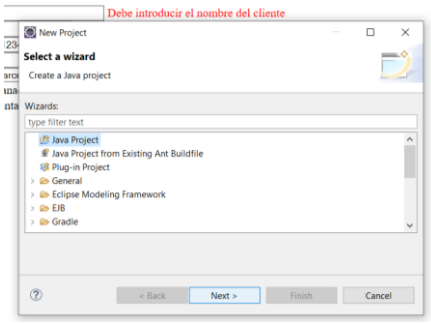
Hibernate

Vamos a instalar el hibernate en un proyecto y lo vamos a probar. Primero sin Spring para entenderlo independientemente del contexto. Así si falla algo sabemos que es del Hibernate y no de otras configuraciones.

Creamos un proyecto Java:

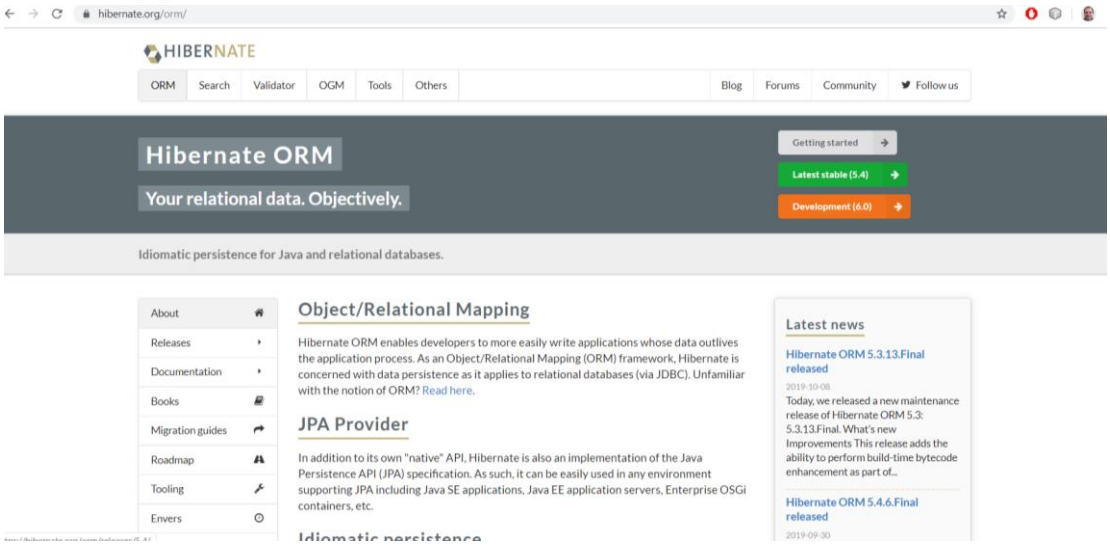


Creamos una carpeta lib para poner los archivos que vamos a necesitar que son:

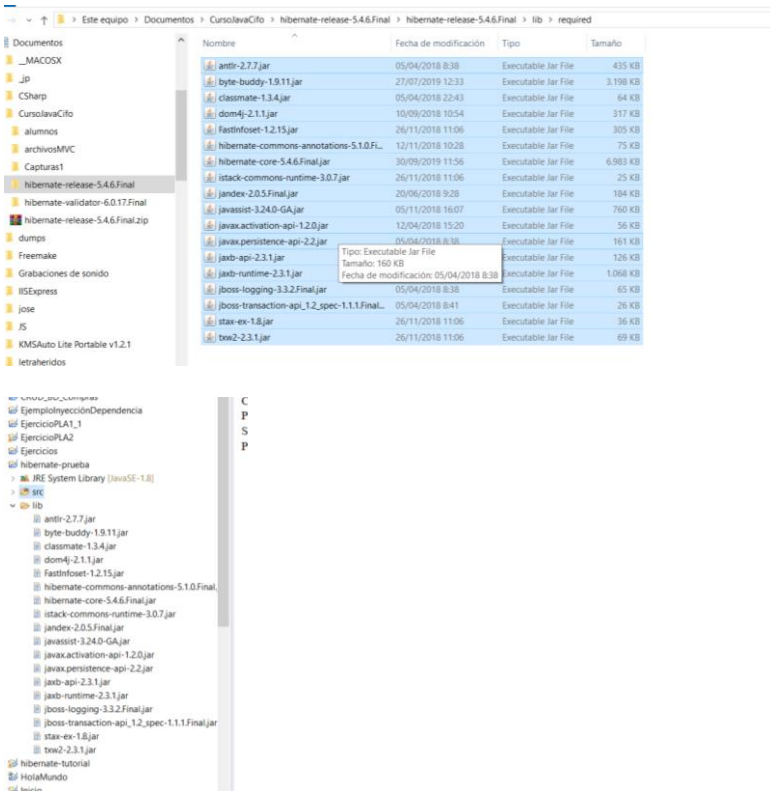
- Jar de Hibernate
- Jar de jdbc para conectar al mysql

