Introducción

Este es un laboratorio de introducción al framework Hibernate, que es una herramienta de software libre para realizar mapeo objeto-relacional. En el mismo, se crearán en Java algunas clases del dominio a las que se les dará persistencia en una base de datos MySQL. Así mismo se mostrará cómo se pueden realizar consultas utilizando el lenguaje HQL. Hibernate permite implementar el acceso a datos que forma parte de una capa de lógica del negocio, de tal manera que la aplicación trabaje con objetos del dominio, e Hibernate se encargue de recuperarlos, modificarlos o borrarlos de la base de datos, de manera automática y transparente (a excepción de la configuración, que no podrá ser transparente).

En el laboratorio de JSFs se ha construido una aplicación Web; en particular la parte de capa de presentación de un caso de uso que permite a los usuarios hacer login, y la parte que invoca a un bean que puede implementar o a su vez invocar a la capa de lógica del negocio. Una vez realizado este laboratorio, se podrá implementar dicha lógica del negocio utilizando Hibernate para acceder a una base de datos MySQL donde se encuentren los usuarios. Esto se propondrá como un ejercicio posterior; no se hará en este laboratorio, pero en este laboratorio se trabajará a partir del proyecto creado en el laboratorio previo de JSFs.

Objetivos

Los objetivos de este laboratorio son los siguientes:

- Desarrollar una aplicación Hibernate utilizando la herramienta Eclipse.
- Usar las utilidades de Hibernate para generar automáticamente los enlaces (mapping) a partir de clases Java (POJOs) a tablas de BD relacionales MySQL.
- Conocer los elementos básicos en toda aplicación Hibernate: clases del dominio (Java Beans o POJOs), ficheros de enlace o mapping, fichero de configuración de Hibernate (hibernate.cfg.xml) y clases de acceso a los datos (DAO) que operan con la base de datos con operaciones objeto a objeto, o que realizan preguntas sobre varios objetos usando un lenguaje como HQL (Hibernate Query Language)

Tareas a realizar

1. Crear un proyecto Java Hibernate que trabaje con una única clase

Para ello hay que realizar los pasos siguientes, habituales en la creación de un proyecto Hibernate.

1.1.- Crear/Abrir un proyecto Java

Para trabajar con Hibernate es suficiente con crear un proyecto Java normal (File => New => Project => Java => Java Project). Sin embargo, en vez de crear un proyecto nuevo, se propone trabajar con el proyecto resultado del laboratorio JSF, que se utilizará en los ejercicios de Hibernate propuestos en el curso.

Abrir el proyecto LoginJSF. Si el proyecto no se encuentra abierto en el workspace, entonces hay que importarlo:

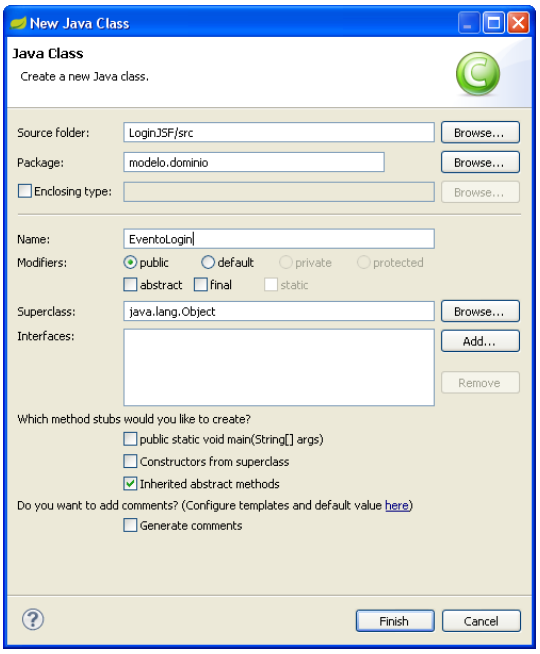
File => Import => General => Existing Projects into Workspace => Browse => Seleccionar directorio workspace y dentro de él el directorio LoginJSF

Nota: si se desea mantener el proyecto anterior tal y como estaba y trabajar con uno nuevo a partir del anterior, se puede realizar una copia del directorio LoginJSF que se encuentra en el directorio donde se almacenan los proyectos del workspace. Al directorio con la copia hay que darle el nuevo nombre del proyecto (por ejemplo LoginJSF-Hibernate). Y además hay que dar ese mismo nombre de proyecto en el fichero .project , que es un fichero XML con la etiqueta <name> que indica el nombre del proyecto. En este caso: <name>LoginJSF-Hibernate</name>.

1.2.- Crear las clases del dominio

Crear la siguiente clase Java (un JavaBean o clase POJO) que representa una clase del dominio llamada EventoLogin, a cuyos objetos se les dará persistencia, esto es, se almacenarán en una base de datos relacional.

