20
     [java] INFO  - autocommit mode: false
     [java] INFO  - using driver: org.hsqldb.jdbcDriver at URL:
jdbc:hsqldb:data/test
     [java] INFO  - connection properties: {user=sa, password=****}
     [java] INFO  - RDBMS: HSQL Database Engine, version: 1.7.3
     [java] INFO  - JDBC driver: HSQL Database Engine Driver, version: 1.7.3
     [java] INFO  - Using dialect: org.hibernate.dialect.HSQLDialect
     [java] INFO  - JDBC batch size: 15
     [java] INFO  - JDBC batch updates for versioned data: disabled
     [java] INFO  - Scrollable result sets: enabled
     [java] INFO  - JDBC3 getGeneratedKeys(): disabled
     [java] INFO  - Aggressive release : disabled
     [java] INFO  - Default batch fetch size: 1
     [java] INFO  - Generate SQL with comments: disabled
     [java] INFO  - Order SQL updates by primary key: disabled
     [java] INFO  - Query translator:
org.hibernate.hql.ast.ASTQueryTranslatorFactory
     [java] INFO  - Using ASTQueryTranslatorFactory
     [java] INFO  - Query language substitutions: {}
     [java] INFO  - Transaction strategy:
org.hibernate.transaction.JDBCTransactionFactory
     [java] INFO  - No TransactionManagerLookup configured (in JTA environment, use
of read-write or transactional second-level cache is not recommended)
     [java] INFO  - Automatic flush during beforeCompletion(): disabled
     [java] INFO  - Automatic session close at end of transaction: disabled
     [java] INFO  - Second-level cache: enabled
     [java] INFO  - Query cache: disabled
     [java] INFO  - Cache provider: org.hibernate.cache.HashtableCacheProvider
     [java] INFO  - Optimize cache for minimal puts: disabled
     [java] INFO  - Structured second-level cache entries: enabled
     [java] INFO  - Echoing all SQL to stdout
     [java] INFO  - Statistics: disabled
     [java] INFO  - Deleted entity synthetic identifier rollback: disabled
     [java] INFO  - Default entity-mode: pojo
     [java] INFO  - building session factory
     [java] INFO  - Not binding factory to JNDI, no JNDI name configured
     [java] INFO  - Using dialect: org.hibernate.dialect.HSQLDialect
     [java] INFO  - Using Hibernate built-in connection pool (not for production
use!)
     [java] INFO  - Hibernate connection pool size: 20
     [java] INFO  - autocommit mode: false
     [java] INFO  - using driver: org.hsqldb.jdbcDriver at URL:
jdbc:hsqldb:data/test
     [java] INFO  - connection properties: {user=sa, password=****}
     [java] INFO  - Running hbm2ddl schema update
     [java] INFO  - fetching database metadata
     [java] INFO  - updating schema
```

```
     [java] INFO  - processing extends queue
     [java] INFO  - processing collection mappings
     [java] INFO  - processing association property references
     [java] INFO  - processing foreign key constraints
     [java] INFO  - table found: LIBRO
     [java] INFO  - columns: [titulo, uid, autor]
     [java] INFO  - foreign keys: []
     [java] INFO  - indexes: [sys_pk_libro]
     [java] INFO  - schema update complete
     [java] INFO  - cleaning up connection pool: jdbc:hsqldb:data/test
     [java] INFO  - Checking 0 named queries
     [java] terminado la inicializacion de Hibernate
     [java] Hibernate: select libro0_.uid as uid, libro0_.titulo as titulo0_,
libro0_.autor as autor0_ from LIBRO libro0_
     [java] Libro: Tutorial, Autor: IngSW
BUILD SUCCESSFUL
Total time: 3 seconds
```

Aplicación sencilla utilizando Hibernate y Derby

En el ejemplo anterior utilizamos HSQL como el medio para hacer la persistencia de nuestro objeto de dominio `Libro`, pero esto no significa que sea la única forma de lograr hacer persistir a nuestros objetos. Hibernate conoce varios dialectos de SQL que permiten realizan la conexión entre Hibernate y otros manejadores de bases de datos.

En este caso vamos a empleas Derby para realizar la persistencia de nuestro objeto de dominio. El primer paso es obtener Derby de la pagina que ya se menciono con anterioridad, después se descomprime el zip, dentro de la carpeta `db-derby-10.1.2.1-bin` que se genera al descomprimir el jars se encuentra la carpeta lib que contiene varios jars, de esta carpeta copias el jar derby.jar a la carpeta lib de nuestro proyecto.

Ahora abrimos Eclipse y agregamos al Build Path el jar que acabamos de copiar.

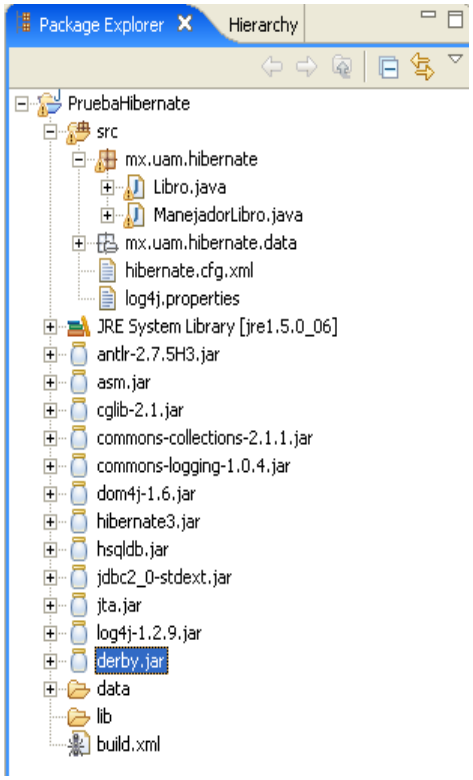

Ilustración 13: JAR de Derby en nuestro proyecto

Configuración de el archivo hibernate.cfg.xml

Para poder hacer uso de Derby requerimos decirle a Hibernate como comunicarse con Derby, para lograr esto requerimos modificar el archivo de configuración de Hibernate.

Para realizar la comunicación con Derby se necesitan modificar las siguientes propiedades con los valores dados:

- hibernate.connection.driver_class -> org.apache.derby.jdbc.EmbeddedDriver
- hibernate.connection.url -> jdbc:derby:build/derby/hibernate;create=true
- dialect -> org.hibernate.dialect.DerbyDialect

El archivo de configuración de Hibernate deberia quedar parecido a:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
   PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <property
name="hibernate.connection.driver_class">org.apache.derby.jdbc.EmbeddedDriver</prop
erty>
        <property
name="hibernate.connection.url">jdbc:derby:build/derby/hibernate;create=true</prope
rty>
        <property name="hibernate.connection.username">sa</property>
        <property name="hibernate.connection.password"></property>
        <property name="dialect">org.hibernate.dialect.DerbyDialect</property>
        <property name="show_sql">true</property>
        <property name="transaction.factory_class">
             org.hibernate.transaction.JDBCTransactionFactory
        </property>
        <property name="hibernate.cache.provider_class">
             org.hibernate.cache.HashtableCacheProvider
        </property>
        <property name="hibernate.hbm2ddl.auto">update</property>

        <mapping resource="mx/uam/hibernate/data/Libro.hbm.xml"/>

    </session-factory>

</hibernate-configuration>
```

