

Prof. Titular: Mg. María Alejandra Vranić

Prof. Ayudantes: Esp. Lic. Gustavo Siciliano
Lic. Ezequiel Scordamaglia
Lic. Oscar Ruina



Índice

1) Herramientas y Framework Hibernate	2
2) Diagrama de conexiones entre los archivos del proyecto y la DB	3
3) Clase de datos Cliente	4
4) Archivo de mapeo de Cliente	6
5) Archivo de configuración de la conexión con la base de datos	6
6) Clase de configuración de la sesión con la base de datos	7
7) Clase de testeo de la conexión	7
8) Clase de acceso a datos para el objeto Cliente (ClienteDao)	8
9) Clase de lógica de negocio del objeto Cliente (ClienteABM)	10
10) Clase de testeo de la lógica de negocio del objeto Cliente	11
Clase TestAgregarCliente.Java	11
Clase TestActualizarCliente.Java	11
11) Tarea	12

1) Herramientas y Framework Hibernate

IDE: Eclipse

Persistencia de datos: MySQL


Bibliografía: ver programa Hibernate

Framework Hibernate

Gavin King in 2001, crea Hibernate un framework Object-Relational Mapping (ORM) cuyo objetivo es la persistencia de objetos y consultas con un modelo de base de datos relacional y de una Java Application.



En MySQL importar bd-hibernate-una-entidad.sql (la clave primaria debe ser autoincrementable)



Name:

Schema:

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
idCliente	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
apellido	VARCHAR(30)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
nombre	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
dni	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
fechaDeNacimiento	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
baja	BIT(1)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	b'0'

Crear Java Project (\\objetos2\\Hibernate-UnaEntidad)

Guardar **fuera de la carpeta** proyecto, la carpeta lib ejemplo \\objetos2\\lib , esta carpeta va contener todos los archivos .jar; cada vez que hacemos un proyecto vamos a tener que mapear las librerías.

Project → Properties

Java Build Path

Add External JARs

Mapear la carpeta lib y hacer clic en el botón aceptar

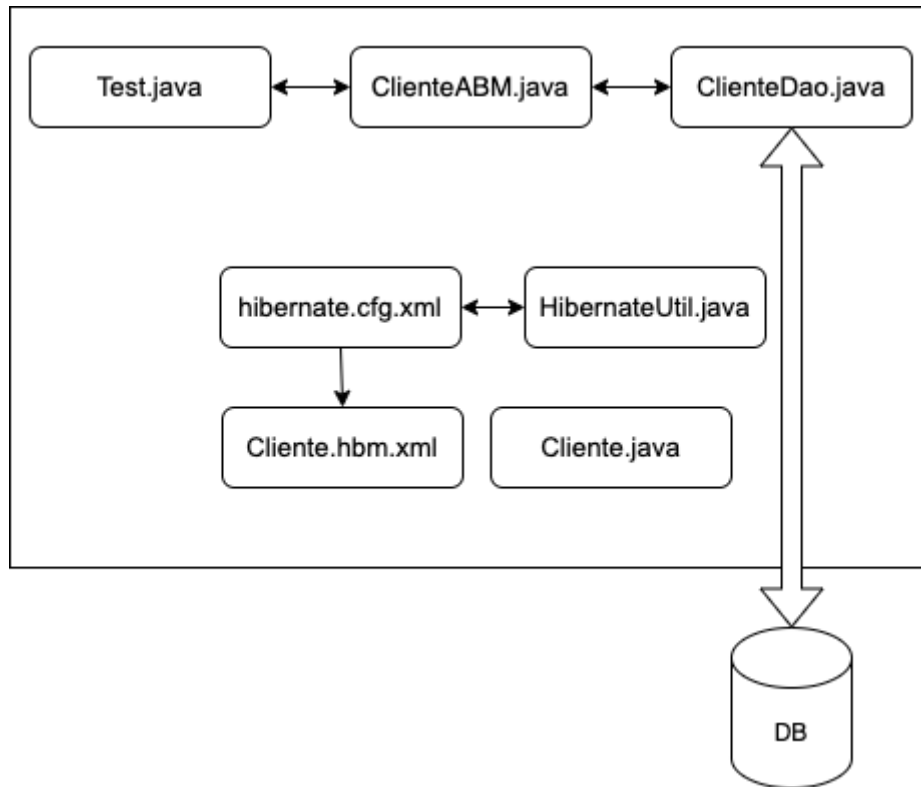
NOTA:

Para bajar la librerías de hibernate con su última versión se puede descargar de su web

<https://sourceforge.net/projects/hibernate/> y para el Connector/ODBC

<http://dev.mysql.com/downloads/connector/j/5.0.html>

2) Diagrama de conexiones entre los archivos del proyecto y la DB



Arquitectura y relación entre los archivos del sistema y la base de datos.

Desde el **Test** se realizan llamados a los métodos de clases de negocio, en este caso **ClienteABM**. Dentro de esta clase se encuentran los métodos relacionados a la lógica de negocio del sistema. A su vez, **ClienteABM** se comunica con la capa de acceso de datos, en el ejemplo llamado **ClienteDao**. La cual es la clase encargada de la programación de consumo de base de datos. Para lograr esto, el proyecto cuenta con archivos de configuración. La clase **HibernateUtil** y el xml **hibernate.hbm** contienen la información para definir la conexión contra la base de datos y especifica las tablas que van a ser utilizadas en el proyecto. Dicha definición, la realiza a través de los xml de asociación entre tabla y clase. En la figura se utiliza el xml **Cliente.hbm** para este fin, pero lo más normal es tener varios xml que representan varias tablas de la base de datos. Finalmente, es importante mencionar que a pesar de que la clase **Cliente** no figura con relaciones en el diagrama, casi todos los archivos la utilizan (a excepción de la clase **HibernateUtil** y el xml **hibernate.hbm**).

3) Clase de datos Cliente

Crear el paquete “datos” y dentro generar la clase “Cliente.java”

```
package datos;
import java.time.LocalDate;

public class Cliente {
    private long idCliente;
    private String apellido;
    private String nombre;
    private int dni;
    private LocalDate fechaDeNacimiento;
    private boolean baja;

    public Cliente() {
    } // siempre hay que implementar el constructor vacío

    // no va el id en el constructor por ser autoincrementable
    public Cliente(String apellido, String nombre, int dni, LocalDate fechaDeNacimiento) {
        super();
        this.apellido = apellido;
        this.nombre = nombre;
        this.dni = dni;
        this.fechaDeNacimiento = fechaDeNacimiento;
        this.baja = false;
    }

    public long getIdCliente() {
        return idCliente;
    }
    // siempre va protected, para que no sea modificado
    protected void setIdCliente(long idCliente) {
        this.idCliente = idCliente;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public String getNombre() {
        return nombre;
    }

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

    public int getDni() {
        return dni;
    }

    public void setDni(int dni) {
        this.dni = dni;
    }

    public LocalDate getFechaDeNacimiento() {
        return fechaDeNacimiento;
    }

    public void setFechaDeNacimiento(LocalDate fechaDeNacimiento) {
```

```
        this.fechaDeNacimiento = fechaDeNacimiento;
    }
    public boolean isBaja() {
        return baja;
    }
    public void setBaja(boolean baja) {
        this.baja = baja;
    }

    @Override
    public String toString() {
        return "Cliente [idCliente=" + idCliente + ", apellido=" + apellido + ", nombre=" + nombre + ",
dni=" + dni + ", fechaDeNacimiento=" + fechaDeNacimiento + ", baja=" + baja + "]";
    }
}
```

4) Archivo de mapeo de Cliente

Crear el paquete "mapeos" y dentro generar un archivo xml llamado "Cliente.hbm.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="datos.Cliente" table="cliente">
    <id column="idCliente" name="idCliente">
      <generator class="identity" />
    </id>
    <property column="apellido" name="apellido" type="string" />
    <property column="nombre" name="nombre" type="string" />
    <property column="dni" name="dni" type="int" />
    <property column="fechaDeNacimiento" name="fechaDeNacimiento"
      type="LocalDate" />
    <property column="baja" name="baja" type="boolean" />
  </class>
</hibernate-mapping>
```

5) Archivo de configuración de la conexión con la base de datos

En la raíz del proyecto crear un archivo xml llamado "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>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">
      jdbc:mysql://localhost/DATA_BASE_NAME</property>
    <property name="connection.username">YOUR_USERNAME</property>
    <property name="connection.password">YOUR_PASSWORD</property>
    <property name="connection.pool_size">1</property>
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="show_sql">>false</property><!-- en true muestra hql en
consola-->
    <!-- Mapeo Entidades -->
    <mapping resource="mapeos/Cliente.hbm.xml" />
  </session-factory>
</hibernate-configuration>
```