New => Class => Poner el nombre del paquete (modelo.dominio) y de la clase (EventoLogin)



Añadir el siguiente código a la clase:

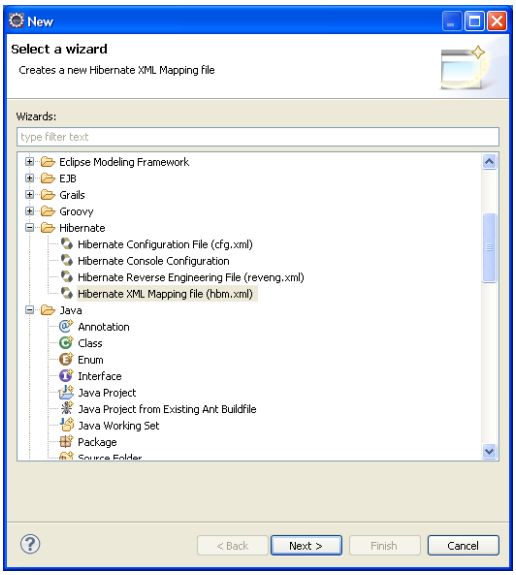
NOTA: debe ser un Bean (o POJO), y no hay que olvidar añadir un constructor vacío, para que posteriormente no se generen errores cuando Hibernate necesite crear objetos de esta clase y no disponga de ese constructor. Además, encontrar ese tipo de errores en la traza de mensajes del entorno es muchas veces complicado.

```java
package modelo.dominio;

import java.util.Date;

public class EventoLogin {
    private Long id;
    private String descripcion;
    private Date fecha;

    public EventoLogin() {}

        public Long getId() {
                return id;
        }
        public void setId(Long id) {
                this.id = id;
        }
        public String getDescripcion() {
                return descripcion;
        }
        public void setDescripcion(String descripcion) {
                this.descripcion = descripcion;
        }
        public Date getFecha() {
                return fecha;
        }
        public void setFecha(Date fecha) {
                this.fecha = fecha;
        }
}
```

1.3.- Crear las ficheros de enlace o mapeo.

Crear el fichero de mapeo, el cual indica a Hibernate la tabla de la base de datos donde se almacenarán los objetos de la clase EventoLogin.

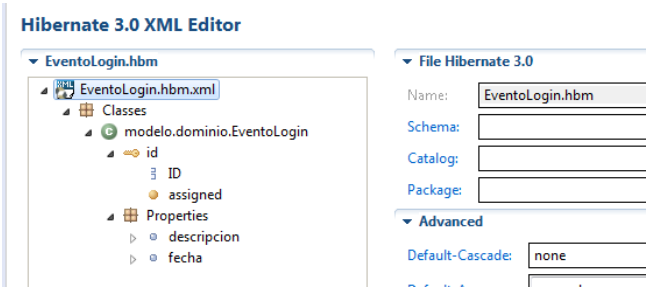
New => Other => Hibernate => Hibernate XML Mapping File



Next => Add Class => Seleccionar “EventoLogin” tras escribir “EventoLogin” (o el comienzo del mismo) en el campo de texto “Select types to be mapped”

Tras pulsar “Finish” se puede comprobar que se ha generado el fichero de mapping EventoLogin.hbm.xml, el cual indica a Hibernate en qué tabla (EVENTOLOGIN) y

columnas (ID, DESCRIPCION, FECHA) se guardarán los objetos de la clase EventoLogin. A destacar que el atributo "id" de tipo "Long" (que se usa como identificador de los objetos de la clase) se mapeará con el atributo "ID" y va a definir como la clave primaria de la tabla EVENTOLOGIN. Además, cada vez que se genere un nuevo objeto de Event habrá que asignar explícitamente un valor para el atributo "id" (esto es lo que se indica con `<generator class="assigned" />` en el fichero XML de mapping)



