UD3: Herramientas de mapeo objeto relacional (ORM)

AD

3ºDAM

Aprenderás a

- Instalar y configurar una herramienta ORM
- Definir los ficheros de mapeo
- Desarrollar aplicaciones para insertar modificar y recuperar objetos
- Realizar consultas con el lenguaje de herramienta ORM
- Gestionar transacciones

Evaluación – Proyecto Final

- El propósito de este trabajo es el desarrollo de una sencilla aplicación que gestione los datos de un grupo de proyectos. Dicha aplicación debe estar dotada de una interfaz gráfica desarrollada en Swing, de manera que facilite su manejo al usuario.
- Se podrá utilizar el entorno de desarrollo que el alumno quiera.

Índice

Índice

Introducción a ORM

- El mapeo objeto-relacional (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional.

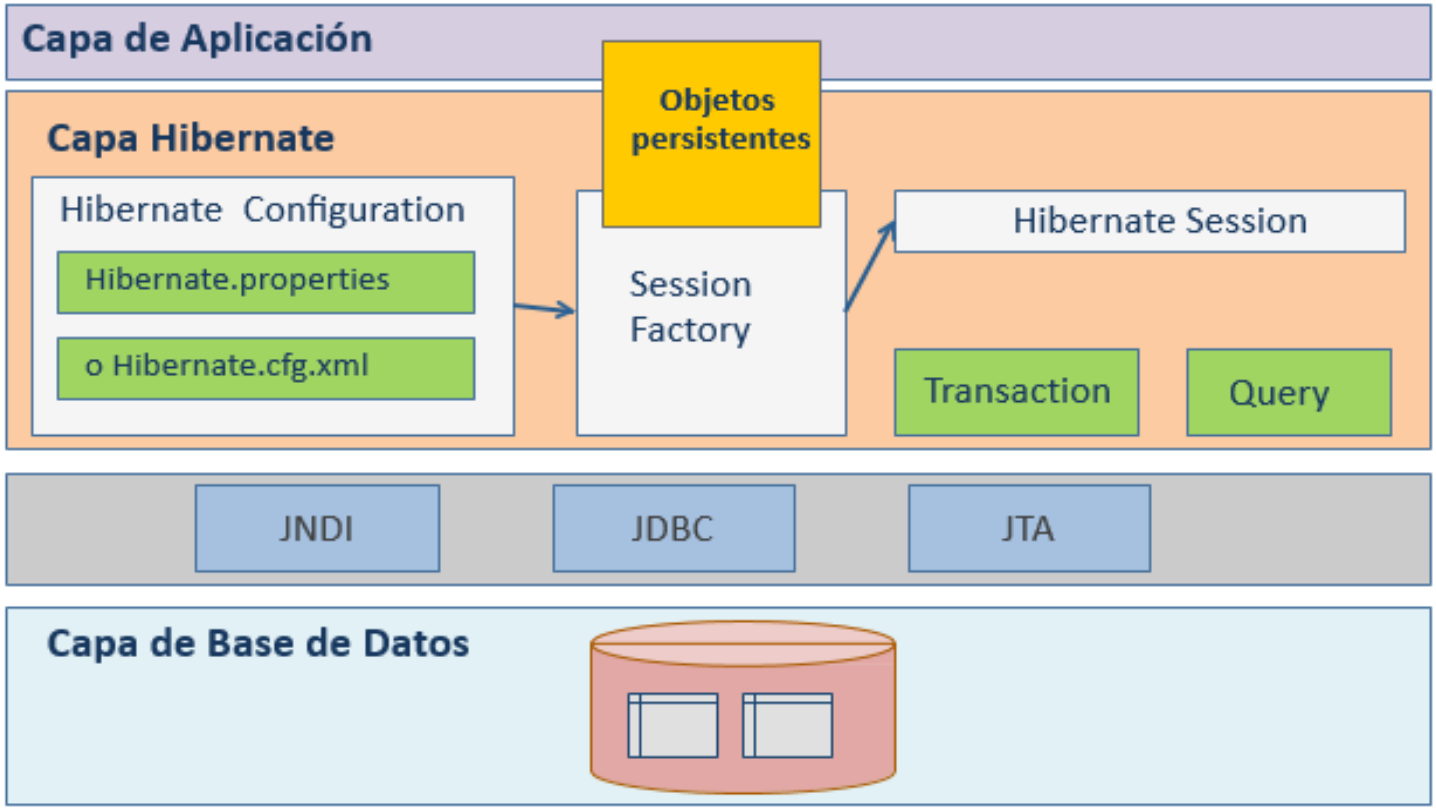
Herramientas ORM

- Características
  - Las herramientas ORM nos permiten crear una capa de acceso a datos: una forma sencilla y válida de hacerlo es crear una clase por cada tabla de la BBDD y mapearlas una a una.
  - El uso de estas herramientas nos permitirá migrar de una base de datos a otra sin tocar nuestro código; solo será necesario cambiar alguna línea del fichero de configuración.
- Ventajas
  - Abstracción de la base de datos
  - Reutilización
  - Son independientes de la base de datos
  - Lenguaje propio para realizar las consultas
  - Facilita la portabilidad de los programas de software
- Inconvenientes
  - Las aplicaciones son algo más lentas porque todas las consultas que se hagan sobre la BBDD, el sistema primero tendrá que transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.