Los archivos los obtenemos de aquí:

Hibernate.org/orm

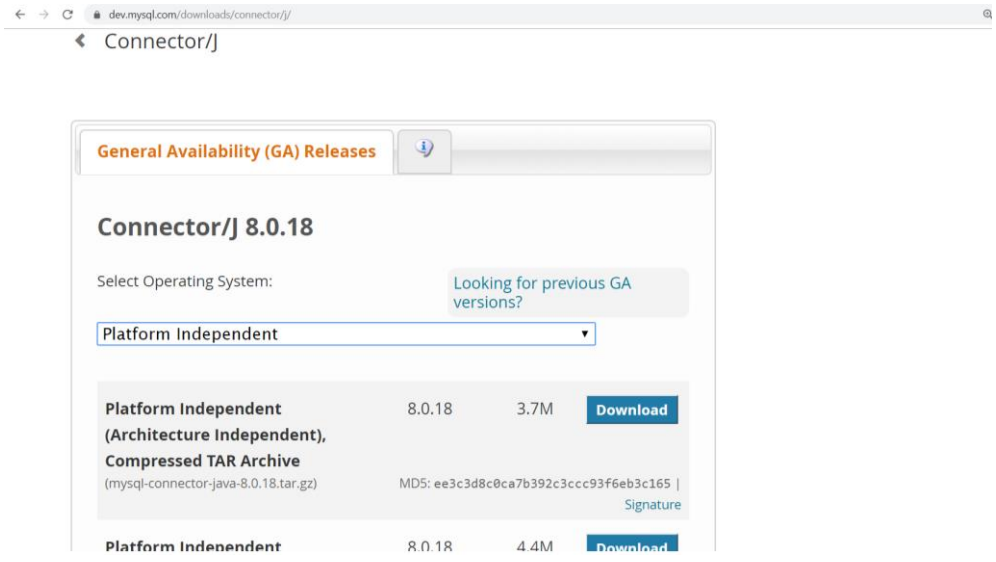


Copiamos los jar dentro de lib/required y los copiamos en nuestra carpeta lib:



Ahora copiaremos el jar de mysql

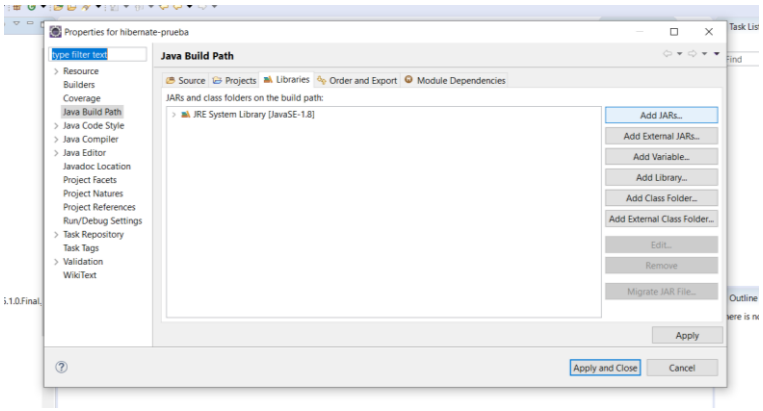
https://dev.mysql.com/downloads/connector/j/



O podemos descargarlos todos de:

https://github.com/juanpablofuentes/Java/tree/master/archivosHibernate

Los añadimos en el builder path:



Podemos crear dentro de nuestro paquete un programa para comprobar si nos funciona bien la conexión jdbc. Hibernate se monta sobre jdbc, si éste no funciona no funcionará hibernate.