Fichero de mapping: EventoLogin.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 05-dic-2016 17:08:40 by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
    <class name="modelo.dominio.EventoLogin" table="EVENTOLOGIN">
        <id name="id" type="java.lang.Long">
            <column name="ID" />
            <generator class="assigned" />
        </id>
        <property name="descripcion" type="java.lang.String">
            <column name="DESCRIPCION" />
        </property>
        <property name="fecha" type="java.util.Date">
            <column name="FECHA" />
        </property>
    </class></hibernate-mapping>
```

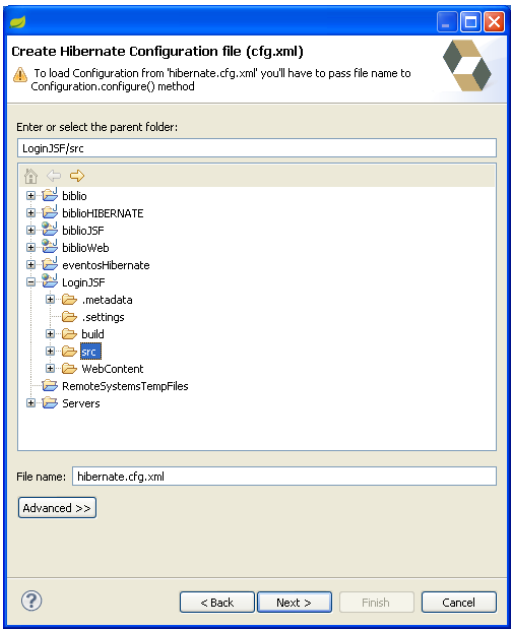
Los valores, tipos, etc. del fichero de configuración pueden ser modificados en el fichero XML generado directamente o bien utilizando el Hibernate 3.0 XML Editor, accesibles en las pestañas "Source" y "Tree" respectivamente. Seleccionar el fichero EventoLogin.hbm.xml y hacer click derecho => Open with => Hibernate 3.0 XML Editor si no es visible.

1.4.- Crear el fichero de configuración de Hibernate (hibernate.cfg.xml).

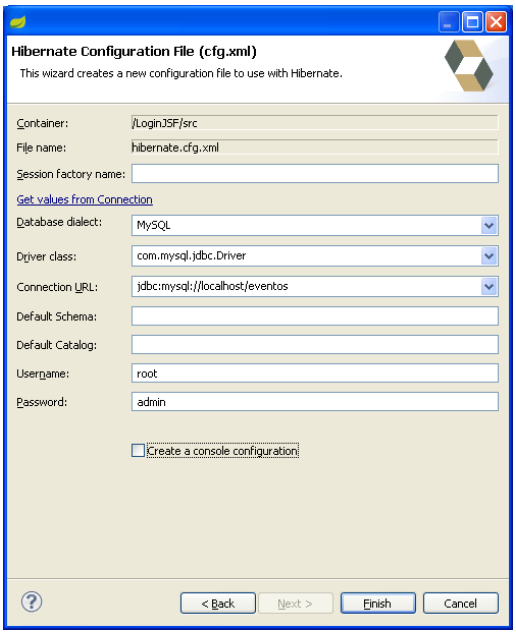
En este fichero se configura un "Session Factory", que es una factoría para crear sesiones de trabajo con una base de datos concreta (en este caso la BD eventos) de un SGBD concreto (que será MySQL).

New => Hibernate => Create Hibernate Configuration File =>

IMPORTANTE: Seleccionar el proyecto "LoginJSF" y ***el directorio "src" que es donde se debe dejar por defecto*** el fichero hibernate.cfg.xml.



Continuar con “Next” e introducir las siguientes opciones: el driver de MySQL, el nombre de la conexión (`jdbc:mysql://localhost/eventos`) que permite trabajar con una BD MySQL local que se va a llamar “eventos” y el nombre de usuario “root” y su password “admin” en la BD, si es que este nombre de usuario y password son los que se hayan definido) y aceptarlas con “Finish”.

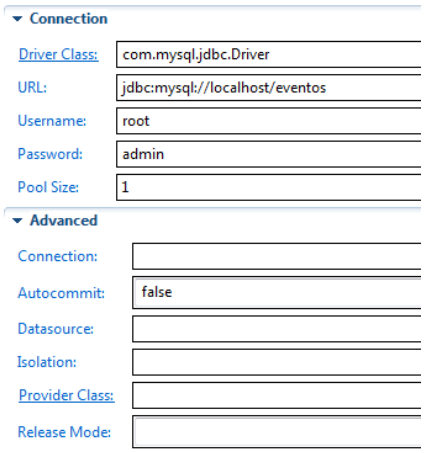


El resto de propiedades las asignaremos utilizando el editor gráfico. Para ello:

Hacer click derecho sobre el fichero hibernate.cfg.xml => Open with => Hibernate Configuration 3.0 XML Editor

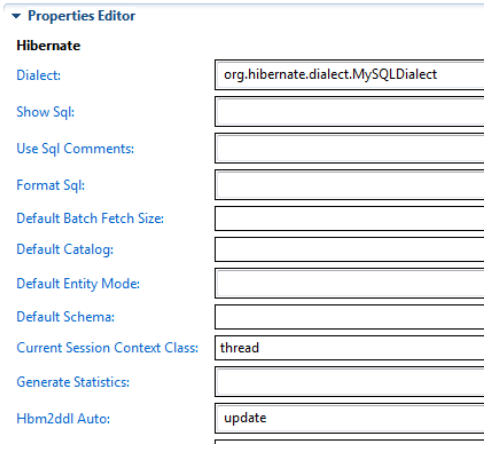
En Session Factory => Properties => Connection => Añadir 1 en “Pool Size” y “false” en Autocommit

Nota: Estas dos opciones establecen un comportamiento adecuado para este caso: que el sistema manejará un "pool" de conexiones de 1, por lo que estará preparado para trabajar con 1 conexión (lo cual es suficiente si no estimamos que haya varias peticiones de conexiones a la vez) y que cada sentencia SQL (de inserción, modificación o borrado) no terminará automáticamente con un commit, por lo que será necesario hacer commit desde la aplicación.

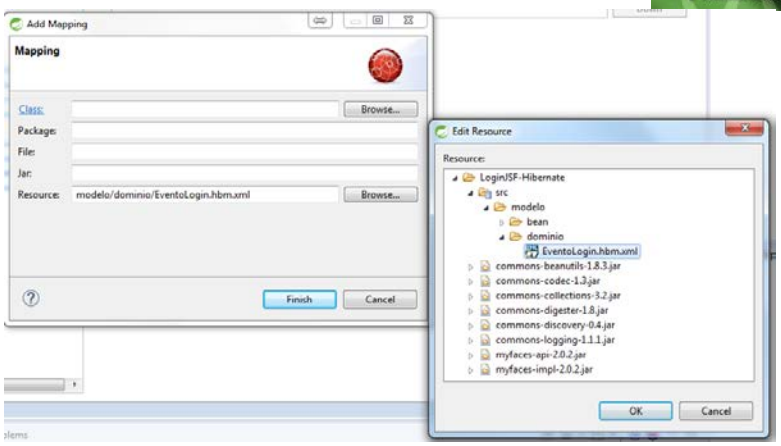


En Session Factory => Properties => Hibernate => Añadir "thread" en "Current Session Context Class" y "update" en Hbm2ddlAuto

Nota: Estas dos opciones permiten un mantenimiento automático de sesiones y la generación automática de los esquemas en la base de datos (con la opción "update" borrará y creará las tablas en la puesta en marcha de la aplicación Hibernate, y después las irá actualizando en las distintas sesiones de trabajo)



En Session Factory => Mappings => Add => Browse de Resource => Seleccionar el mapping anteriormente generado para la clase EventoLogin.hbm.xml



El fichero hibernate.cfg.xml así generado tendrá el contenido siguiente (que se puede ver en la pestaña Source, tras seleccionar el fichero hibernate.cfg.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
                "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory name="">
  <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="hibernate.connection.password">admin</property>
  <property name="hibernate.connection.url">jdbc:mysql://localhost/eventos</property>
  <property name="hibernate.connection.username">root</property>
  <property name="hibernate.connection.autocommit">false</property>
  <property name="hibernate.connection.pool_size">1</property>
  <property name="hibernate.hbm2ddl.auto">update</property>
  <property name="hibernate.current_session_context_class">thread</property>
  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  <mapping resource="modelo/dominio/EventoLogin.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

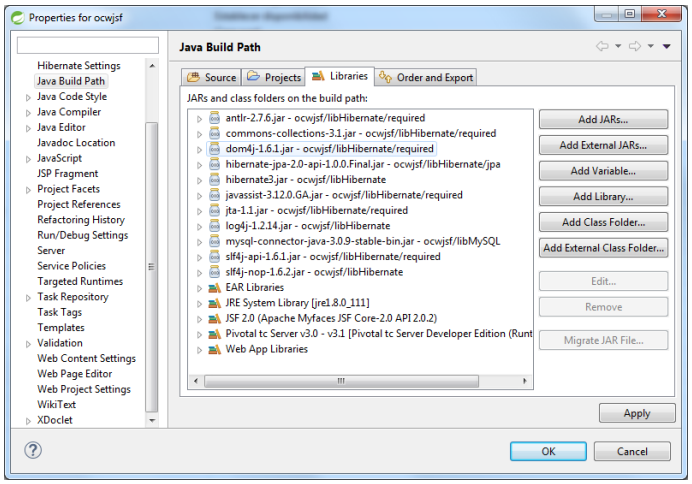
1.5.- Crear el fichero HibernateUtil

Crear la clase modelo.HibernateUtil que obtendrá la instancia de “Session Factory” y la guardará en atributo estático para que pueda ser utilizada más adelante por medio de `HibernateUtil.getSessionFactory()`