Lo que le decimos a Hibernate es:

- Que Driver utilice para comunicarse con Derby
- Que URL utilizara para comunicarse
- En que dialecto debe hablar

Esta es la única modificación que se le debe realizar a la aplicación para que funcione, después de esto debe funcionar tal y como funcionaba con HSQL, a excepción de que la información que contenía HSQL evidentemente no esta en Derby

Corriendo la aplicación

Una vez realizadas las modificaciones mencionadas modificamos los parámetros que se le pasan al programa para que agregue una entrada en la base de datos Derby.

Ejemplo:

```
<target name="run" depends="compile">
      <java classname="${mainclass}" fork="true">
            <arg value="agrega"/>
            <arg value="Tutorial"/>
            <arg value="IngSW"/>
            <classpath refid="classpath" />
      </java>
</target>
```

Ahora salvamos el archivo build.xml y en la vista de ANT damos doble click sobre la tarea run, esto debe generar una entrada en la base de datos y dar la siguiente traza:

```
Buildfile: C:\home\hsqldb\hibernate\PruebaHibernate\build.xml
init:
prepare:
copy-resources:
compile:
run:
     [java] Inicalizando Hibernate
     [java] INFO  - Hibernate 3.0.2
     [java] INFO  - hibernate.properties not found
     [java] INFO  - using CGLIB reflection optimizer
     [java] INFO  - using JDK 1.4 java.sql.Timestamp handling
     [java] INFO  - configuring from resource: /hibernate.cfg.xml
     [java] INFO  - Configuration resource: /hibernate.cfg.xml
     [java] INFO  - Mapping resource: mx/uam/hibernate/data/Libro.hbm.xml
     [java] INFO  - Mapping class: mx.uam.hibernate.Libro -> LIBRO
     [java] INFO  - Configured SessionFactory: null
     [java] INFO  - processing extends queue
     [java] INFO  - processing collection mappings
     [java] INFO  - processing association property references
     [java] INFO  - processing foreign key constraints
     [java] INFO  - Using Hibernate built-in connection pool (not for production
use!)
     [java] INFO  - Hibernate connection pool size: 20
     [java] INFO  - autocommit mode: false
     [java] INFO  - using driver: org.apache.derby.jdbc.EmbeddedDriver at URL:
jdbc:derby:build/derby/hibernate;create=true
     [java] INFO  - connection properties: {user=sa, password=****}
     [java] INFO  - RDBMS: Apache Derby, version: 10.1.2.1
     [java] INFO  - JDBC driver: Apache Derby Embedded JDBC Driver, version:
10.1.2.1
     [java] INFO  - Using dialect: org.hibernate.dialect.DerbyDialect
     [java] INFO  - Scrollable result sets: enabled
     [java] INFO  - JDBC3 getGeneratedKeys(): disabled
     [java] INFO  - Aggressive release : disabled
     [java] INFO  - Default batch fetch size: 1
     [java] INFO  - Generate SQL with comments: disabled
     [java] INFO  - Order SQL updates by primary key: disabled
     [java] INFO  - Query translator:
org.hibernate.hql.ast.ASTQueryTranslatorFactory
     [java] INFO  - Using ASTQueryTranslatorFactory
```

```
     [java] INFO  - Query language substitutions: {}
     [java] INFO  - Transaction strategy:
org.hibernate.transaction.JDBCTransactionFactory
     [java] INFO  - No TransactionManagerLookup configured (in JTA environment, use
of read-write or transactional second-level cache is not recommended)
     [java] INFO  - Automatic flush during beforeCompletion(): disabled
     [java] INFO  - Automatic session close at end of transaction: disabled
     [java] INFO  - Second-level cache: enabled
     [java] INFO  - Query cache: disabled
     [java] INFO  - Cache provider: org.hibernate.cache.HashtableCacheProvider
     [java] INFO  - Optimize cache for minimal puts: disabled
     [java] INFO  - Structured second-level cache entries: enabled
     [java] INFO  - Echoing all SQL to stdout
     [java] INFO  - Statistics: disabled
     [java] INFO  - Deleted entity synthetic identifier rollback: disabled
     [java] INFO  - Default entity-mode: pojo
     [java] INFO  - building session factory
     [java] INFO  - Not binding factory to JNDI, no JNDI name configured
     [java] INFO  - Using dialect: org.hibernate.dialect.DerbyDialect
     [java] INFO  - Using Hibernate built-in connection pool (not for production
use!)
     [java] INFO  - Hibernate connection pool size: 20
     [java] INFO  - autocommit mode: false
     [java] INFO  - using driver: org.apache.derby.jdbc.EmbeddedDriver at URL:
jdbc:derby:build/derby/hibernate;create=true
     [java] INFO  - connection properties: {user=sa, password=****}
     [java] INFO  - Running hbm2ddl schema update
     [java] INFO  - fetching database metadata
     [java] INFO  - updating schema
     [java] INFO  - processing extends queue
     [java] INFO  - processing collection mappings
     [java] INFO  - processing association property references
     [java] INFO  - processing foreign key constraints
     [java] INFO  - table not found: LIBRO
     [java] INFO  - table not found: LIBRO
     [java] INFO  - schema update complete
     [java] INFO  - cleaning up connection pool:
jdbc:derby:build/derby/hibernate;create=true
     [java] INFO  - Checking 0 named queries
     [java] terminado la inicializacion de Hibernate
     [java] Hibernate: insert into LIBRO (titulo, autor, uid) values (?, ?, ?)
BUILD SUCCESSFUL
Total time: 6 seconds
```

