

<b>ACCESO A DATOS</b>	<b>PRÁCTICA HIBERNATE</b>	<b>2 DAM</b>
-----------------------	-------------------------------	--------------

## INDICE

Base de datos:

La BD debe constar de al menos tres tablas relacionadas entre sí.

2.1 Deben existir relaciones 1:N

2.2 Debe existir algún campo autonumérico.

2.3 Tipo enumerado en alguna tabla

2.4 Deben poder realizarse operaciones de inserción, consultar, modificar, eliminar en todas las tablas de la BD

Menú de cada clase.

3.1 Clase cliente

3.2 Clase coche

3.3 Clase revisión

Utilización del patrón DAO.

Gestión de transacciones

Opciones no obligatorias

6.2 Tipo fecha en alguna tabla, con consultas donde se maneje este tipo.

Consultas que se realizan contra la BD, indicando si son HQL consultas nativas.

Opciones de cascade que se han configurado, explicando la causa de elección.

Dificultades encontradas en el desarrollo.

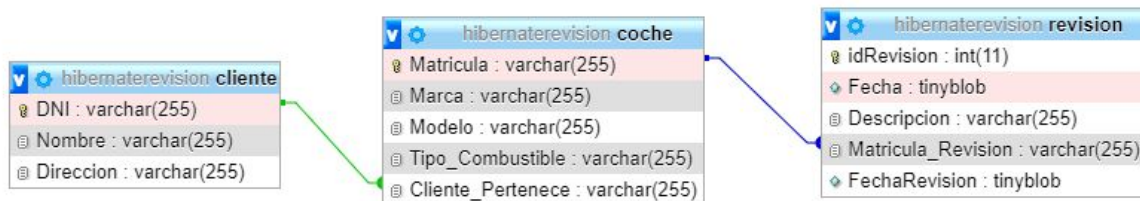
9.1. A la hora de añadir los JARs.

9.2. Llamar al toString()

9.3. Sentencia query

Decisiones de optimización o cualquier otra decisión de desarrollo o aclaración que interese destacar.

## 1. Base de datos:



## 2. La BD debe constar de al menos tres tablas relacionadas entre sí.

### 2.1 Deben existir relaciones 1:N

Todas tiene relaciones 1:N

### 2.2 Debe existir algún campo autonumérico.

En la clase Revisión el idRevisión es autonumerico.

### 2.3 Tipo enumerado en alguna tabla

La tabla Coche tiene el tipo enumerado que se llama Tipo\_Combustible el cual tiene tres tipo de combustible un coche  
ELECTRICO | DIESEL | GASOLINA

### 2.4 Deben poder realizarse operaciones de inserción, consultar, modificar, eliminar en todas las tablas de la BD

En cada una de las tabla se puede insertar datos.  
En cada una de las tabla se puede consultar datos.  
En cada una de las tabla se puede modificar datos.  
En cada una de las tabla se puede eliminar datos.

<b>ACCESO A DATOS</b>	<b>PRÁCTICA HIBERNATE</b>	<b>2 DAM</b>
-----------------------	-------------------------------	--------------

### 3. Menú de cada clase.

#### 3.1 Clase cliente

```
System.out.println("|1| Dar de alta a un cliente.");
System.out.println("|2| Modificar el nombre de un cliente.");
System.out.println("|3| Borrar a un cliente por el dni.");
System.out.println("|4| Consultar Cliente por un nombre.");
System.out.println("|5| Consultar todos los clientes.");
System.out.println("|6| Salir.");
```

#### 3.2 Clase coche

```
System.out.println("|1| Dar de alta a un Coche.");
System.out.println("|2| Modificar modelo de un coche.");
System.out.println("|3| Borrar coche por su matricula.");
System.out.println("|4| Consultar lista de revision de un coche por matricula.");
System.out.println("|5| Consultar todos los Coches.");
System.out.println("|6| Salir.");
```




#### 3.3 Clase revisión

```
System.out.println("|1| Dar de alta a una revision.");
System.out.println("|2| Modificar fecha de una revision.");
System.out.println("|3| Borrar revision por su id.");
System.out.println("|4| Consultar revision de un cliente.");
System.out.println("|5| Consultar todas las revisiones.");
System.out.println("|6| Salir.");
```