```
package modelo;

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

public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();
    private static SessionFactory buildSessionFactory() {
        try {
            return new Configuration().configure().buildSessionFactory();
// Nota: forma de obtener el Session Factory en Hibernate 3
// Crea una instancia de SessionFactory con los datos de configuración
// de hibernate.cfg.xml, que debe estar situado en el subdirectorio “src”
// Si se creara la instancia de SessionFactory de la siguiente manera:
// new Configuration().configure(new File("hibernate.cfg.xml")).buildSessionFactory();
// entonces, habría que colocar el fichero en el directorio raíz del proyecto
// Y podría tener un nombre diferente a "hibernate.cfg.xml"
        } catch (Throwable ex) {
            System.err.println("Fallo creando el SessionFactory." + ex);
            throw new ExceptionInInitializerError(ex);}}
    public static SessionFactory getSessionFactory() {
        return sessionFactory;} }
```

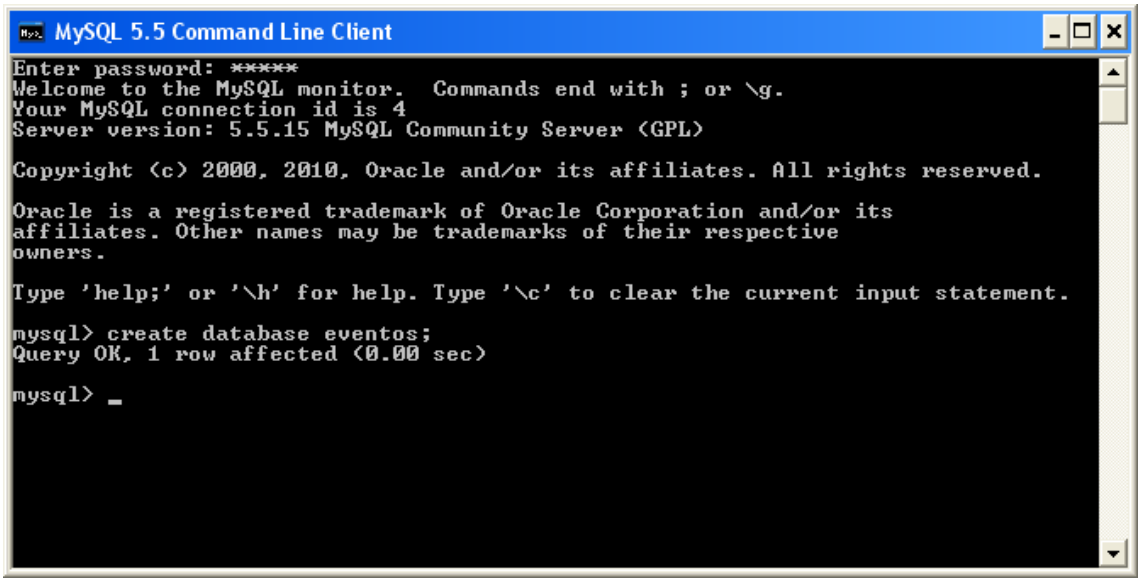
En este punto se necesita añadir las librerías de Hibernate y MySQL al proyecto (que se encuentran en el este enlace del curso: Librerías utilizadas en la implementación).

Se pueden descomprimir las librerías en el directorio donde se encuentra el proyecto LoginJSF en el workspace. Se refresca el proyecto (F5 en el explorador de proyecto), entonces se pueden añadir las librerías Project => Properties => Java Build Path => Add JARs (y añadir todos los JAR de libHibernate (incluyendo las de los subdirectorios required y jpa) y el JAR de libMySQL)



Hay que crear la BD “eventos” con `create database eventos;` Para ello, hay que abrir una consola de MySQL (entrando con la cuenta “root” y password “admin”).

NOTA: Nótese que no creamos la tabla “EVENTOLOGIN”, sólo la BD.



1.6.- Crear las clases de acceso a los datos (clases DAO)

Crear la clase siguiente (principal.CrearEventos) la cual genera unos eventos y los lista a continuación por medio de transacciones realizadas utilizando Hibernate. Para ello, se obtiene el objeto "Session Factory" (invocando a la clase HibernateUtil), y por cada transacción, se le pide a dicho objeto que obtenga la sesión actual para operar con la base de datos, que comience una transacción, que inserte datos o realice preguntas y que termine la transacción. Esas transacciones siguen este esquema:

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();

// OPERACIONES con la BD (que componen la transacción)

session.getTransaction().commit();
```

```
package principal;

import modelo.HibernateUtil;
import modelo.dominio.EventoLogin;
import org.hibernate.Session;
import java.util.*;


public class CrearEventos {

        public CrearEventos(){
        }

    private void createAndStoreEventoLogin(Long id, String descripcion, Date fecha) {
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();

        EventoLogin e = new EventoLogin();
        e.setId(id);
        e.setDescripcion(descripcion);
        e.setFecha(fecha);
        session.save(e);
        session.getTransaction().commit();
    }

    private List listaEventos() {
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        List result = session.createQuery("from EventoLogin").list();
        session.getTransaction().commit();
        return result;
    }
    public static void main(String[] args) {
        CrearEventos e = new CrearEventos();
        System.out.println("Creación de eventos:");
        e.createAndStoreEventoLogin(1L,"Pepe ha hecho login correctamente", new Date());
        e.createAndStoreEventoLogin(2L,"Nerea ha intentado hacer login", new Date());
        e.createAndStoreEventoLogin(3L,"Kepa ha hecho login correctamente", new Date());
        System.out.println("Listado de eventos:");

        List eventos = e.listaEventos();
        for (int i = 0; i < eventos.size(); i++) {
            EventoLogin ev = (EventoLogin) eventos.get(i);
            System.out.println("Id: " + ev.getId() + " Descripcion: "
                               + ev.getDescripcion() + " Fecha: " + ev.getFecha());
        }
    }
}
```