Arquitectura Hibernate

- Para almacenar y recuperar objetos JAVA de la BBDD, el desarrollador debe mantener una conversación con el motor de Hibernate mediante un objeto especial que es la sesión (clase Session) (equiparable al concepto de conexión JDBC).
- Igual que con las conexiones JDBC hemos de crear y cerrar sesiones.
- La clase Session (org.hibernate.Session) ofrece métodos como save(Object objeto), createQuery(String consulta), beginTransaction(), close(), etc. para interactuar con la BBDD.
- Una instancia de Session no consume mucha memoria y su creación y destrucción es muy barata; esto es importante, porque nuestra aplicación necesitará crear y destruir sesiones todo el tiempo, quizás en cada petición.

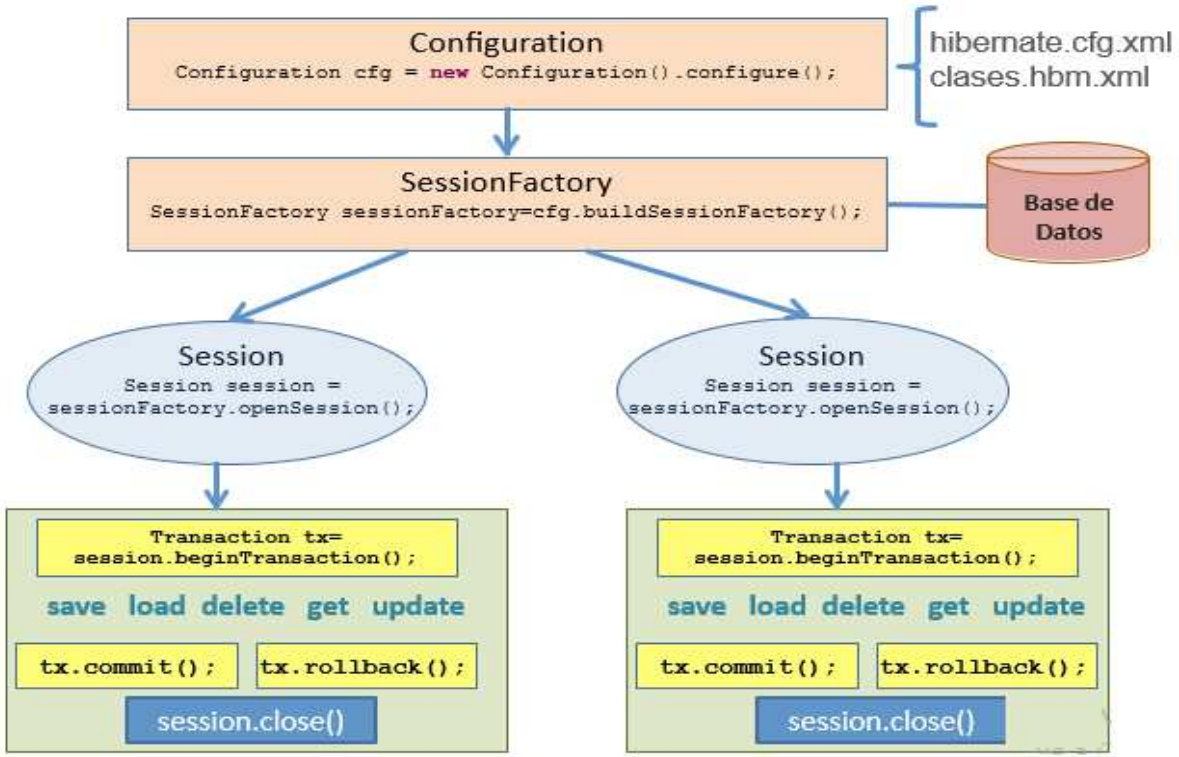
Arquitectura Hibernate



Arquitectura Hibernate
- Interfaces -

- **SessionFactory**
  - Permite obtener instancias Session
  - Se comparte entre muchos hilos de ejecución.
  - Normalmente hay una única SessionFactory para toda la aplicación, creada durante la inicialización de la misma.
  - Si la aplicación accede a varias BBDD se necesitará una SessionFactory por cada BD.
- **Configuration**
  - Se utiliza para configurar Hibernate
  - Indica el mapeado de los objetos y propiedades específicas de Hibernate.
- **Query**
  - Permite realizar consultas a la base de datos
  - Las consultas se escriben en HQL
  - Una instancia Query se utiliza para enlazar los parámetros de la consulta, limitar el número de resultados devueltos por la consulta y para ejecutar dicha consulta.
- **Transaction**
  - Cualquier error que ocurra entre el inicio y final de la transacción producirá un fallo.