Nunca está de más ir comprobando las cosas poco a poco, porque si hacemos un programa complejo que involucra varias partes y nos falla nos costará más averiguar donde está el error. Yo he hecho un paquete para meter una clase que lo único que hace es conectarse a la BD:

```
package com.trifulcas.hibernate;

import java.sql.Connection;
import java.sql.DriverManager;

public class TestJDBC {

	public static void main(String[] args) {
		String bd="empresa";
		String jdbcUrl = 
"jdbc:mysql://localhost:3306/"+bd+"?useSSL=false&serverTimezone=UTC";
		String user = "root";
		String pass = "";

		try {
			System.out.println("Conectando: " + jdbcUrl);

			Connection myConn =
				DriverManager.getConnection(jdbcUrl, user,
pass);

			System.out.println("Todo bien. Circulen.");

		}
		catch (Exception exc) {
			exc.printStackTrace();
		}

	}

}
```

Y como me funciona, sigo adelante ☺

Configurar Hibernate

Vamos a crear en la raíz de src el siguiente archivo:

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- JDBC Database parámetros de conexión -->
        <property
name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost:3306/compras?useSSL=false&amp;se
rverTimezone=UTC</property>
        <property name="connection.username">root</property>
        <property name="connection.password"></property>

        <!-- JDBC connection pool Configuración -->
        <property name="connection.pool_size">1</property>

        <!-- Dialecto SQL -->
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- Que se muestre el SQl que se genera (para debugar) -->
        <property name="show_sql">true</property>

             <!-- Poner el contexto de la sesión -->
             <property name="current_session_context_class">thread</property>

    </session-factory>

</hibernate-configuration>
```

Yo he puesto mis parámetros de conexión, pero vosotros debéis poner los vuestros

Entidades Hibernate, mapeo con tablas

Las entidades van a ser los objetos de Java que serán nuestros intermediarios con las tablas de la base de datos. Para trabajar con los registros utilizaremos las propiedades que nos proporcionan estas entidades.

Tendremos que crear una entidad para mapear una tabla y lo haremos con anotaciones.

Vamos a usar la base de datos ‘comprar’ que vimos en el ejercicio anterior.

Crearé un paquete entidades para añadir la clase Categorias, que mapearé con la tabla categorías:

```
package com.trifulcas.hibernate.entidades;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "categorias")
public class Categorias {

	@Id
	@Column(name = "idcategoria")
	private int idcategoria;

	@Column(name = "nombre")
	private String nombre;

	public Categorias() {

	}

	public Categorias(String nombre) {
		super();
		this.nombre = nombre;
	}

	public int getIdcategoria() {
		return idcategoria;
	}

	public void setIdcategoria(int idcategoria) {
		this.idcategoria = idcategoria;
	}

	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	@Override
	public String toString() {
		return "Categorias [idcategoria=" + idcategoria + ", nombre=" +
nombre + "]";
	}

}
```

La clase es un trasunto de nuestra tabla. Tiene el mismo nombre y los mismos campos. He creado getters y setters para todas las propiedades, un toString que nos servirá para debugar y dos contructores, uno vacío y otro pasando el nombre de la categoría.

Lo nuevo está en estas líneas. La primera nos indica que es una entidad y que se mapea con la tabla categorías:

```
@Entity
@Table(name = "categorias")
```

Las que están encima de los campos nos indican a qué campos estamos haciendo referencia y en el caso del id, que es el id:

```
	@Id
	@Column(name = "idcategoria")
	private int idcategoria;

	@Column(name = "nombre")
	private String nombre;
```

Y con esto ya está, vamos a ver cómo usar esta entidad en nuestro programa.

Crear entorno de sesión para utilizar las entidades

Para utilizar las entidades tenemos que crear una sesión, pero para crear una sesión necesitamos una factoría de sesiones. Tenemos que indicarle dónde está nuestro archivo de configuración y las clases que queremos utilizar.

Esta configuración es un poco espesa, y va cambiando dependiendo de las versiones. Os pongo aquí como se recomienda ahora (he tenido que cambiar el ejemplo que tenía, con lo que es posible que en un futuro cambie también):

```
//Crear la configuración cogíendola del xml y añadiendo la clase Categorias
		Configuration configuration = new
Configuration().configure("hibernate.cfg.xml")
				.addAnnotatedClass(Categorias.class);
		StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder()
				.applySettings(configuration.getProperties());
		//Crear la factoría de sesiones
		SessionFactory factory =
configuration.buildSessionFactory(builder.build());
		// Crear la sesión
		Session session = factory.getCurrentSession();

		try {
		//Aquí irá el código propiamente dicho de las entidades, crear,
leer, etcétera.