Con respecto a las operaciones con la BD indicadas en el esquema de transacción anterior

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();

// OPERACIONES con la BD (que componen la transacción)

session.getTransaction().commit();
```

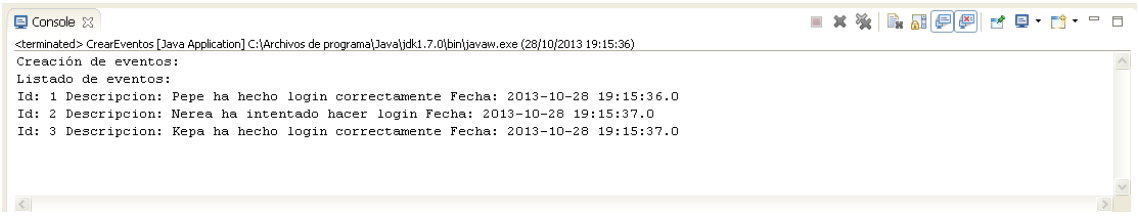
se pueden encontrar de dos tipos:

- objeto a objeto: actualización, inserción, borrado o recuperación de un objeto
- sobre varios objetos: consultar varios objetos mediante una pregunta

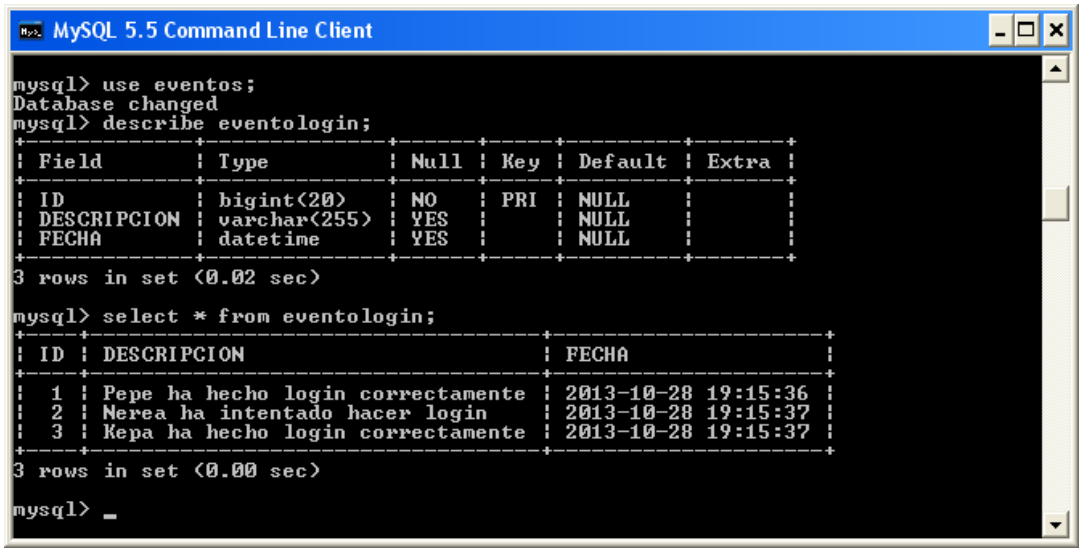
Como ejemplo del primer tipo, en la clase anterior, la instrucción `session.save(e)` inserta en la BD el objeto `e`, Como ejemplo del segundo tipo aparece una pregunta a la BD expresada en el lenguaje HQL (Hibernate Query Language). Es la pregunta `from EventoLogin` que se ejecuta en la instrucción `session.createQuery("from EventoLogin")`

1.7.- Ejecutar la aplicación para comprobar su correcto funcionamiento

Ejecutar la clase CrearEventos [Click derecho sobre ella en el explorador de paquetes => Run As => Java Aplication] y comprobar que se han creado y a continuación listado los eventos insertados.



En un intérprete de comandos de la BD MySQL se puede comprobar que se ha generado automáticamente la tabla EVENTOLOGIN en la base de datos “eventos” con los nombres y tipos siguientes (y con ID como clave primaria), así como el contenido de dicha tabla, mediante las correspondientes sentencias SQL.



Ejecutar de nuevo la clase CrearEventos y comprobar que se produce el siguiente error: `Duplicate entry '1' for key 'PRIMARY'`.

Este error se produce porque la llamada:

```
e.createAndStoreEventoLogin(1L,"Pepe ha hecho login correctamente", new Date());
```

crea un nuevo EventoLogin `e` al que se le asigna `1L` como identificador, y después se ejecuta un `session.save(e)`. En ese momento, Hibernate intenta insertar una tupla de EventoLogin con el valor 1 como ID en la tabla eventologin de la BD MySQL `eventos`. Como ya existe una tupla con ese valor en la clave primaria, entonces se lanza la excepción: `Duplicate entry '1' for key 'PRIMARY'`

1.8.- Probar algunas opciones de configuración de Hibernate

1.8.1.- Método de asignación de valores a los atributos que son claves primarias.

Una posible solución al problema de claves primarias duplicadas (`Duplicate entry for key 'PRIMARY'`) consistiría en que no sea el programador quien se preocupe de asignar los ID de los eventos sino que se le delegue la responsabilidad al SGBD. Para ello habría que:

Modificar el fichero de enlace EventoLogin.xbm.xml y cambiar el modo en que se da el valor a la clave primaria de “assigned” a, por ejemplo, “increment” para que lo haga de manera incremental.

EventoLogin.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 05-dic-2016 17:08:40 by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
    <class name="modelo.dominio.EventoLogin" table="EVENTOLOGIN">
        <id name="id" type="java.lang.Long">
            <column name="ID" />
            <generator class="increment" />
        </id>
...
```

Modificar la clase CrearEventos, añadiendo un nuevo método (sobrecargado porque tiene el mismo nombre, pero diferentes parámetros) para crear los eventos, que no necesite que se le pase el identificador del evento:

```
    private void createAndStoreEventoLogin(String descripcion, Date fecha) {
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();

        EventoLogin e = new EventoLogin();
        e.setDescripcion(descripcion);
        e.setFecha(fecha);
        session.save(e);
        session.getTransaction().commit();
    }
```

Invocar al método `createAndStoreEventoLogin` desde el método principal main de la clase CrearEventos tantas veces como objetos se quiera crear (3 en este caso). Nótese que se han comentado las instrucciones anteriores para que no dé un error de clave primaria repetida.

```
        //e.createAndStoreEventoLogin(1L,"Pepe ha hecho login correctamente", new Date());
        //e.createAndStoreEventoLogin(2L,"Nerea ha intentado hacer login", new Date());
        //e.createAndStoreEventoLogin(3L,"Kepa ha hecho login correctamente", new Date());
        e.createAndStoreEventoLogin("Pepe ha hecho login correctamente", new Date());
        e.createAndStoreEventoLogin("Nerea ha intentado hacer login", new Date());
        e.createAndStoreEventoLogin("Kepa ha hecho login correctamente", new Date());
```

En ese caso, la ejecución de CrearEventos asignaría los siguientes IDs: 4, 5 y 6.

```
Console
<terminated> CrearEventosSinIDs [Java Application] C:\Archivos de programa\Java\jdk1.7.0\bin\javaw.exe (28/10/2013 19:28:30)
Listado de eventos:
Id: 1 Descripcion: Pepe ha hecho login correctamente Fecha: 2013-10-28 19:15:36.0
Id: 2 Descripcion: Nerea ha intentado hacer login Fecha: 2013-10-28 19:15:37.0
Id: 3 Descripcion: Kepa ha hecho login correctamente Fecha: 2013-10-28 19:15:37.0
Id: 4 Descripcion: Pepe ha hecho login correctamente Fecha: 2013-10-28 19:28:30.0
Id: 5 Descripcion: Nerea ha intentado hacer login Fecha: 2013-10-28 19:28:31.0
Id: 6 Descripcion: Kepa ha hecho login correctamente Fecha: 2013-10-28 19:28:31.0
```

1.8.2.- Mostrar información de las sentencias SQL que Hibernate envía a la BD

Lamentablemente, los mensajes de error que se lanzan usando estos frameworks son a veces difíciles de entender, ya que se realizan acciones de manera automática. Lo que es una ventaja, que Hibernate envíe y ejecute sentencias SQL en vez de programarlas, se puede transformar en una desventaja cuando las cosas no funcionan como uno espera. Por ello, conviene conocer qué sentencias SQL se crean y envían a la BD.