6) Clase de configuración de la sesión con la base de datos

Crear el paquete “dao” (Data Access Objects) y dentro generar una clase llamada “HibernateUtil.java”

```
package dao;
import org.hibernate.HibernateException;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class HibernateUtil {
    private static SessionFactory sessionFactory;
    public static SessionFactory getSessionFactory() {
        try {
            if (sessionFactory == null) {
                StandardServiceRegistry standardRegistry = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

                Metadata metaData = new
MetadataSources(standardRegistry).getMetadataBuilder().build();

                sessionFactory = metaData.getSessionFactoryBuilder().build();
            }
        } catch (HibernateException he) {
            System.err.println("ERROR en la inicialización de la SessionFactory:
" + he);

            throw new ExceptionInInitializerError(he);
        }
        return sessionFactory;
    }
}
```

7) Clase de testeo de la conexión

Crear el paquete “test” y dentro generar una clase llamada “TestHBM.java”

```
package test;
import org.hibernate.Session;
import dao.HibernateUtil;

public class TestHBM {

    public static void main(String[] args) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.close();
        System.out.println("OK");
    }
}
```

8) Clase de acceso a datos para el objeto Cliente (ClienteDao)

Dentro del paquete dao crear la clase "ClienteDao.java"

```
package dao;
import java.util.ArrayList;
import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;
import datos.Cliente;

public class ClienteDao {
    private static Session session;
    private Transaction tx;
    private void iniciaOperacion() throws HibernateException {
        session = HibernateUtil.getSessionFactory().openSession();
        tx = session.beginTransaction();
    }
    private void manejaExcepcion(HibernateException he) throws HibernateException {
        tx.rollback();
        throw new HibernateException("ERROR en la capa de acceso a datos", he);
    }
    public int agregar(Cliente objeto) {
        int id = 0;
        try {
            iniciaOperacion();
            id = Integer.parseInt(session.save(objeto).toString());
            tx.commit();
        } catch (HibernateException he) {
            manejaExcepcion(he);
        } finally {
            session.close();
        }
        return id;
    }

    public void actualizar(Cliente objeto) {
        try {
            iniciaOperacion();
            session.update(objeto);
            tx.commit();
        } catch (HibernateException he) {
            manejaExcepcion(he);
        } finally {
            session.close();
        }
    }
}
```



```

public void eliminar(Cliente objeto) {
    try {
        iniciaOperacion();
        session.delete(objeto);
        tx.commit();
    } catch (HibernateException he) {
        manejaExcepcion(he);
    } finally {
        session.close();
    }
}

public Cliente traer(long idCliente) {
    Cliente objeto = null;
    try {
        iniciaOperacion();
        objeto = (Cliente) session.get(Cliente.class, idCliente);
    } finally {
        session.close();
    }
    return objeto;
}

public Cliente traer(int dni) {
    Cliente cliente = null;
    try {
        iniciaOperacion();
        cliente = (Cliente) session.createQuery("from Cliente c where c.dni
= :dni").setParameter("dni", dni)
        .uniqueResult();
        // En este caso :dni es un marcador de posición para el parámetro.
        // Al utilizar el método setParameter para asignar el valor del
        parámetro dni esto ayuda a prevenir la inyección de SQL.
    } finally {
        session.close();
    }
    return cliente;
}

public List<Cliente> traer() {
    List<Cliente> lista = new ArrayList<Cliente>();
    try {
        iniciaOperacion();
        Query<Cliente> query = session.createQuery("from Cliente c order by
c.apellido asc, c.nombre asc", Cliente.class);
        lista = query.getResultList();
    } finally {
        session.close();
    }
    return lista;
}

public List<Cliente> traer(String apellido) {
    List<Cliente> lista = new ArrayList<Cliente>();
    try {
        iniciaOperacion();
        lista = session.createQuery("from Cliente c where c.apellido=:apellido", Cliente.class).setParameter("apellido",apellido).list();
    } finally {
        session.close();
    }
    return lista;
}