Arquitectura Hibernate
- Interfaces -



Instalación y configuración de Hibernate

- Configuración de Hibernate en Eclipse Mars
  - Seguir los pasos de la presentación adjuntada en el Moodle.
- Configuración de Hibernate en Netbeans
  - Seguir los siguientes pasos Configuración Hibernate en Netbeans
  - La versión recomendada es la 8.2 con el jre 1.8
  - Es recomendable utilizar la clase HibenateUtil tal y como aparece en las diapositivas
- Configuración de Hibernate en Intellij
  - Seguir los siguientes pasos: Configuración de Hibernate en Intellij (opción 1) o Configuración de Hibernate en Intellij (opción 2)
  - Es recomendable utilizar la clase HibenateUtil tal y como aparece en las diapositivas

Instalación y configuración de Hibernate
- Clase HibernateUtil -

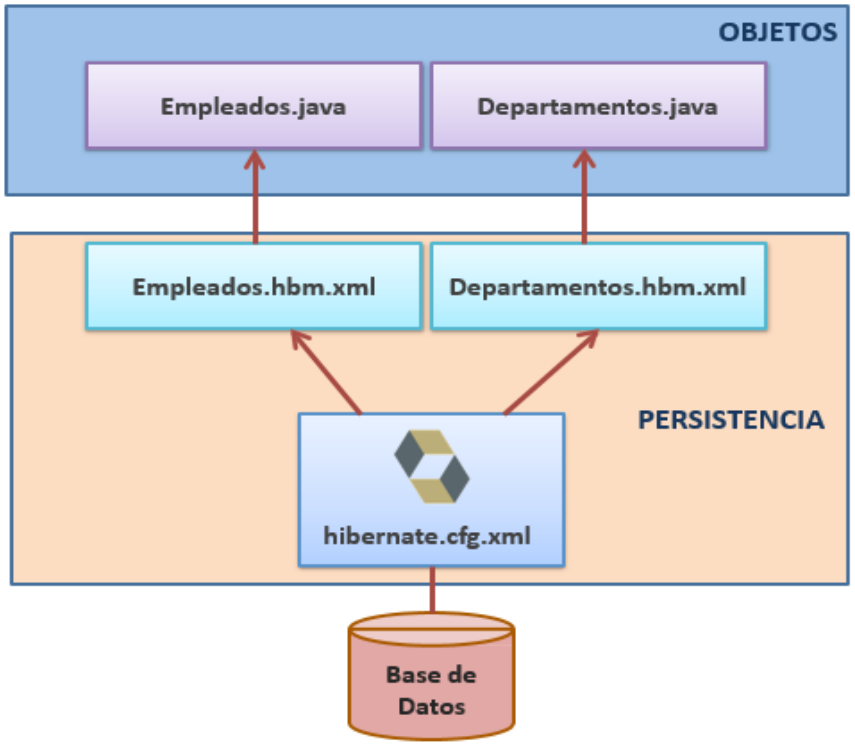
- La clase HibernateUtil proporciona una instancia de la clase SessionFactory a partir del fichero de configuración xml de Hibernate creado durante la fase de configuración de Hibernate.
- Se debe instanciar la clase SessionFactory por cada BBDD a la que nos queramos conectar.
- A partir de la instancia de la clase SessionFactory podremos instanciar objetos de la clase Session que nos permitirán interactuar con la BBDD.

Instalación y configuración de Hibernate
- Clase HibernateUtil -

```
package tema3_ORM.util;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
		private static final SessionFactory sessionFactory = buildSessionFactory();
		private static SessionFactory buildSessionFactory()
		{
			try {
				return new Configuration().configure().buildSessionFactory(
				new StandardServiceRegistryBuilder().configure().build ());
			}
			catch (Throwable ex)
			{
				System.err.println("Initial SessionFactory creation failed." + ex);
				throw new ExceptionInInitializerError(ex);
			}
		}
		public static SessionFactory getSessionFactory()
		{
			return sessionFactory;
		}
}
```

Estructura de los fichero de mapeo




```
<?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 31-oct-2018 16:40:41 by Hibernate Tools 3.5.0.Final -->
<hibernate-mapping>
    <class name="Singleton.Departamentos" table="departamentos" catalog="scotchbox">
        <id name="deptNo" type="byte">
            <column name="dept_no" />
            <generator class="assigned" />
        </id>
        <property name="dnombre" type="string">
            <column name="dnombre" length="15" />
        </property>
        <property name="loc" type="string">
            <column name="loc" length="15" />
        </property>
        <set name="empleadoses" table="empleados" inverse="true" lazy="true"
fetch="select">
            <key>
                <column name="dept_no" not-null="true" />
            </key>
            <one-to-many class="Singleton.Empleados" />
        </set>
    </class>
</hibernate-mapping>
```

Clases persistentes

- Una **clase persistente equivale a una tabla** en una base de datos y **un registro o fila es un objeto** persistente de esa clase.
- A clases Empleados.java y Departamentos.java que hemos generado se les llama **clases persistentes** y deben implementar la interfaz Serializable.
- Para más información ver las clases Empleados.java y Departamentos.java que hemos generado.

```
public class Departamentos  implements java.io.Serializable {

    private int deptNo;
    private String dnombre;
    private String loc;
    private Set empleadoses = new HashSet(0);

    public Departamentos() {
    }
    public Departamentos(int deptNo) {
        this.deptNo = deptNo;
    }
    public Departamentos(int deptNo, String dnombre, String loc, Set empleadoses) {
       this.deptNo = deptNo;
       this.dnombre = dnombre;
       this.loc = loc;
       this.empleadoses = empleadoses;
    }

    public int getDeptNo() {
        return this.deptNo;
    }

    public void setDeptNo(int deptNo) {
        this.deptNo = deptNo;
    }
    public String getDnombre() {
        return this.dnombre;
    }
```