<b>ACCESO A DATOS</b>	<b>PRÁCTICA HIBERNATE</b>	<b>2 DAM</b>
-----------------------	-------------------------------	--------------

## 4.Utilización del patrón DAO.

Cada clase tiene su patrón DAO que todas extiendes de la clase GenericDAO()

 ClienteDao.java	 CocheDao.java	 RevisionDao.java
<p>Tiene 3 método Cliente buscarClientePorDni(String)</p> <p>List&lt;Cliente&gt; consultarClientesPorNombre(String)</p> <p>List&lt;Cliente&gt; consultarTodosLosClientes()</p>	<p>Tiene 3 método void cambiarMatricula (Coche ,String)</p> <p>Coche buscarCochePorMatricula (String)</p> <p>List&lt;Coche&gt; consultarTodosLosCoche s()</p>	<p>Tiene 3 método Revision consultarRevisionPorid(in t)</p> <p>Cliente buscarClientePorDni(Strin g)</p> <p>List&lt;Revision&gt; consultarTodasRevisione s()</p>

## 5. Gestión de transacciones

En la clase GenericDAO

Utilización de beginTransaction() y getTransaction

```
public class GenericDAO<T> {  
    public void guardar(T entidad) {  
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
        session.beginTransaction();  
        session.save(entidad);  
        session.getTransaction().commit();  
    }  
    public void borrar(T entidad) throws RevisionException{  
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
        session.beginTransaction();  
        session.delete(entidad);  
        session.getTransaction().commit();  
    }  
}
```

## 6. Opciones no obligatorias

6.2 Tipo fecha en alguna tabla, con consultas donde se maneje este tipo.

En la clase revision hay una varia de tipo LocalDate fechaRevision

```
@Column(name = "FechaRevision")  
private LocalDate fechaRevision;
```

Consulta

1. Cliente
2. Coche
3. Revision
4. Salir

- 1| Dar de alta a una revision.
- 2| Modificar fecha de una revision.
- 3| Borrar revision por su id.
- 4| Consultar revision de un cliente.
- 5| Consultar todas las revisiones.
- 6| Salir.

Donde se pedirá el id y la matrícula del coche

En el caso que el id que se introduzca no exista en la base de datos. Aparezca el siguiente mensaje

```
|2| Modificar fecha de una revision.
Ingrese el id de la revision que cambiara la fecha:
5
Ingresa la matricula del coche:
1258MPK
Hibernate: select coche0_.Matricula as Matricula1_, co
Hibernate: select revision0_.idRevision as idRevision2.
Vaya. Parece que no existe el id 5
```

Y por último también si la matrícula introducida no existe en la base de datos. Aparece el siguiente mensaje.

```
Ingresa la matricula del coche:
1234
Hibernate: select coche0_.Matricula as Matricula1_, coche0_.Cli
Vaya. Parece que no existe ese coche con esa matricula 1234
```

En el caso de que todos los datos sean correctos

```
Revision [idRevision=4, fechaRevision=2021-02-02, descripcion=Presion de los neumaticos, coche=Coche
matricula=1258MPK
marca=Seat
modelo=Leon
cliente=Cliente
dni=29658978Y
    nombre=Manuel
    direccion=Calle conde
    listCoches=1

Tipo combustible ELECTRICO
listRevisiones= 2

Procedemos a cambiar la fecha
Ingresa el día [NUEVA FECHA]
16
Ingresa el mes [NUEVA FECHA]
02
Ingresa el año [NUEVA FECHA]
2021
Hibernate: update Revision set Matricula_Revision=?, Descripcion=?, FechaRevision=? where idRevision=?
Fecha de la revision cambiada correctamente.

Revision [idRevision=4, fechaRevision=2021-02-16, descripcion=Presion de los neumaticos, coche=Coche
matricula=1258MPK
marca=Seat
modelo=Leon
cliente=Cliente
dni=29658978Y
    nombre=Manuel
    direccion=Calle conde
    listCoches=1

Tipo combustible ELECTRICO
listRevisiones= 2
```

Los cambios se modifican correctamente.

ACCESO A DATOS	PRÁCTICA HIBERNATE	2 DAM
----------------	-----------------------	-------

## 7.Consultas que se realizan contra la BD,indicando si son HQL consultas nativas.

Todas las consultas que se han realizado son **HQL**.