Ahora para probar que se agrego la entrada en la base de datos cambiando los argumentos de la tarea run por:

```
<arg value="lista"/>
```

Esto genera una traza parecida a la siguiente:

```
Buildfile: C:\home\hsqldb\hibernate\PruebaHibernate\build.xml

init:
prepare:
copy-resources:
compile:
run:
     [java] Inicalizando Hibernate
     [java] INFO  - Hibernate 3.0.2
     [java] INFO  - hibernate.properties not found
     [java] INFO  - using CGLIB reflection optimizer
     [java] INFO  - using JDK 1.4 java.sql.Timestamp handling
     [java] INFO  - configuring from resource: /hibernate.cfg.xml
     [java] INFO  - Configuration resource: /hibernate.cfg.xml
     [java] INFO  - Mapping resource: mx/uam/hibernate/data/Libro.hbm.xml
     [java] INFO  - Mapping class: mx.uam.hibernate.Libro -> LIBRO
     [java] INFO  - Configured SessionFactory: null
     [java] INFO  - processing extends queue
     [java] INFO  - processing collection mappings
     [java] INFO  - processing association property references
     [java] INFO  - processing foreign key constraints
     [java] INFO  - Using Hibernate built-in connection pool (not for production
use!)
     [java] INFO  - Hibernate connection pool size: 20
     [java] INFO  - autocommit mode: false
     [java] INFO  - using driver: org.apache.derby.jdbc.EmbeddedDriver at URL:
jdbc:derby:build/derby/hibernate;create=true
     [java] INFO  - connection properties: {user=sa, password=****}
     [java] INFO  - RDBMS: Apache Derby, version: 10.1.2.1
     [java] INFO  - JDBC driver: Apache Derby Embedded JDBC Driver, version:
10.1.2.1
     [java] INFO  - Using dialect: org.hibernate.dialect.DerbyDialect
     [java] INFO  - Scrollable result sets: enabled
     [java] INFO  - JDBC3 getGeneratedKeys(): disabled
     [java] INFO  - Aggressive release : disabled
     [java] INFO  - Default batch fetch size: 1
     [java] INFO  - Generate SQL with comments: disabled
     [java] INFO  - Order SQL updates by primary key: disabled
     [java] INFO  - Query translator:
org.hibernate.hql.ast.ASTQueryTranslatorFactory
     [java] INFO  - Using ASTQueryTranslatorFactory
     [java] INFO  - Query language substitutions: {}
     [java] INFO  - Transaction strategy:
org.hibernate.transaction.JDBCTransactionFactory
     [java] INFO  - No TransactionManagerLookup configured (in JTA environment, use
of read-write or transactional second-level cache is not recommended)
     [java] INFO  - Automatic flush during beforeCompletion(): disabled
     [java] INFO  - Automatic session close at end of transaction: disabled
     [java] INFO  - Second-level cache: enabled
     [java] INFO  - Query cache: disabled
     [java] INFO  - Cache provider: org.hibernate.cache.HashtableCacheProvider
     [java] INFO  - Optimize cache for minimal puts: disabled
     [java] INFO  - Structured second-level cache entries: enabled
     [java] INFO  - Echoing all SQL to stdout
     [java] INFO  - Statistics: disabled
     [java] INFO  - Deleted entity synthetic identifier rollback: disabled
```

```
     [java] INFO  - Default entity-mode: pojo
     [java] INFO  - building session factory
     [java] INFO  - Not binding factory to JNDI, no JNDI name configured
     [java] INFO  - Using dialect: org.hibernate.dialect.DerbyDialect
     [java] INFO  - Using Hibernate built-in connection pool (not for production
use!)
     [java] INFO  - Hibernate connection pool size: 20
     [java] INFO  - autocommit mode: false
     [java] INFO  - using driver: org.apache.derby.jdbc.EmbeddedDriver at URL:
jdbc:derby:build/derby/hibernate;create=true
     [java] INFO  - connection properties: {user=sa, password=****}
     [java] INFO  - Running hbm2ddl schema update
     [java] INFO  - fetching database metadata
     [java] INFO  - updating schema
     [java] INFO  - processing extends queue
     [java] INFO  - processing collection mappings
     [java] INFO  - processing association property references
     [java] INFO  - processing foreign key constraints
     [java] INFO  - table found: .SA.LIBRO
     [java] INFO  - columns: [titulo, uid, autor]
     [java] INFO  - foreign keys: []
     [java] INFO  - indexes: [sql1060907004226140]
     [java] INFO  - schema update complete
     [java] INFO  - cleaning up connection pool:
jdbc:derby:build/derby/hibernate;create=true
     [java] INFO  - Checking 0 named queries
     [java] terminado la inicializacion de Hibernate
     [java] Hibernate: select libro0_.uid as uid, libro0_.titulo as titulo0_,
libro0_.autor as autor0_ from LIBRO libro0_
     [java] INFO  - cleaning up connection pool:
jdbc:derby:build/derby/hibernate;create=true
     [java] WARN  - SQL Warning: 10000, SQLState: 01J01
     [java] WARN  - Database 'build/derby/hibernate' not created, connection made
to existing database instead.
     [java] Libro: Tutorial, Autor: IngSW
BUILD SUCCESSFUL
Total time: 5 seconds
```

Creando aplicaciones con Hibernate[1]

Hasta este momento hemos creado una aplicación sencilla empleando Hibernate, pero hasta ahora no hemos entrado en detalles de como están estructurados los archivos de mapeo de Hibernate, en esta sección vamos a detallar estos archivos y de que manera podemos hacer que dos o mas tablas interactúen.