Sesiones y objetos Hibernate

- Para poder utilizar los mecanismos de persistencia de Hibernate se debe inicializar el entorno Hibernate y obtener un objeto Session utilizando la clase **SessionFactory.** En este proceso podemos diferenciar 3 partes:
  - La parte del código en la que se carga el fichero de configuración hibernate.cfg.xml (va en el HibernateUtil)

```
// Inicializa el entorno Hibernate
// Carga el fichero de configuración hibernate.cfg.xml
Configuration cfg =   new Configuration().configure();
```

  - La parte del código en el que se crea el SessionFactory (Recordad que debe haber un SessionFactory por cada almacén (base) de datos (va en el HibernateUtil).

```
// Crea el ejemplar de SessionFactory
SessionFactory sessionFactory = cfg.buildSessionFactory(
    new StandardServiceRegistryBuilder().configure().build() );
```

  - La parte del código en el que se obtiene una session del SessionFactory (va en el programa del desarrollador).

```
// Obtiene un objeto Session
Session session = sessionFactory.openSession();
```

Sesiones y objetos Hibernate
- Transacciones -

- **Al crear la session** se **crea la transacción** para dicha sesión.
- Las transacciones son obligatorias cuando se **inserte, modifique o elimine** algún dato de la BBDD
- **Se deben cerrar las sesiones** cuando se haya completado todo el trabajo de una transacción.
- Métodos:
  - **BeginTransaction():** marca el comienzo de una transacción
  - **Commit():** valida la transacción
  - **Rollback:** deshace la transacción.

```
Session session = sesion.OpenSession();
//crea la transacción
Transaction tx = session.beginTransaction(); //crea la transacción

// Código de persistencia
. . . . . .
tx.commit();                    //valida la transacción
Session.close();    //finaliza la sesión
```

Sesiones y objetos Hibernate
- Estados de un objeto Hibernate -

- **Transitorio (Transient):** Si ha sido recién instaciado utilizando el operador new y **no está asociado** a una Session de Hibernate.
- **Persistente (Persistent):** Cuando ya está **almacenado** en BBDD.
- **Separado (Detached):** Cuando **cerramos la sesión** mediante el método close() de Session.

Sesiones y objetos Hibernate
- Operaciones con el objeto Session. Carga de objetos -

MÉTODO	DESCRIPCIÓN
<T> T load (Class<T> Clase, Serializable id)	Devuelve la instancia persistente de la clase indicada con el identificador dado. La instancia tiene que existir, si no existe el método lanza una excepción.
Object load (String nombreClase, Serializable id)	Similar al método anterior, pero en este caso indicamos en el primer parámetro el nombre de la case de formato de String
<T> T get(Class<T> Clase, Serializable id)	Devuelve la instancia persistente de la clase indicada con el identificador dado. Si la instancia no existe, devuelve null
Object get(String nombreClase, Serializable id)	Similiar al método anterior, pero en este caso indicamos en el primer parámetro el nombre de la clase

La diferencia entre get() y load() es que si el objeto que queremos recuperar no existe, **get()** nos devolverá **null**, mientras que **load()** nos lanzará la excepción **ObjectNotFoundException**.

Carga de objetos
- Método load vs Método get -

```
// Con load

// Visualiza los datos del departamento 20;
Departamentos dep = new Departamentos();

try{
dep = (Departamentos)session.load(Departamentos.class,
(byte) 20);
System.out.printf("Nombre Dep: %s%n,dep.getDnombre());
System.out.printf("Localidad: %s%n, dep.getLoc());
}

catch (ObjectNotFoundException o) {
System.out.printf("NO EXISTE EL DEPARTAMENTO!);
}
```

```
// Con get

// Comprueba que el departamento 11 existe
Departamentos dep = new Departamentos();

dep = (Departamentos)session.get(Departamentos.class,
(byte) 11);

If(dep==null)
{
     System.out.printf("NO EXISTE EL DEPARTAMENTO 11!);
}
else
{
     System.out.printf("Nombre Dep: %s%n,dep.getDnombre());
     System.out.printf("Localidad: %s%n, dep.getLoc());
}
```

Carga de objetos
- Ejemplo con Método load -

```
import java.util.Iterator;
import java.util.Set;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import Singleton.*;

public class ListarDepartamento {

     public static void main(String[] args) {
     // TODO Auto-generated method stub
     SessionFactory sesion =
     HibernateUtil.getSessionFactory();
     Session session = sesion.openSession();

     System.out.println("============================");
     System.out.println("DATOS DEL DEPARTAMENTO 10.");

     Departamentos dep = new Departamentos();
     dep = (Departamentos)
     session.load(Departamentos.class, (byte) 10);

     System.out.println("Nombre Dep:" + dep.getDnombre());
     System.out.println("Localidad:" + dep.getLoc());
     System.out.println("============================");
     System.out.println("EMPLEADOS DEL DEPARTAMENTO 10.");

     //obtenemos empleados
     Set<Empleados> listaemple = dep.getEmpleadoses();
     Iterator<Empleados> it = listaemple.iterator();

     System.out.printf("Número de empleados: %d %n",
     listaemple.size());

     while (it.hasNext())
     {
          Empleados emple = it.next();
          System.out.printf("%s * %.2f %n",
          emple.getApellido(), emple.getSalario());
     }
     System.out.println("============================");
     session.close();
     System.exit(0);
     }

}
```