Para ello, hay que modificar el fichero de configuración de Hibernate para que muestre las sentencias SQL que se envían, con un formato más legible, añadiendo las siguientes líneas en el fichero hibernate.cfg.xml. Hay que decir que lamentablemente, no se muestran los valores de los parámetros que se envían, por lo que esta facilidad tal vez no sirva en muchos casos.

```
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.format_sql">true</property>
```

Si se ejecuta de nuevo la clase CrearEventos tras haber modificado hibernate.cfg.xml, se puede comprobar cómo antes del “insert” se realiza un “select max(ID)…” que da una idea de cómo realiza Hibernate el método “increment” para asignar valores a la clave primaria.

```
Creación de eventos:
Hibernate:
    select
        max(ID)
    from
        EVENTOLOGIN
Hibernate:
    insert
    into
        EVENTOLOGIN
        (DESCRIPCION, FECHA, USUARIO, LOGIN, ID)
    values
        (?, ?, ?, ?, ?)
```

1.8.3.- Indicar si se quiere trabajar con la BD existente o crear de nuevo el esquema

En la propiedad `hibernate.hbm2ddl.auto` de hibernate.cfg.xml está establecido el valor "update": `<property name="hibernate.hbm2ddl.auto">update</property>`. Es por ello que la primera vez que se ha ejecutado la clase CrearEventos, se ha creado el esquema de la BD e insertado las 3 tuplas. Y las siguientes veces, se ha actualizado la BD anterior e insertado nuevas tuplas. Así funciona la opción "update": se crea el esquema de la BD si no existe, y si existe se trabaja con el esquema anterior y sus datos.

Otra opción posible es "create", la cual siempre crea de nuevo el esquema de la BD, borrando el anterior si existe.

Asignar la opción "create" en la propiedad hibernate.hbm2ddl.auto de hibernate.cfg.xml. En la pestaña SessionFactory del fichero hibernate.cfg.xml => Properties => hibernate.hbm2ddl.auto=> escribir create. O bien modificar lo siguiente en hibernate.cfg.xml:

```
<property name="hibernate.hbm2ddl.auto">create</property>
```

Ejecutar varias veces la clase CrearEventos, y comprobar que siempre crea los eventos 1, 2 y 3, ya que en cada ejecución borra y crea de nuevo el esquema de la base de datos.

1.9.- Trabajar directamente con la BD usando Java JDBC y no Hibernate

Nota: lo que se explica en este apartado no es estrictamente necesario para realizar una aplicación Hibernate.

Se muestra a continuación la clase `VerEventosUsandoJDBC` implementada con Java JDBC, que es la API que permite ejecutar sentencias SQL directamente sobre una BD relacional desde un programa Java.

```
package principal;

import java.sql.*;
import java.util.Date;

public class VerEventosUsandoJDBC {

public static void main(String[] args) {
Connection c;
Statement s;
ResultSet rs;
try {
  Class.forName("com.mysql.jdbc.Driver");
  c = DriverManager.getConnection("jdbc:mysql://localhost/eventos","root","admin");
  s = c.createStatement();
  rs = s.executeQuery("SELECT * FROM EVENTOLOGIN)");
  System.out.println("EVENTOLOGIN (ID, DESCRIPCION, FECHA");
  while (rs.next()){
        Long id = rs.getLong("ID");
        String descripcion = rs.getString("DESCRIPCION");
        Date fecha = rs.getDate("FECHA");
        System.out.println(id+" / "+descripcion+" / "+fecha);
  }
} catch (Exception e) {e.printStackTrace();} }}
```

Run `VerEventosUsandoJDBC` => Comprobar que el resultado en la consola, tras ejecutar la clase anterior es equivalente al siguiente (con las fechas actuales)

```
EVENTOLOGIN (ID, DESCRIPCION, FECHA
1 / Pepe ha hecho login correctamente / 2017-11-24
2 / Nerea ha intentado hacer login / 2017-11-24
3 / Kepa ha hecho login correctamente / 2017-11-24
```

No se necesita saber JDBC para trabajar con Hibernate. Se incluye en este laboratorio por dos razones: 1) para ver un ejemplo sencillo de cómo funciona JDBC:

- Hay que cargar el driver apropiado de la BD:
  `Class.forName("com.mysql.jdbc.Driver")`
- Abrir una conexión con la BD:
  `DriverManager.getConnection("jdbc:mysql://localhost/eventos","root","admin")`
- Ejecutar la pregunta SQL (sobre un Statement) :
  `s.executeQuery("SELECT * FROM EVENTOLOGIN")`
- Y recorrer el resultado tupla a tupla (usando ResultSet);
  `rs.next(), rs.getLong("ID"),...`

y 2) para entender por qué se necesitan algunas de las opciones de configuración en el fichero hibernate.cfg.xml (las que aparecen en amarillo)