## 8.Opciones de cascade que se han configurado,explicando la causa de elección.

### Clase Cliente

```
@OneToMany(cascade = CascadeType.ALL)
@JoinColumn(name = "Cliente_Pertenece")
private List<Coche> listCoches;
```

**CascadeType.ALL** para cuando algo se modifica. En la listCoche también será modificado o eliminado.

### Clase Coche

```
@ManyToOne(fetch = FetchType.LAZY)
@Cascade(org.hibernate.annotations.CascadeType.REFRESH)
@JoinColumn(name = "Cliente_Pertenece")
```

**CascadeType.SAVE\_REFRESH** para cuando se borre o se modifique un coche no se borre ni modifique el cliente.

```
@OneToMany
@Cascade(org.hibernate.annotations.CascadeType.ALL)
@JoinColumn(name = "Matricula_Revision")
```

**CascadeType.ALL** para cuando modifique o borre un coche, la lista de revisiones se modifique o se borre

### Clase Revision



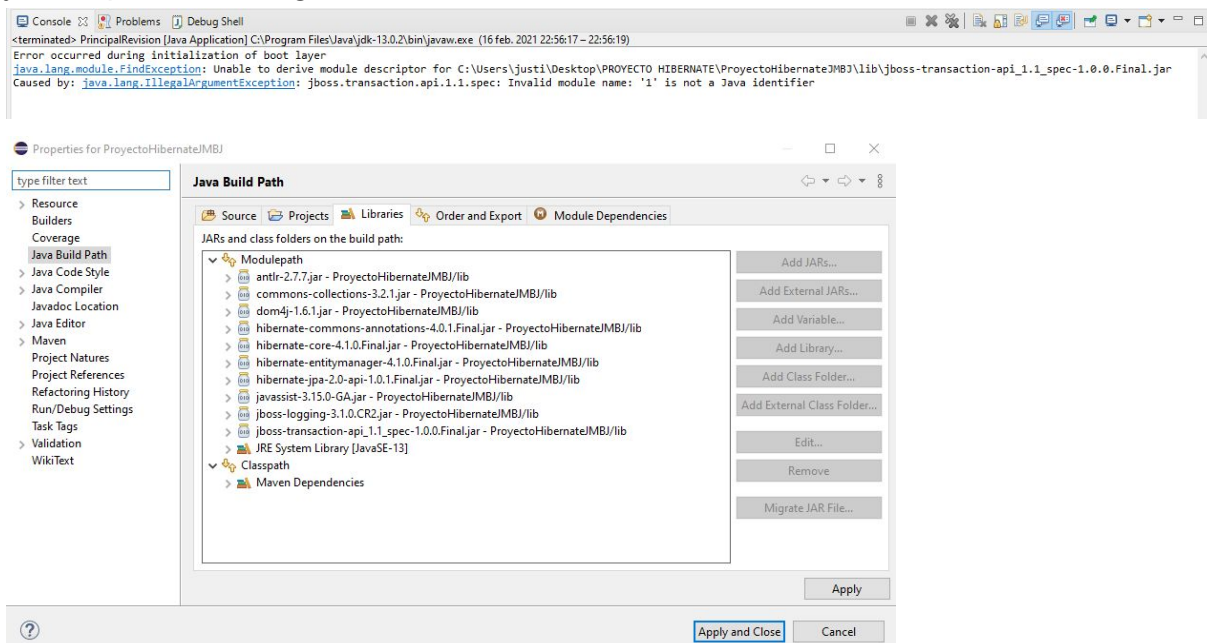
```
@ManyToOne()
@Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
@JoinColumn(name = "Matricula_Revision")
private Coche coche;
```

en este caso he decidido ponerle **CascadeType.SAVE\_UPDATE** para cuando se borre o se modifique una revisión no se borre ni modifique el coche.