```

9) Clase de lógica de negocio del objeto Cliente (ClienteABM)

Crear el paquete “negocio” y dentro generar una clase llamada “ClienteABM.java”

```
package negocio;
import java.time.LocalDate;
import java.util.List;
import dao.ClienteDao;
import datos.Cliente;

public class ClienteABM {

    ClienteDao dao = new ClienteDao();

    public Cliente traer(long idCliente) {
        return dao.traer(idCliente);
    }

    public Cliente traer(int dni) {
        return dao.traer(dni);
    }

    public int agregar(String apellido, String nombre, int dni, LocalDate fechaDeNacimiento) {
        // consultar si existe un cliente con el mismo dni, y si existe, arrojar la Excepcion
        Cliente c = new Cliente(apellido, nombre, dni, fechaDeNacimiento);
        return dao.agregar(c);
    }

    public void modificar(Cliente c) {
        /*
         * En caso de editar el dni, antes de actualizar, validar que no exista un cliente con el mismo
         dni y si eso pasa lanzar la Exception
         */
        dao.actualizar(c);
    }

    public void eliminar(long idCliente) {
        /*
         * En este caso la baja es física y sabemos que la entidad no tiene relaciones
         * pero en caso de tenerlas, hay que validar que el cliente no tenga dependencias que
         generen errores al borrarlo.
         */

        Cliente c = dao.traer(idCliente);
        // Implementar que si es null que arroje la excepción la Excepción de que el cliente no existe
        dao.eliminar(c);
    }

    public List<Cliente> traer() {
        return dao.traer();
    }
}
```

10) Clase de testeo de la lógica de negocio del objeto Cliente

Clase TestAgregarCliente.Java

```
package test;
import java.time.LocalDate;
import negocio.ClienteABM;

public class TestAgregarCliente {

    public static void main(String[] args) {
        ClienteABM abm = new ClienteABM();
        long ultimoIdCliente = abm.agregar("Apellido", "Nombre", 35000000,
LocalDate.now());
        System.out.printf("Id cliente: %d", ultimoIdCliente);
        /*
        En el formateo de Strings con printf tiene varios especificadores de
formato.
        Algunos de los más comunes son:
            %d: Entero con signo (para int y long).
            %f: Número de punto flotante (para float y double).
            %s: Cadena de caracteres.
            %c: Carácter.
            %b: Valor booleano.
        */
    }
}
```

Clase TestActualizarCliente.Java

```
package test;
import datos.Cliente;
import negocio.ClienteABM;
public class TestActualizarCliente {
    public static void main(String[] args) {
        ClienteABM abm = new ClienteABM();

        // traer el obj a modificar
        // Nota: Se usa 'L' al final del número para especificar que de tipo long
        Cliente cliente = abm.traer(1L);
        System.out.printf("Cliente a Modificar: %s\n\n", cliente);

        // modificar por set los atributos
        // Nota: Esto no se hace desde el test, es a modo demostrativo.
        cliente.setDni(35000001);
        abm.modificar(cliente);

        // En este caso se usa el traer por int
        // ya que al no aclarar el tipo (como en el traer anterior) se asume que
es int
        Cliente clienteMod = abm.traer(35000001);
        System.out.printf("Cliente Modificado: %s\n", clienteMod);
    }
}
```

11) Tarea

Queda pendiente implementar los comentarios de los métodos:

1. agregar
2. modificar
3. eliminar

En todos los casos se debe hacer el testeo correspondiente. Comenzar por archivos separados y luego probar un testeo integral uniendo varios métodos, por ejemplo:

- Traer lista de clientes (para su visualización).
- Agregar un cliente nuevo.
- Traer el objeto del primer cliente.
- Modificar algún dato del cliente traído.
- Volver a mostrar la lista de clientes.