2. Añadir otra clase y una asociación entre las dos clases del dominio.

Añadir la clase del dominio Usuario y modificar la clase del dominio anterior (EventoLogin), la cual tiene ahora una asociación con Usuario, mediante el atributo usuario, otro atributo para indicar si ha hecho o no login, y otro atributo que almacena una descripción que se inicializa automáticamente dependiendo del valor de login.

```
package modelo.dominio;

public class Usuario {
        String nombre;
        String password;
        String tipo;

        public Usuario(){}

        public String getTipo() {
                return tipo; }
        public void setTipo(String tipo) {
                this.tipo = tipo; }
        public String getNombre() {
                return nombre; }
        public void setNombre(String nombre) {
                this.nombre = nombre; }
        public String getPassword() {
                return password; }
        public void setPassword(String password) {
                this.password = password; }
}
```

```
package modelo.dominio;

import java.util.Date;

public class EventoLogin {
        private Long id;
        private String descripcion;
        private Date fecha;
        private Usuario usuario;
        private boolean login;

        public EventoLogin() {}

        public Long getId() {
                return id; }
        public void setId(Long id) {
                this.id = id; }
        public String getDescripcion() {
                return descripcion; }
        public void setDescripcion(String descripcion) {
                this.descripcion = descripcion; }
        public Date getFecha() {
                return fecha; }
        public void setFecha(Date fecha) {
                this.fecha = fecha; }
        public Usuario getUsuario() {
                return usuario; }
        public void setUsuario(Usuario usuario) {
                this.usuario = usuario; }
        public boolean isLogin() {
                return login; }
        public void setLogin(boolean login) {
                this.login = login;
                if (login)
                        this.descripcion = usuario.getNombre() +
                                        " ha hecho login correctamente";
                else this.descripcion = usuario.getNombre() +
                                        " ha intentado hacer login"; }
}
```

Volver a generar los ficheros de enlace o mapping

New => Other => Hibernate => Hibernate XML Mapping File => Add Class => Escoger las clases Usuario y EventoLogin.

Asegurarse de que los dos ficheros de enlace están definidos en el fichero de configuración hibernate.cfg.xml

```
<mapping resource="modelo/dominio/EventoLogin.hbm.xml"/>
<mapping resource="modelo/dominio/Usuario.hbm.xml"/>
```

Asegurarse de poner “increment” en el atributo ID de EventoLogin

Modificar la clase principal CrearEventos, añadiendo un nuevo método para insertar usuarios `createAndStoreUsuario`, y para insertar eventos proporcionando la instancia del usuario `createAndStoreEventoLogin`. Es necesario importar la clase Usuario al principio: `import modelo.dominio.Usuario;`

```
private void createAndStoreUsuario(String nombre, String password, String tipo) {
     Session session = HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();

     Usuario u = new Usuario();
     u.setNombre(nombre);
     u.setPassword(password);
     u.setTipo(tipo);
     session.save(u);

     session.getTransaction().commit();
}

private void createAndStoreEventoLogin(String usuario, boolean login, Date fecha) {
     Session session = HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();

     List result = session.createQuery("from Usuario where nombre='"
                                       +usuario+"'").list();

     EventoLogin e = new EventoLogin();
     try {
       e.setUsuario((Usuario)result.get(0));
     }
     catch (Exception ex)
    {System.out.println("Error al crear instancia de EventoLogin: no existe usuario"
                        +ex.toString());}
     e.setLogin(login);
     e.setFecha(fecha);
     session.save(e);

     session.getTransaction().commit();
}
```

Modificar la clase CrearEventos para añadir 1 usuario y 2 eventos, pero antes hay que configurar hibernate.cfg.xml para que se cree de nuevo el esquema de la BDs, ya que las clases EventoLogin y Usuario han cambiado.

Entonces, en hibernate.cfg.xml hay que poner esta opción:

```
<property name="hibernate.hbm2ddl.auto">create</property>
```

E invocar al método `createAndStoreUsuario` y `createAndStoreEventoLogin` desde el método principal main de la clase CrearEventos tantas veces como objetos se quiera crear (1 usuario y 2 eventos, en este caso). Nótese que se han comentado las instrucciones anteriores de creación de eventos para que se visualicen mejor los resultados esperados, aunque ya no habría problemas con las mismas si se pusiera la opción `update` en *hibernate.hbm2ddl.auto* y se volvieran a ejecutar debido a que la opción para el id de EventoLogin (que se llama también "id") es "increment". Sin embargo, en ese caso, sí daría un error la instrucción `e.createAndStoreUsuario("Pepe", "125", "estudiante")` porque la opción para el id de Usuario (que se llama "nombre") es "assigned", y no se puede asignar otro usuario con el nombre `"Pepe"`.

```
        //e.createAndStoreEventoLogin(1L,"Pepe ha hecho login correctamente", new Date());
        //e.createAndStoreEventoLogin(2L,"Nerea ha intentado hacer login", new Date());
        //e.createAndStoreEventoLogin(3L,"Kepa ha hecho login correctamente", new Date());
        //e.createAndStoreEventoLogin("Pepe ha hecho login correctamente", new Date());
        //e.createAndStoreEventoLogin("Nerea ha intentado hacer login", new Date());
        //e.createAndStoreEventoLogin("Kepa ha hecho login correctamente", new Date());

        e.createAndStoreUsuario("Pepe", "125", "estudiante");
        e.createAndStoreEventoLogin("Pepe",true, new Date());
        e.createAndStoreEventoLogin("Pepe",false, new Date());
```

NOTA: para algunas versiones de Hibernate y trabajando con MySQL, es posible que se lance el siguiente error: `Data too long for column 'LOGIN' at row 1` , tal y como se indica en http://stackoverflow.com/questions/3383169/hibernate-jpa-mysql-and-tinyint1-for-boolean-instead-of-bit-or-char

En ese caso entonces cambiad el tipo boolean por NumericBooleanType en el fichero de mapping EventoLogin.xbm.xml. En vez de:

```
        <property name="login" type="boolean">
            <column name="LOGIN" />
        </property>
```

Hay que definirlo así:

```
        <property name="login" type="org.hibernate.type.NumericBooleanType">
            <column name="LOGIN" />
        </property>
```