Sesiones y objetos Hibernate

- Operaciones con el objeto Session. Almacenamiento de objetos -

MÉTODO	DESCRIPCIÓN
Serializable save (Object obj)	Guarda el objeto que se pasa como argumento en la base de datos. Hace que la instancia transitoria del objeto sea persistente.
Void update (Object objeto)	Actualiza en la base de datos el objeto que se pasa como argumento. El objeto a modificar debe ser cargado con el método **load()** o **get()**
Void delete (Object objeto)	Elimina de la base de datos el objeto que e pasa como argumento. El objeto a eliminar debe ser cargado con el método **load() o get()**

Almacenamiento de objetos
- Código -

- A partir de la excepción `ConstraintViolationException` controla que el objeto insertado no esté duplicado
- A partir de la excepción `TransientPropertyValueException` controla que no se inserte un objeto el cual contenga como uno de sus atributos otro objeto que no exista en la BBDD. Con esto mantendría la integridad relacional de la BBDD.
- La estructura básica de código sería la siguiente:

```
// Almacenar en la BBDD

Departamentos dep =new Departamentos();
dep.setDeptNo((byte) 70);
Dep.setDnombre("INFORMATICA");
dep.setLoc("TOLEDO");
Session.save(dep);              //almacena el objeto
```

Almacenamiento de objetos
- Ejemplo de programa -

```
import org.hibernate.SessionFactory;
import org.hibernate.Session;
import org.hibernate.Transaction;
import Singleton.*;
import org.hibernate.exception.ConstraintViolationException;
import org.hibernate.TransientPropertyValueException;

public class InsertarEmpleado {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		SessionFactory sesion = HibernateUtil.getSessionFactory();
		Session session = sesion.openSession();
		Transaction tx = session.beginTransaction();

		System.out.println("Inserto un EMPLEADO EN EL DEPARTAMENTO 10.");

		Float salario = new Float(1500);
		Float comision = new Float(10);

		Empleados em = new Empleados();
		em.setEmpNo((short) 4455);
		em.setApellido("PEPE");
		em.setDir((short) 7499);
		em.setOficio("VENDEDOR");
		em.setSalario(salario);
		em.setComision(comision);
```

Almacenamiento de objetos
- Ejemplo de programa -

```
        //se crea un objeto Departamentos para asignárselo al empleado
        Departamentos d =new Departamentos();
        d.setDeptNo((byte) 10);
        d.setDnombre("MARKETING");
        d.setLoc("GUADALAJARA");
        em.setDepartamentos(d);
        java.util.Date hoy = new java.util.Date();  java.sql.Date fecha = new java.sql.Date(hoy.getTime());
        em.setFechaAlt(fecha);
        session.save(d);
        try
        {     session.save(em);
              try
              {tx.commit();
              } catch (ConstraintViolationException e){
                    System.out.println("EMPLEADO DUPLICADO");
                    System.out.printf("MENSAJE:%s%n", e.getMessage());
                    System.out.printf("COD ERROR:%d%n", e.getErrorCode());
                    System.out.printf("ERROR SQL:%s%n", e.getSQLException().getMessage());}
        }catch (TransientPropertyValueException e){
              System.out.println("EL DEPARTAMENTO NO EXISTE");
              System.out.printf("MENSAJE:%s%n", e.getMessage());
              System.out.printf("Propiedad:%s%n", e.getPropertyName());
        }
        session.close();
        System.exit(0);
    }
}
```

Sesiones y objetos Hibernate

- Operaciones con el objeto Session. Almacenamiento de objetos -

MÉTODO	DESCRIPCIÓN
Serializable save (Object obj)	Guarda el objeto que se pasa como argumento en la base de datos. Hace que la instancia transitoria del objeto sea persistente.
Void update (Object objeto)	Actualiza en la base de datos el objeto que se pasa como argumento. El objeto a modificar debe ser cargado con el método load() o get()
Void delete (Object objeto)	Elimina de la base de datos el objeto que e pasa como argumento. El objeto a eliminar debe ser cargado con el método **load() o get()**

Modificación de objetos
- Código -

- A partir de la excepción `ObjectNotFoundException` se controla que no se pueda modificar un objeto que no exista en la BBDD.
- A partir de la excepción `ConstraintViolationException` se controla que no se asigne a un objeto otro que no exista (como atributo del propio objeto). Con esto mantendría la integridad relacional de la BBDD.
- La estructura básica de código sería la siguiente:

```
// Actualizamos el salario del empleado

Empleados em = new Empleados();
em = (Empleados) session.load(Empleados.class, (short)7369);
//cargamos el objeto
float NuevoSalario = em.getSalario() + 1000;
em.setSalario(NuevoSalario);
Session.update(em);          //actualiza el objeto
```