1 Los datos tomados en esta sección están basados en el capitulo 6 del libro Pro Hibernate 3 que se encuentra en books24x7

Anatomía de los archivo de mapeo

Tipos de datos soportados por Hibernate

Tipo de dato de Hibernate	*Equivalencia a tipo de datos Java*
int	int, java.lang.Integer
short	short, java.lang.Short
long	long, java.lang.Long
float	float, java.lang.Float
double	double, java.lang.Double
character	char, java.lang.Character
byte	byte, java.lang.Byte
boolean	boolean, java.lang,Boolean
string	java.lang.Stirng
date, time	java.util.Date
calendar	java.util.Calendar
big_decimal	java.math.BigDecimal
big_integer	java.math.BigInteger
locale	java.util.Locale
timezone	java.util.TimeZone
currency	java.util.Currency
class	java.lang.Class
binary	byte[]
text	java.lang.String
serializable	java.io.Serializable
clob	java.sql.Clob
blob	java.sql.Blob

Tabla 1: Tipos de datos en Hibernate

Atributos del elemento <hibernate-mapping>

Este es el elemento central y permite establecer los comportamientos a los demás elementos

Atributo	*Valores*	*Default*	*Descripción*
auto-import	true,false	true	Permite usar nombres de clases no cualificados en los queries de Hibernate.
catalog			El catalogo de la base de datos al cual se aplican los queries
default-access		property	Tipo de acceso a los datos de la clase, puesto en property se utilizan los métodos de get y set con el valor asignado a name del elemento property
default-cascade		none	Define como los cambios en el objeto afectan a los objetos relacionados con el.
default-lazy	true,false	true	Define si la instanciación perezosa es utilizada por default
package			El package de donde se hacen los import de las clases
schema			El esquema de la base de datos en donde se aplican los queries

Tabla 2: Tag Hibernate Mapping

Atributos del elemento <class>

En este elemento se describe la relación entre el objeto Java y la base de datos.

Atributo	*Valores*	*Default*	*Descripción*
abstract	true, false	false	Este atributo solo debe tener el valor true si la clase es abstracta
batch-size		1	El numero de objetos que pueden ser obtenidos con el mismo id de la clase.
catalog			Ya descrito en Tabla 2
check			SQL para crear una sentencia multirow check para generar un esquema
discriminator-value	null, not-null		Valor usado para distinguir entre subclases idénticas de un tipo persistente común.
dynamic-insert	true, false	false	Si es true, columnas nulas aparecerán al ejecutar un comando INSERT cuando no ha colocado un valor por default.
dynamic-update	true, false	false	Si es true, columnas sin cambios aparecerán al ejecutar un comando UPDATE
entity-name			Nombre que se usara en lugar del nombre de clase.
lazy	true, false		Se utiliza para habilitar y deshabilitar el cargado perezoso.
mutable	true, false	true	Indica si la clase es mutable
name			El nombre completo de la clase que se hará persistir.
optimistic-lock	none, version, dirty, all	version	Especifica la estrategia de optimistic lock que sera usada.
persister			Permite que un objeto ClassPersister sea usado al hacer persistir esta clase.
polymorphism	implicit, explicit	implicit	Determina el tipo de polimorfismo que sera usado, el polimorfismo implícito permite regresar instancias de la clase cuando la superclase sea utilizada en el query
proxy			Especifica una clase o interface que sera utilizada como proxy para la inicialización perezosa.
rowid			Indica que identificador de columna debe ser usado
schema			Ya descrito en Tabla 2

Atributo	*Valores*	*Default*	*Descripción*
select-before-update	true, false	false	Si es true, Hibernate realiza un SELECT antes de un UPDATE para asegurarse que la actualización es necesaria.
table			El nombre de la tabla asociada con la clase.
where			Condición de la sentencia SQL WHERE que sera usada al recuperar objetos.

Tabla 3: Tag Class

Atributos del elemento <id>

Este elemento permite definir la llave primaria de la tabla, este es un elemento obligatorio

Atributo	*Valor*	*Default*	*Descripción*
access			Indica como se debe acceder a las propiedades, si es puesto en field, se accede directamente a los atributos de la clase, si es property se accede por los métodos de get y set. Se puede especificar una clase de tipo PropertyAccesor para definir otros tipos de acceso
column			El nombre de la columna que servirá de llave primaria.
length			El tamaño de la columna.
name			El nombre del atributo de la clase que representa la llave primaria
type			Algún tipo de dato soportado por Hibernate.
unsaved-value			El valor que el atributo debe tomar cuando una instancia de la clase ha sido creado. Es obligatorio.

Tabla 4: Tag Id

Atributos del elemento <generator>

Al establecer una llave primaria se debe especificar la forma en que esta se genera, Hibernate tiene implementadas varias formas de generar la llave primaria, pero no todas las formas son portables a todos los manejadores de bases de datos.

En la Tabla 5 se muestran los formas en que Hibernate genera llaves primarias y si son portables o no.

Nombre	***Descripción***
guid	Genera un identificador globalmente único, no es portable.
hilo	Genera identificadores únicos empleando una tabla y una columna de la base de datos, es portable.
identity	Soporta el tipo de columna identity proporcionado por algunos manejadores de bases de datos
increment	Genera llaves que empiezan en 1 y que se van incrementando en 1 cada vez que se agrega un nuevo registro a la base de datos, puede ser aplicado a los tipos int, short y long. Es portable
native	Selecciona entre sequence, identity e hilo como la forma de generar la llave primaria.
sequence	Soporta el tipo sequence proporcionado por algunos manejadores de bases de datos.
uuid	La llave se compone de la dirección IP local, la hora de JVM, lo hora del sistema y un valor continuo, no garantiza que la llave primaria sea única.

Tabla 5: Tag Generator

Atributos del elemento <property>

Esta propiedad permite especificar los atributos de la clase que no se relacionan con otras clases del modelo de dominio.