## 9. Dificultades encontradas en el desarrollo.

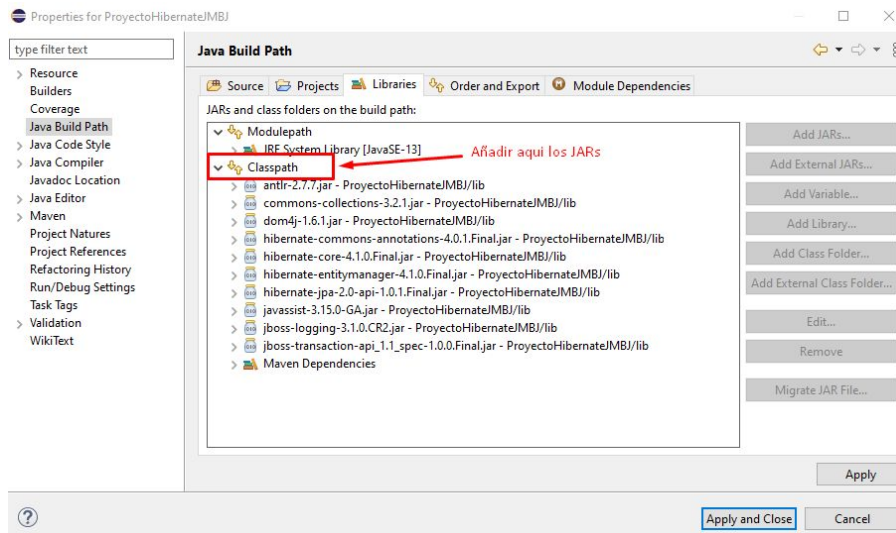
### 9.1. A la hora de añadir los JARs.

Lo estaba agregando en el Modulepath  
y me aparecía lo siguiente



Para solucionar este problema fue necesario borrar del modulepath y agregarlo donde tenían que estar los JARs.





## 9.2. Llamar al toString()

al hacer una consulta a todos los clientes.

```
[5] Consultar todos los clientes.
Hibernate: select cliente0_.DNI as DNI0_, cliente0_.Direccion as Direccion0_, cliente0_.Nombre as Nombre0_ from Cliente cliente0_
Hibernate: select listcoches0_.Cliente_Pertenece as Cliente4_0_1_, listcoches0_.Matricula as Matricula1_, listcoches0_.Matricula as
Exception in thread "main" java.lang.StackOverflowError
    at java.base/java.lang.String.valueOf(String.java:3352)
    at java.base/java.lang.StringBuilder.append(StringBuilder.java:166)
    at java.base/java.util.AbstractCollection.toString(AbstractCollection.java:457)
    at org.hibernate.collection.internal.PersistentBag.toString(PersistentBag.java:501)
    at java.base/java.lang.String.valueOf(String.java:3352)
    at java.base/java.lang.StringBuilder.append(StringBuilder.java:166)
    at clases.Cliente.toString(Client.java:127)
    at java.base/java.lang.String.valueOf(String.java:3352)
    at java.base/java.lang.StringBuilder.append(StringBuilder.java:166)
    at clases.Coche.toString(Coche.java:129)
```

descubrí que no tenía el tamaño .size();

```
@Override
public String toString() {

    return "Cliente \ndni=" + this.dni + ""
        + "\n\tnombre=" + this.nombre + ""
        + "\n\tdireccion=" + this.direccion + ""
        + "\n\tlistCoches=" + this.listCoches
        + "\n";
}
```

Una vez encontrado el fallo el método toString() funciona correctamente.  
Clase Cliente

```

@Override
public String toString() {
    return "Cliente \ndni=" + this.dni + ""
        + "\n\tnombre=" + this.nombre + ""
        + "\n\tdireccion=" + this.direccion + ""
        + "\n\tlistCoches=" + this.listCoches.size()
        + "\n";
}

```

Para la clase coche

```

@Override
public String toString() {
    return "Coche [matricula=" + matricula + ", marca=" + marca + ", modelo=" + modelo + ", cliente=" + cliente
        + ", listRevisiones=" + listRevisiones.size() + "];"
}

```

tuve que modificar hibernate.cfg.xml

```

<property name="hibernate.hbm2ddl.auto">update</property>

```

## 9.3. Sentencia query

### 9.3.1 Al realizar esa sentencia no pillaba el \*

```

public static List<Cliente> consultarClientesPorNombre(String nombre){
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    List<Cliente> listClienteNombres;

    Query sentencia = session.createQuery("SELECT * FROM Cliente WHERE Nombre = '"+nombre+"'");
    //Nos devuelve una lista con los nombre que sea igual
    listClienteNombres = sentencia.list();
    return listClienteNombres;
}