Modificación de objetos
- Ejemplo de programa -

```
import org.hibernate.ObjectNotFoundException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.exception.ConstraintViolationException;

import Singleton.*;
//import org.hibernate.exception.ConstraintViolationException;
//import org.hibernate.ObjectNotFoundException;

public class ModificarSalario {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		SessionFactory sesion = HibernateUtil.getSessionFactory();
		Session session = sesion.openSession();
		Transaction tx = session.beginTransaction();
		Empleados em = new Empleados();
		try{
			em = (Empleados) session.load(Empleados.class, (short) 7369);
			System.out.printf("Modificacion empleado: %d%n", em.getEmpNo());
			System.out.printf("Salario antiguo: %.2f%n", em.getSalario());
			System.out.printf("Departamento antiguo: %s%n", em.getDepartamentos().getDnombre());

			//nuevo salario
			float NuevoSalario = em.getSalario() + 1000;
			em.setSalario(NuevoSalario);
```

Modificación de objetos
- Ejemplo de programa -

```
            //nuevo departamento
                  Departamentos dep = (Departamentos)session.get(Departamentos.class,(byte) 30);
                  if(dep==null)
                  {
                        System.out.println("El departamento no existe");
                  }
                  else
                  {
                        em.setDepartamentos(dep);
                        session.update(em); //modifica el objeto
                        tx.commit();
                        System.out.printf("Salario nuevo: %.2f%n", em.getSalario());
                        System.out.printf("Departamento nuevo: %d%n", em.getDepartamentos().getDnombre());
                  }
            }catch (ObjectNotFoundException o){
                  System.out.println ("NO EXISTE EL EMPLEADO");
            }catch (ConstraintViolationException c){
                  System.out.println ("NO SE PUEDE ASIGNAR UN DEPARTAMENTO QUE NO EXISTE");
            }catch (Exception e){
                  System.out.println ("ERROR NO CONTROLADO");
                  e.printStackTrace();
            }
      }
}
```

Sesiones y objetos Hibernate
- Operaciones con el objeto Session. Borrado de objetos -

MÉTODO	DESCRIPCIÓN
Serializable save (Object obj)	Guarda el objeto que se pasa como argumento en la base de datos. Hace que la instancia transitoria del objeto sea persistente.
Void update (Object objeto)	Actualiza en la base de datos el objeto que se pasa como argumento. El objeto a modificar debe ser cargado con el método **load()** o **get()**
Void delete (Object objeto)	Elimina de la base de datos el objeto que e pasa como argumento. El objeto a eliminar debe ser cargado con el método load() o get()

Borrado de objetos
- Código -

- A partir de la excepción `ObjectNotFoundException` se controla que no se pueda eliminar un objeto que no exista en la BBDD.
- A partir de la excepción `ConstraintViolationException` se controla que no se elimine un objeto el cual contenga otros objetos asociados (como atributos). Con esto mantendría la integridad relacional de la BBDD.
- La estructura básica de código sería la siguiente:

```
// Borrar el empleado cuyo nº de empleado es 7369

Empleados em = new Empleados();
em = (Empleados) session.load(Empleados.class, (short)7369);
Session.delete(em);          //borra el objeto
```

Borrado de objetos
- Ejemplo de programa -

```
import org.hibernate.Session; import org.hibernate.SessionFactory; import org.hibernate.Transaction;
import Singleton.*;
import org.hibernate.exception.ConstraintViolationException; import org.hibernate.ObjectNotFoundException;

public class BorradoDepartamento {

	public static void main(String[] args) {
		SessionFactory sesion = HibernateUtil.getSessionFactory();
		Session session = sesion.openSession();
		Transaction tx = session.beginTransaction();
		//DEPARTAMENTO A ELIMINAR
		Departamentos de = (Departamentos) session.load(Departamentos.class, (byte) 10);
		try{
			session.delete(de);
			tx.commit();
		} catch (ObjectNotFoundException o){
			System.out.println ("NO EXISTE EL DEPARTAMENTO");
		} catch (ConstraintViolationException c){
			System.out.println ("NO SE PUEDE ELIMINAR, TIENE EMPLEADOS");
		} catch (Exception e){
			System.out.println ("ERROR NO CONTROLADO");
			e.printStackTrace();
		}
		session.close();
		System.exit(0);
	}
}
```

Consultas

- Hibernate soporta HQL
- HQL es una extensión **orientada a objetos** de SQL
- Las consultas HQL y SQL nativas son representadas con una instancia de **org.hibernate.Query**
- Esta interfaz ofrece métodos para:
  - Ligar parámetros
  - Obtener un conjunto de resultados
  - Ejecutar una consulta real

Consultas
- Métodos -

MÉTODO	DESCRIPCIÓN
Iterator iterate()	Devuelve en un objeto Iterator el resultado de la consulta
List list()	Devuelve el resultado de la consulta en un List
Query setFetchSize (int size)	Fija el número de resultados a recuperar en cada acceso a la base de datos al valor indicado en size()
Int executeUpdate()	Ejecuta la sentencia de modificación o borrado. Devuelve el número de entidades afectadas.
String getQueryString()	Devuelve la consulta en un String
Object uniqueResult()	Devuelve un objeto (Cuando sabemos que la consulta devuelve un objeto) o nulo si la consulta no devuelve resultados
Query setCharacter(int posición, char valor) Query setCharacter(String nombre, char valor)	Asigna el valor indicado en el método a un parámetro de tipo char. Posición indica la posición del parámetro dentro de la consulta, empieza en 0. Nombre es el nombre (se indica como :nombre) del parámetro de la consulta
Query setDate (int posición, Date fecha) Query setDate (String nombre, Date fecha)	Asigna la fecha a un parámetro de tipo DATE