Atributo	*Valor*	*Default*	*Descripción*
access			Ya descrito en Tabla 4
column			El nombre de la columna que almacenara el valor del atributo, por default el nombre de la columna es el nombre del atributo.
formula			Un query SQL que representa una propiedad calculada dinamicamente.
index			El nombre de un index relacionado con esta columna.
insert	true, false	true	Especifica si en la creación de una instancia de la clase el atributo relacionado con esta propiedad debe ser almacenado.
lazy	true, false	false	Ya descrito en Tabla 3
length			Longitud de la columna.
name			Nombre del atributo en la clase que debe empezar con minúscula, este atributo es obligatorio.
not-null	true, false	false	Especifica si se puede almacenar un valor nulo.
optimistic-lock	true, false	true	Ya descrito en Tabla 3
precision			Permite especificar la presición del dato numérico.
scale			Permite especificar la escala del dato numérico.
type			Algún tipo de dato soportado por Hibernate.
unique	true, false	false	Indica si valores duplicados son permitidos.
update	true, false	true	Especifica si en la actualización del atributo de la clase esta columna también es modificada.

Tabla 6: Tag Property

Relaciones entre tablas

Como mencionamos antes las aplicaciones de la vida real no están descritas por tablas independientes,

sino por tablas que interactúen entre si, el problema ahora es como hacemos que Hibernate maneje estas relaciones entre las tablas.

Hasta donde nosotros sabemos la interacción entre tablas se hace colocando una llave foránea en la tabla que deseamos que interactúe con otra, esta llave foránea corresponde a la llave primaria de la otra tabla, sabemos también, que la interacción entre tablas debe tener una cardinalidad y que nosotros debemos manejar esta cardinalidad (en una relación n a n, nosotros debemos definir el indice que administra la relación).

En Hibernate la adminstración de las relaciones entre tablas es mas sencilla ya que tiene elementos que ademas de definir un atributo como llave foránea le asignan la cardinalidad de la relación, estos elementos son descritos en las siguientes secciones con los atributos que pueden tener.

Atributos del elemento <one-to-one>

Esta relación expresa que el atributo de la clase esta relacionado con un solo atributo de otra.

Atributo	*Valor*	*Default*	*Descripción*
access			Ya descrito en Tabla 4
batch-size			Ya descrito en Tabla 3
cascade			Determina como los cambios en la entidad padre afectan a los objetos relacionados.
catalog			Ya descrito en Tabla 2
check			Ya descrito en Tabla 3
fetch	join, select, subselect		El modo con el que el elemento sera recuperado
inverse	true, false	false	Especifica si la relación es bidireccional
lazy	true, false		Ya descrito en Tabla 3
name			Le asigna un nombre a la entidad (requerido en caso de que el mapa sea dinámico).
optimistic-lock	true, false	true	Ya descrito en Tabla 3
outer-join	true, false, auto		Especifica si un outer-join sera usado
persister			Especifica que ClassPersister sera usado para sobreescribir el que se encuentra definido por default
schema			Ya descrito en Tabla 2
table			El nombre de la tabla con la que esta asociado este atributo.
where			Ya descrito en Tabla 3

Tabla 7: Tag one-to-one

Atributos del elemento <one-to-many>

Especifica que un atributo de la clase esta relacionado con 0 o muchos elementos de otra clase

Atributo	*Valor*	*Default*	*Descripción*
access			Ya descrito en Tabla 4
cascade			Ya descrito en Tabla 7
class			El tipo del atributo, es decir, el nombre de la clase a la que se hace referencia.
column			El nombre que tendrá la columna que almacenara al atributo
entity-name			El nombre de la entidad asociada al atributo.
fetch	join, select		Ya descrito en Tabla 7
foreign-key			El nombre de la llave foránea generada por esta asociación. Por default Hibernate asigna un nombre
index			Indica el orden de la entidades cuando sean recuradas por la tabla inversa.
insert	true, false	true	Puesto en false, si al atributo ya tiene un valor asignado advierte cuando se intente cambiar.
lazy	true, false	false	Ya descrito en Tabla 3
name			El nombre del atributo en la clase.
not-found	exception, ignore	exception	El comportamiento que se toma cuando la entidad no existe
not-null	true, false	false	Especifica si se permiten valores nulos
optimistic-lock	true, false	true	Ya descrito en Tabla 3
outer-join	true, false, auto		Ya descrito en Tabla 7
property-ref			Si en la relación no se utiliza la llave primaria, este atributo se utiliza para indicar con que propiedad se tiene la relación.
unique	true, false	false	Especifica si se permiten identificadores duplicados

Atributo	*Valor*	*Default*	*Descripción*
update	true, false	true	Ya descrito en Tabla 6

Tabla 8: Tag one-to-many

Atributos del elemento <many-to-many>

Especifica que un atributo de la clase tiene una relación de muchos a muchos con un atributo de otra clase.

Atributo	*Valor*	*Default*	*Descripción*
column			El nombre que tendrá la columna que almacenara al atributo
class			El tipo del atributo, es decir, el nombre de la clase a la que se hace referencia.
outer-join	true, false, auto		Ya descrito en Tabla 7

Tabla 9: Tag many-to-many

Es común que los atributos que están relacionados con otros atributos de otras clases estén almacenados en colecciones de datos, por ejemplo:

```
public class Prestamo{
	private Set<Usuario> listaDeudores = new HashSet<Usuario>();
	// métodos de get y set
}
```

Hibernate proporciona soporte para poder hacer persistir a estos elementos y hacer mas fácil su manipulación.

En la siguiente sección mostraremos las colecciones que soporta Hibernate, como se describen en el archivo de mapeo y como podemos trabajar con ellas.

Atributos comunes de los elementos de colección

Los objetos de colección tienen varios atributos en común, para no estar repitiendo la información la Tabla 10 muestra la información común a los objetos de colección soportados por Hibernate.

Atributo	*Valor*	*Default*	*Descripción*
access			Ya descrito en Tabla 4
batch-size			Ya descrito en Tabla 3
cascade			Ya descrito en Tabla 7
catalog			Ya descrito en Tabla 2
check			Ya descrito en Tabla 3
fetch	join, select, subselect		Ya descrito en Tabla 7
lazy	true, false		Ya descrito en Tabla 3
name			El nombre que tiene el atributo en la clase, debe empezar con minúscula
optimistic-lock	true, false	true	Ya descrito en Tabla 3
outer-join	true, false, auto		Ya descrito en Tabla 7
persister			Ya descrito en Tabla 7
schema			Ya descrito en Tabla 2
table			El nombre de la tabla asociada con este atributo.
where			Ya descrito en Tabla 3

Tabla 10: Atributos de los elementos de colección

Atributos del elemento <set>

Este objeto es el equivalente a la interfaz java.util.Set, como en este objeto no se permiten objetos idénticos, por lo que se recomienda sobrecargar los métodos de `equals` y `hashCode`, para determinar que es un objeto idéntico que es un objeto igual.