```

```

Exception in thread "main" org.hibernate.hql.internal.ast.QuerySyntaxException: unexpected token: " near line 1, column 8 [SELECT * FROM clases.Cliente WHERE Nombre = 'jose']
at org.hibernate.hql.internal.ast.QuerySyntaxException.convert(QuerySyntaxException.java:54)
at org.hibernate.hql.internal.ast.QuerySyntaxException.convert(QuerySyntaxException.java:47)
at org.hibernate.hql.internal.ast.ErrorCounter.throwQueryException(ErrorCounter.java:79)
at org.hibernate.hql.internal.ast.QueryTranslatorImpl.parse(QueryTranslatorImpl.java:276)
at org.hibernate.hql.internal.ast.QueryTranslatorImpl.doCompile(QueryTranslatorImpl.java:180)
at org.hibernate.hql.internal.ast.QueryTranslatorImpl.compile(QueryTranslatorImpl.java:136)

```

Para solucionarlo fue necesario añadir a Cliente un alias "c"

```

public static List<Cliente> consultarClientesPorNombre(String nombre){
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    List<Cliente> listClienteNombres;

    Query sentencia = session.createQuery("SELECT c FROM Cliente c WHERE Nombre = '"+nombre+"'");
    //Nos devuelve una lista con los nombre que sea igual
    listClienteNombres = sentencia.list();
    return listClienteNombres;
}

```

```
|4| Consultar Cliente por un nombre.
Ingrese el nombre del cliente que desea consultar:
antonio
Hibernate: select cliente0_.DNI as DNI0_, cliente0_.Direccion as Direccion0_, cliente0_.Nombre as Nombre0_ from Cliente cliente0_ where Nombre='antonio'
Cliente
dni=30286639M
nombre=antonio
direccion=clipper
listCoches=0
```

Mismo error en la siguiente sentencia y misma solución.

```
public static List<Cliente> consultarTodosLosClientes(){
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    List<Cliente> listClientes;

    Query sentencia = session.createQuery("SELECT c FROM Cliente c");

    listClientes= sentencia.list();
    return listClientes;
}
```

## 10. Decisiones de optimización o cualquier otra decisión de desarrollo o aclaración que interese destacar.

Añadir Tag `<query>`

Existe dos manera de hacerlo

**name:** Este atributo define el nombre de la consulta. Es el nombre que posteriormente usaremos desde el código Java para acceder a la consulta.

**contenido:** El contenido del tag `<query>` es la consulta en formato **HQL** que ejecutará Hibernate.

Primera forma es creando un **XML**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
>
<hibernate-mapping>
>
    <class name="clases.Cliente">
        <id column="Id" name="DNI" />
        <property name="Nombre" />
        <property name="Direccion" />

    </class>

>
    <query name="Cliente.findAllClientes"><![CDATA[
        SELECT c FROM Cliente c
    ]]></query>
</hibernate-mapping>
```

Segunda forma es añadirla en la propia clase

```

@Entity()
@Table (name = "Cliente")
@NamedQueries ({
    @NamedQuery ( name = "Cliente.findAll",
                  query = "SELECT c FROM Cliente c")
})
public class Cliente implements Serializable {

```

#### Posibles Errores

```
name = "Cliente.findAll"
```

En el método no tiene el mismo nombre.

```

|5| Consultar todos los clientes.
Exception in thread "main" org.hibernate.MappingException: Named query not known: findAllClientes
at org.hibernate.internal.AbstractSessionFactory.<init>MethodQuery(AbstractSessionFactory.java:148)

```

#### Para utilizarla

1. llama al método getNameQuery () que se le pasa el nombre mediante el tag <query>.
2. se ha definido la consulta llamada findAllClientes

```

public static List<Cliente> consultarTodosLosClientes(){
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    List<Cliente> listClientes;

    Query sentenciaQuery = session.getNameQuery("findAllClientes");

    listClientes= sentenciaQuery.list(); 1
    return listClientes;
}

```

## Dos versiones una grafica con Java FX y otra por consola.

Lo he realizado en JavaFX para tener dos versiones una seria por consola y la otra mas graficamente. Aquí dejo el link al proyecto.

LINK =

[https://drive.google.com/file/d/1ok9w9QmyWlx-JKaMxSbVORNh8Z1W\\_i2y/view?usp=sharing](https://drive.google.com/file/d/1ok9w9QmyWlx-JKaMxSbVORNh8Z1W_i2y/view?usp=sharing)