Consultas
- Métodos -

MÉTODO	DESCRIPCIÓN
Continuación	
Query setDouble (int posición, double valor) Query setDouble (String nombre, double valor)	Asigna la fecha a un parámetro de tipo DATE
Query setInteger (int posición, integer valor) Query setInteger (String nombre, integer valor)	Asigna valor a un parámetro de tipo entero
Query setString (int posición, String valor) Query setString (String nombre, String valor)	Asigna valor a un parámetro de tipo VARCHAR
Query setParameterList (String nombre, Collection valores)	Asigna una colección de valores al parámetro cuyo nombre se indica en nombre
Query setParameter (int posicion, Collection valores)	Asigna el valor al parámetro indicado en posición
Query setParameter (String nombre, Object valor)	Asigna el valor al parámetro indicado en nombre
Int executeUpdate()	Ejecuta una sentencia UPDATE o DELETE, devuelve el nº de entidades afectadas por la operación

Consultas
- Crear una consulta -

- Crear la consulta
  - Para realizar una consulta → **createQuery** de la interface SharedSessionContract.

```
Query createQuery(String queryString)
Query q = session.createQuery("from departamentos")
```

  - Para recuperar los datos tenemos 2 opciones: List() e Iterate().
    - List():
      - Devuelve una colección de todos los resultados de la consulta:
      - Realiza una única conexión contra la BBDD trayéndose todos los resultados a memoria

```
List <Departamentos> lista = q.list();
```

    - Iterate():
      - Devueve un iterador Java para recuperar los datos de la consulta
      - Obtiene sólo los ids de las entidades

```
Iterator iter = q.iterate();
Iterator.next() //ejecuta la consulta para obtener la entidad completa
setFetchSize() //Fija la cantidad de resultados a recuperar
```

Consultas
- Recuperar datos: List vs Iterator -

```
// Recuperar con Lista

Query q = session.createQuery("from Departamentos");
List <Departamentos> lista = q.list();

// Obtenemos un Iterador y recorremos la lista
Iterator <Departamentos> iter = lista.iterator();
System.out.printf("Número de departamentos: %d%n",
lista.size());
while (iter.hasNext())
{
     //extraer el objeto
     Departamentos depar = (Departamentos) iter.next();
     System.out.printf ("%d, %s%n", depar.getDeptNo(),
     depar.getDnombre());
}
Session.close();
```

```
// Recuperar con Iterator

Query q = session.createQuery("from Departamentos");
q.setFetchSize(10);
Iterator iter = q.iterate();

while (iter.hasNext())
{
     //extraer el objeto
     Departamentos depar = (Departamentos) iter.next();
     System.out.printf ("%d, %s%n", depar.getDeptNo(),
     depar.getDnombre());
}
Session.close();
```

Consultas
- Construir la Query -

- Usando parámetros
  - Hibernate soporta parámetros del estilo JDBC.
  - Numera los parámetros desde 0.
  - Para asignar los valores de los parámetros se utilizan los métodos setXXX
  - La sintaxis más simple es utilizando setParameter()

```
//Muestra el apellido y oficio del empleado con número 7369
String hql = "from Empleados where empNo = :numemple";
Query q = session.createQuery (hql);
q.setParameter("numemple", (short) 7369);
Empleados emple = (Empleados) q.uniqueResult();
System.out.printf ("%s, %s %n",emple.getApellido(), emple.getOficio());

//Con 2 parámetros
String hql2 = "from Empleados emp where emp.departamentos.deptNo = :ndep and emp.oficio = :ofi";
Query q = session.createQuery (hql2);
q.setParameter("ndep", (byte) 10);
q.setParameter("ofi", "DIRECTOR");
```

Consultas
- Construir la Query -

- Usando posiciones
  - También se puede sustituir los parámetros por la posición que estos ocupan dentro de la consulta que se quiere realiza.

```
String hql2 = "from Empleados emp where emp.departamentos.deptNo = ? and emp.oficio = ?";
Query q = session.createQuery (hql);

q.setParameter(0, (byte) 10);
q.setParameter(1, "DIRECTOR");

//También se podría
q.setInteger("ndep", (byte) 10);
q.setString("ofi", "DIRECTOR");
```

Consultas
- Consultas sobre clases no asociadas -

- Si queremos recuperar los datos de una consulta en la que intervienen varias tablas y no tenemos asociada a ninguna clase los atributos que devuelve esa consulta podemos utilizar la clase Object.
- Los resultados se devuelven en un array de objetos, donde el primer elemento se corresponde con la primera clase que ponemos a la derecha del FROM, el siguiente elemento con la siguiente clase y así sucesivamente.

```
String hql = "from Empleados  e, Departamentos d where e.departamentos.deptNo = d.deptNo order by
apellido";

Query cons = session.createQuery (hql);
Iterator q = cons.iterate();

While (q.hasNext()){
     Object[] par = (Object[]) q.next();
     Empleados em = (Empleados) par[0];
     Departamentos de = (Departamentos) par[1];
     System.out.printf ("%s, %.2f, %s, %s %n", em.getApellido(), em.getSalario(), de.getDnombre(),
     de.getLoc());
```