Atributo	*Valor*	*Default*	*Descripción*
inverse	true, false	false	Ya descrito en Tabla 7
order-by			Especifica un criterio SQL para ordenar a los atributos cuando se recuperan de la base de datos.
sort			Especifica la clase Collection que sera usada, puede tomar los valores de unsorter, natural o cualquier otra clase Comparator.

Tabla 11: Tag Set

Ejemplo del uso de un elemento Set en una relación unidireccional de muchos a muchos sin objetos replicados:

Supongamos que estamos mapeando la siguiente relación:

Ilustración 14: Ejemplo de uso de Set

Nota: Cabe mencionar que esto es solo un ejemplo y que por comodidad se obligo a que sea unidireccional.

En la clase `Usuario` existe el atributo:

```
private Set<Libro> librosPrestados = new HashSet<Libro>();
```

En el archivo Usuario.hbm.xml debe agregarse el siguiente tag:

```
<set name="librosPrestados" table="Prestamo" lazy="false">
      <key column="idUsuario"/>
      <many-to-many class="mx.uam.ayd.biblioteca.Libro" column="idLibro"/>
</set>
```

En el tag Set se especifica el nombre del atributo en la clase (name), la tabla que se asocia como indice de la relación (table), ya que en las relaciones de muchos a muchos es necesario contar con uno, y si los objetos están en memoria o no (lazy).

En el tag Key se especifica la columna que se usa como llave primaria en la tabla.

En el tag many-to-many se espacifica con que clase se relaciona y que columna usa como llave primaria.

Atributos del elemento <list>

Es el equivalente a la interfaz java.util.List, este objeto permite objetos replicados.

Atributo	*Valor*	*Default*	*Descripción*
inverse	true, false	false	Ya descrito en Tabla 7

Tabla 12: Tag List

Ejemplo del uso del elemento List en una relación unidireccional de muchos a muchos permitiendo objetos replicados en la asociación:

Tomando el mismo ejemplo ilustrado en Ilustración 14, vamos a suponer que ahora existe un atributo de tipo List en la clase `Usuario`:

```
private List<Libro> librosPrestados = new ArrayList<Libro>();
```

En el archivo Usuario.hbm.xml debe agregarse el siguiente tag:

```
<list name=”librosPrestados” table=”Prestamo” lazy=”false”>
      <key column=”idUsuario”/>
      <many-to-many class=”mx.uam.ayd.biblioteca.Libro” column=”idLibro”/>
</list>
```

Al parecer no existe diferencia entre Set y List, la diferencia consiste en el contexto de la aplicación y en que Hibernate no soporta que en relaciones bidireccionales en el lado de la relación que tiene cardinalidad muchos que se utilicen listas, mapas o arreglos.[2]

Atributos del elemento <idbag>

Atributo	*Valor*	*Default*	*Descripción*
order-by			Ya descrito en Tabla 11

Tabla 13: Tag IdBag

Atributos del elemento <map>

Es el equivalente a la interfaz java.util.Map.

2 Para mas información consultar la pagina http://www.hibernate.org/hib_docs/reference/en/html/collections.html

Atributo	*Valor*	*Default*	*Descripción*
inverse	true, false	false	Ya descrito en Tabla 7
order-by			Ya descrito en Tabla 11
sort			Ya descrito en Tabla 11

Tabla 14: Tag Map

Atributos del elemento <bag>

Atributo	*Valor*	*Default*	*Descripción*
inverse	true, false	false	Ya descrito en Tabla 7
order-by			Ya descrito en Tabla 11

Tabla 15: Tag Bag

Relaciones bidireccionales

Así como es común que interactúen 2 tablas, también lo es que se pueda navegar en ambos sentidos de la relación, en esta sección vamos a tratar este problema para mostrar como se mapean estas visibilidades empleando Hibernate.

Hay que mencionar que Hibernate soporta 2 tipos de asociaciones bidireccionales

- one-to-many
- many-to-many

por lo que en las 2 siguientes secciones vamos a tratar la forma en que se manejan estas relaciones por medio de un ejemplo y vamos a omitir la descripción de los atributos de cada relación.

Relaciones del tipo <one-to-many>

Supongamos el siguiente ejemplo:

Ilustración 15: Ejemplo de relación bidireccional one-to-many

En esta relación un objeto Padre puede tener de 0 a muchos Hijos y los Hijos están interesados en conocer a su respectivo Padre.

Archivo Padre.hbm.xml:

```
<hibernate-mapping>
      <class name=”package.Padre” table=”Padre”>
            <id name=”idPadre” type=”int”>
                  <generator class=”increment”/>
            </id>
            <!-- Otros atributos del Padre -->
            <set name=”hijos” inverse=”true” lazy=”false”>
                  <key column=”idPadre”/>
                  <one-to-many class=”package.Hijo”/>
            </set>
      </class>
</hibernate-mapping>
```

Archivo Hijo.hbm.xml:

```
<hibernate-mapping>
      <class name=”package.Hijo” table=”Hijo”>
            <id name=”idHijo” type=”int”>
                  <generator class=”increment”/>
            </id>
            <!-- Otros atributos del Hijo -->
            <many-to-one name=”padre” class=”package.Padre” column=”idPadre”/>
      </class>
</hibernate-mapping>
```

Las cosas que cabe hacer notar es que el padre tiene un atributo:

```
private Set<Hijo> hijos = new HashSet<Hijo>();
```

que el Hijo tiene un atributo:

```
private Padre padre;
```

y que en el caso del Padre la relación va como one-to-many con inverse = true y que esta relación se invierte en el Hijo a many-to-one (el hecho de que la relación se invierta de esta forma no tiene nda que ver con que inverse sea igual a true).

Relaciones del tipo <many-to-many>

Supongamos ahora la siguiente relación:

Ilustración 16: Ejemplo relación bidireccional muchos a muchos

En esta relación un objeto Padre tiene de 0 a muchos Hijos, y los Hijos tienen al menos un Padre (idealmente solo 2).