		} finally {
```

```
            factory.close();
        }
```

CRUD con entidades

Utilizar las entidades es bastante sencillo. Para todas las operaciones deberemos crear una transacción, hacer la operación que queramos, y hacer un commit (todo ok) de la transacción.

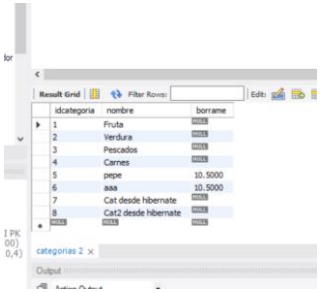
Para dar de alta:

```
package com.trifulcas.hibernate;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;

import com.trifulcas.hibernate.entidades.Categorias;

public class CrearCategoria {

	public static void main(String[] args) {

		// Crear la configuración cogíendola del xml y añadiendo la clase Categorias
		Configuration configuration = new Configuration().configure("hibernate.cfg.xml")
				.addAnnotatedClass(Categorias.class);
		StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder()
				.applySettings(configuration.getProperties());
		// Crear la factoría de sesiones
		SessionFactory factory = configuration.buildSessionFactory(builder.build());
		// Crear la sesión
		Session session = factory.getCurrentSession();

		try {

			// Iniciar transacción
			session.beginTransaction();

			// Creamos una categoría

			Categorias cat = new Categorias("Cat2 desde hibernate");

			// Guardar categoría
			session.save(cat);

			// commit de la transacción
			session.getTransaction().commit();

		} finally {
```

```
                factory.close();
            }
        }

}
```

He puesto todo el código de la clase, en los siguientes ejemplos sólo pondré el código que va dentro del try.

Esto nos ha creado una categoría en la tabla categorías:



Si ejecutamos el programa varias veces vemos que no tenemos problema pero ¿Qué pasa si hacemos lo siguiente?

```
                for (int i = 0; i < 10; i++) {
                    Categorias cat = new Categorias("Cat" + i + " desde
hibernate");

                    // Guardar categoría
                    session.save(cat);
                }
```

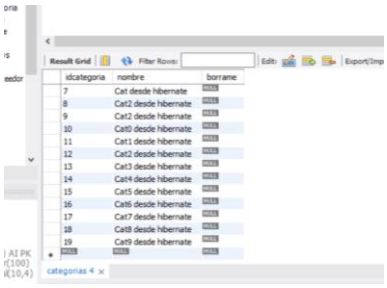
Nos salta el siguiente error:

A different object with the same identifier value was already associated with the session : [com.trifulcas.hibernate.entidades.Categorias#0]

Esto viene porque cuando creamos dos objetos el id está a cero. Hasta que no hagamos el commit no se pondrán los valores. Tenemos dos opciones, o hacemos un commit cada vez que guardemos una categoría o indicar en nuestra clase que el id es un campo autonumérico. Haremos lo segundo, basta con poner lo siguiente en nuestra clase:

```
    @Id
    @Column(name = "idcategoria")
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int idcategoria;
```

Ahora el programa se ejecuta correctamente y el resultado funciona:



Leer una entidad por su ID

Si conocemos la id de una entidad tenemos una manera muy cómoda de recuperarla:

```
	Categorias cat = session.get(Categorias.class, 1);
```

Si queremos hacer una búsqueda más compleja podemos usar el lenguaje Hibernate Query Language (HQL):

```
@SuppressWarnings("unchecked")
		List<Categorias> lista = session
				.createQuery("from Categorias c where
c.nombre like '%cat%' ").getResultList();

		for (Categorias c : lista) {
			System.out.println(c);
		}
```

Pongo el SuppressWarnings para que no me avise del error de tipos del createQuery. El siguiente código nos busca y muestra las anteriores categorías:

```
	session.beginTransaction();

		// Creamos una categoría

		Categorias cat = session.get(Categorias.class, 1);
		System.out.println(cat);

		@SuppressWarnings("unchecked")
		List<Categorias> lista = session
				.createQuery("from Categorias c where
c.nombre like '%cat%' ").getResultList();

		for (Categorias c : lista) {
			System.out.println(c);
		}
		// commit de la transacción
		session.getTransaction().commit();
```

Modificar un registro

Para modificar un registro el proceso es sencillo. Primero lo obtenemos (de la manera que sea, con el id, con query..) modificamos el valor que queramos y lo guardamos. Veamos un ejemplo:

```
session.beginTransaction();

            // Creamos una categoría

            Categorias cat = session.get(Categorias.class, 1);

            cat.setNombre("Deliciosas frutas");
            session.save(cat);
            // commit de la transacción
            session.getTransaction().commit();
```

Eliminar registros

Muy parecido al anterior, obtenemos el registro y lo eliminamos:

```
// Iniciar transacción
            session.beginTransaction();

            // Creamos una categoría

            Categorias cat = session.get(Categorias.class, 19);

            session.delete(cat);
            // commit de la transacción
            session.getTransaction().commit();
```