Consultas
- Funciones de grupo en las consultas -

- Los resultados devueltos por una consulta HQL o SQL en la que se ha utilizado una función de grupo como *avg(), sum(), count(),* etc. Se pueden recoger con un único valor utilizando el método ***uniqueResult()***.
- El método ***uniqueResult()*** se utiliza cuando sabemos de antemano que la consulta nos va a devolver un único registro.

```
// MOSTRAR SALARIO MEDIO DE LOS EMPLEADOS
String hql = "select AVG (em.salario) from Empleados as em";
Query cons = session.createQuery(hql);
Double suma = (Double) cons.uniqueResult();
System.out.printf("Salario medio: %.2f%n", suma);
```

- Si en la consulta intervienen varias funciones de grupo y además devuelve varias filas, podemos utilizar objetos devueltos por las consultas.

```
// MOSTRAR SALARIO MEDIO Y EL NÚMERO DE EMPLEADOS POR DEPARTAMENTOS
String hql = "select e.departamentos.deptNo,AVG(salario),count(empNo) from Empleados e group by e.departamentos.deptNo";
Query cons = session.createQuery(hql);
Iterator iter = cons.iterate();
While (iter.hasNext()){
     Object[] par = (Object[])iter.next();
     Byte depar = (Byte) par[0];
     Double media = (Double) par[1];
     Long cuenta = (Long) par[2];
     System.out.printf("Dep: %d, Media: %.2f, N° emp %d %n, depar, media, cuenta);
```

Consultas
- Resumen del Lenguaje HQL -

- La claúsula más simple es from que obtiene todas las instancias de una clase.
- La cláusula where permite refinar la lista de instancias retornadas
- Order by ordena dicha lista
- Podemos asignar alias a las clases usando la cláusula as o sin ella (`from Empleados as emp`)
- Pueden aparecer múltiples clases a la derecha de FROM (`from Empleados as emp, Departamentos as dep`)
- Utilizamos select para obtener determinadas propiedades (columnas).
- Se puede utilizar alias para los atributos y expresiones.
- Las columnas pueden retornar múltiples objetos y/o propiedades como un array de tipo Object[], una lista, una clase, etc.

Consultas
- Insertar, modificar y eliminar objetos mediante consultas -

- Con HQL también podemos hacer **Insert, Update y Delete**.
- (UPDATE | DELETE) [FROM] NombreEntidad [WHERE condición]
- Para ejecutar **UPDATE o DELETE** utilizaremos el método **executeUpdate()**
- El método **executeUpdate()** nos devuelve el nº de entidades afectadas por la operación.

```
//Ejemplo de modificación

//Modificamos el salario de GIL

String hqlModif = "update Empleados set salario =
:nuevoSal where apellido = :ape";

Query q1 = session.createQuery(hqlModif);
q1.setParameter("nuevoSal", (float) 2500.34);
q1.setString("ape", "GIL");

Int filasModif = q1.executeUpdate();

System.out.printf ("FILAS MODIFICADAS: %d %n",
filasModif);
```

```
//Ejemplo de borrado

//Eliminamos los empleados del departamento 20

String hqlDel= "delete Empleados e where
e.departamentos.deptNo = :dep";

Query q = session.createQuery(hqlDel);
q.setInteger("dep", 20);

Int filasDel = q.executeUpdate();

System.out.printf ("FILAS ELIMINADAS: %d %n",
filasDel);
```

Consultas
- Insertar, modificar y eliminar objetos mediante consultas -

- La sentencia insert debe incluir una sentencia select (no se acepta la forma VALUES)
- Por lo tanto deberíamos crear una nueva tabla para insertar los registros que queremos insertar.
- En el caso del ejemplo sería la tabla “Nuevos”
- No es una metodología muy útil

```
//Ejemplo de inserción

//Se inserta un nuevo Departamento

String hqlInsert = "insert into Departamentos (deptNo, dnombre, loc)" + "
select n.deptNo, n.deptNo, n.dnombre, n.loc from Nuevos n";

Query cons = session.createQuery (hqlInsert);

Int filascreadas = cons.executeUpdate();

System.out.printf ("FILAS INSERTADAS: %d %n", filascreadas);
```

Resumen

- **HibernateUtil:** Es la clase que nos permite obtener una **instancia de la clase SessionFactory** a partir del fichero de **configuración** xml de Hibernate creado durante la fase de configuración de la herramienta.
- **SessionFactory: Una conexión por BBDD** (en nuestro caso por programa). A través de **HibernateUtil**.
- **Session**: Se crea un objeto por **cada operación** con la BBDD. Se crea a partir del SessionFactory.
- **Transaction**: Se crea un objeto cuando la operación con la BBDD conlleve una alteración de los datos de la misma (**insertado, modificación o eliminación** de datos). Se crea a partir del **objeto Session**.
- Para realizar inserciones, modificaciones o borrado de datos utilizaremos los métodos de la clase **Session** (**save(), delete(), update()** o los métodos de la clase **Query** (el insert es demasiado complejo).
- Para **recuperar datos** de la BBDD utilizaremos los métodos **get() o load()** de la clase **Session** (siempre y cuando queramos recuperar los datos a partir de Id del objeto) o la clase **List o Iterator** para recuperar los datos a partir de una consulta realizada con la clase **Query** (para recuperar los datos a partir de cualquiera de los otros atributos).