Archivo Padre.hbm.xml:

```
<hibernate-mapping>
      <class name="package.Padre">
            <id name=”idPadre” type=”int”>
                  <generator class=”increment”/>
            </id>
            <!-- Otros atributos del Padre -->

            <set name="hijos" table="Hijos" inverse="true" lazy="false">
                  <key column="idPadre"/>
                  <many-to-many class="package.Hijo" column="idHijo"/>
            </set>
      </class>

</hibernate-mapping>
```

Archivo Hijo.hbm.xml:

```
<hibernate-mapping>
      <class name="package.Hijo">
            <id name=”idHijo” type=”int”>
                  <generator class=”increment”/>
            </id>
            <!-- Otros atributos del Hijo -->

            <set name="padres" table="Hijos" lazy="false">
                  <key column="idHijo"/>
                  <many-to-many class="package.Padre" column="idPadre"/>
            </set>
      </class>

</hibernate-mapping>
```

Las cosas que cabe hacer notar es que ahora el Hijo tiene un atributo:

```
private Set<Padre> padres = new HashSet<Padre>();
```

y que en ambos casos los identificadores de cada clase se encuentran en una misma tabla (Hijos) que Hibernate utiliza y administra como indice.

Mapeo de clases y sus relaciones

Hasta el momento hemos realizado mapeos de objetos a tablas relacionales empleando relaciones de asociación entre los objetos, pero en algunas ocasiones quisiéramos mapear otros tipos de relaciones entre objetos. Dos de la asociaciones mas empleadas en orientación a objetos son la composición y la herencia, Hibernate permite realizar el mapeo de estas relaciones de forma muy sencilla.

En las siguientes 2 secciones emplearemos ejemplos para ilustrar la forma en que se puede realizar el mapeo de relaciones de composición y herencia empleando Hibernate.

Mapeo de la relación de composición

En la Ilustración 17 se muestra una relación en la que una Persona esta compuesta de un Nombre

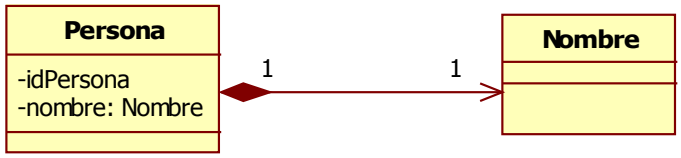

Ilustración 17: Ejemplo de relación de composición

El archivo Persona.hbm.xml:

```
<hibernate-mapping>
      <class name=”package.Persona" table="Persona">
            <id name="idPersona" type="string">
                  <generator class="assigned"/>
            </id>
            <!-- Otros atributos de Persona -->
            <component name="nombre" class="package.Name">
                  <!-- Propiedades de la clase Nombre -->
            </component>
      </class>

</hibernate-mapping>
```

En esta relación el objeto de tipo Nombre es guardado como un valor no como una entidad, por lo que en la clase no es necesario definir un identificador, pero en la clase Persona se deben definir los getters y setters para el objeto de tipo Nombre.

El tag component puede tener los atributos de cualquier otro elemento y ademas define un subelemento llamado <parent> que permite definir una asociación con el objeto que lo contiene. Por ejemplo:

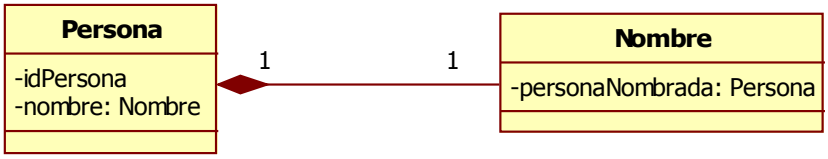

Ilustración 18: Ejemplo de relación de composición bidireccional

en este caso el archivo de Persona.hbm.xml queda:

```
<hibernate-mapping>
      <class name=”package.Persona" table="Persona">
            <id name="idPersona" type="string">
                  <generator class="assigned"/>
            </id>
            <!-- Otros atributos de Persona -->
            <component name="nombre" class="package.Name">
                  <parent name="personaNombrada"/>
                  <!-- Propiedades de la clase Nombre -->
            </component>
      </class>

</hibernate-mapping>
```

Con esto permitimos que desde el objeto de tipo Nombre se pueda tener visibilidad con un objeto de tipo Persona.

Mapeo de la relación de herencia

La Ilustración 19 muestra una relación de herencia entre 3 clases:

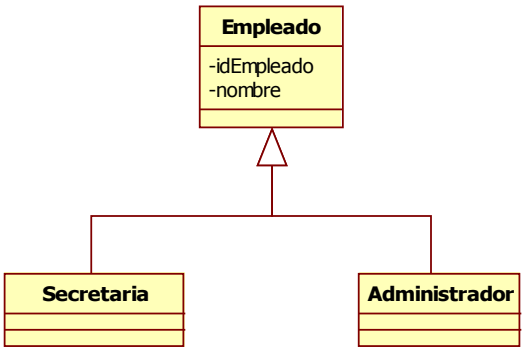

Ilustración 19: Ejemplo de relación de Herencia

Hibernate soporta 3 formas de mapear este tipo de asociaciones que describiremos a continuación.

1. Una tabla por clase concreta

Se mapea cada clase de la relación de herencia como una clase concreta normal, en cada clase se mapean todos los atributos incluyendo los que son heredados. En esta filosofía de mapeado no se mapean interfaces ni clases abstractas.

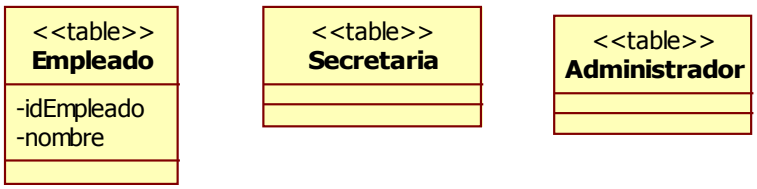

Ilustración 20: Esquema de la base de datos

Desventajas:

- En las subclases se tienen datos replicados, es decir, que por cada subclase de la relación se deben repetir los datos que son heredados.
- Los cambios en la superclase afectan a muchas tablas.

Ejemplo de como se mapearía la clase Secretaria con esta estrategia

Archivo Secretaria.hbm.xml:

```
<hibernate-mapping>
      <class name="package.Secretaria" table="Secretaria">
            <!-- El identificador y el nombre son los atributos que
                  hereda de Empleado -->
            <id name="id" type="int">
                  <generator class="increment">
            </id>
            <property name="nombre" tupe="string"/>
            <!-- Otros atributos de Secretaria -->
      </class>
</hibernate-mapping>
```

2. Una tabla por subclase

Se mapea cada clase en la relación incluyendo clases abstractas e interfaces. La relación *"es un"* de la herencia se convierte en una relación *"tiene un"* en el esquema de la base de datos.

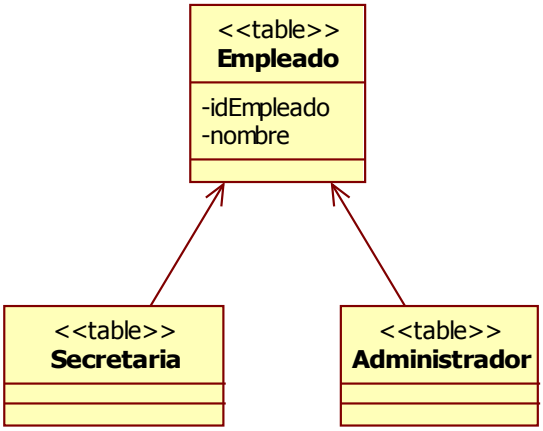

Ilustración 21: Conversión de la relación a "tiene un"

Ventajas:

- Es conceptualmente mas fácil de comprender
- No requiere de cambios complejos en la base de datos cuando se modifica a la superclase

Desventajas:

- Puede resultar en un pobre desempeño de la aplicación si la arquitectura del sistema esta mal planteada.

Ejemplo de mapeo de la clase Administrador empleando esta estrategia

Archivo Administrador.hbm.xml:

```
<hibernate-mapping>
      <joined-subclass name=”package.Administrador” extends=”package.Empleado”>
            <!-- Identificador de la clase Empleado -->
            <key column=”idEmpleado”>

            <!-- Lista de los atributos de la clase Administrador -->
      </joined-subclass>
</hibernate-mapping>
```

En esta ocasión sustituimos el tag class por el joined-subclass para establecer una relación directa con la superclase, en este tag especificamos cual es la clase que extiende y con el tag key hacemos referencia a los elementos de la superclase que no han sido agregados en este archivo, es decir, no es necesario redefinir los elementos de idEmpleado y nombre.

El tag joined-subclass no puede contener tags de subclass y viceversa.

3. Una tabla por jerarquía de la clase.

Esta estrategia ponemos en una sola tabla a todos los atributos que existan en la relación de herencia, es decir, realizamos una lista con todos los atributos de todas la clases de la relación de herencia y los mapeamos todos en una sola tabla.

A esta tabla le agregamos un elemento discriminador para distinguir el tipo base representado por cada fila en la tabla.

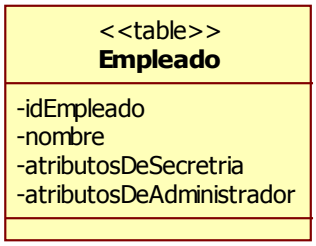

Ilustración 22: Esquema de la base de datos con la estrategia de una tabla por jerarquía

Ventajas:

- Ofrece un mejor desempeño ya que los queries se realizan en una misma tabla y se puede recuperar el objeto mas fácilmente.
- Los cambios en los atributo solo requieren del cambio de una columna.

Desventajas:

- No es una buena representación de los atributos en las clases.
- Si el numero de subclases crece también lo hace el numero de columnas.
- Cada subclase debe definir la clase que extiende y su elemento discriminador.

Ejemplo de como se realiza el mapeo con esta estrategia

Archivo Empleado.hbm.xml:

```
<hibernate-mapping>
      <class name="package.Empleado" table="Empleado">
            <!-- Identificador de la superclase -->
            <id name="idEmpleado" type="int">
                  <generator class="increment"/>
            </id>
            <!-- Columna que sirve como discriminador del tipo de subclase -->
            <discriminator column="tipoEmpleado" type="string"/>

            <!-- Lista de los atributos de la clase Empleado -->
```

```
            <!-- Definicion de la subclase Secretaria -->
            <subclass name="package.Secretaria" discriminator-value="SECRE">
                  <!-- Lista de los atributos de la clase Secretaia -->
            </subclass>

            <!-- Definicion de la subclase Administrador -->
            <subclass name="package.Administrador" discriminator-value="ADMIN">
                  <!-- Lista de los atributos de la clase Administrador -->
            </subclass>

      </class>
</hibernate-mapping>
```

Referencias

En ingles

- Pro Hibernate 3. Jeff Linwood y Dave Minter. (Este libro se encuentra en books24x7).
- http://www.hibernate.org/hib_docs/reference/en/html/index.html. (Documentación de Hibernate, esta disponible en otros idiomas menos en español).
- http://anonhibernate.labs.jboss.com/trunk/sandbox/jdbc/src/test/resources/hibernate.properties. (Da una lista de propiedades para configurar Hibernate con cualquiera de los Manejadores de Bases de Datos que conoce).
- http://www.hibernate.org/116.html#A8. (FAQ del sitio de Hibernate)

En español

- http://www.javahispano.org/articles.article.action?id=93. (Articulo de Francsec Roses Albiol sobre Hibernate).
- http://www.programacion.com/java/tutorial/hibernate/. (Tutorial para aprender a manejar Hibernate).
- http://www.programacion.com/java/articulo/jap_persis_hib/. (Versión en español del articulo publicado por Jeff Hanson, trata de la persistencia de objetos Java con Hibernate).
- http://www.javahispano.org/articles.article.action?id=95. (Articulo de Martín Pérez Mariñán, que trata sobre manejo de caches y concurrencia con Hibernate).
- http://www.epidataconsulting.com/tikiwiki/tiki-pagehistory.php?page=Hibernate&diff2=21&diff_style=sideview&PHPSESSID=8abd9818af1607b0f7df3a86764d39b7. (Trata sobre problemas de mapeo recursivo de objetos).