2. Objetos persistentes

2.1. Estructura de los ficheros de mapeo

Hibernate utiliza unos ficheros de mapeo para relacionar las tablas de la base de datos con los objetos Java. Estos ficheros están en formato XML y tienen la extensión **.hbm.xml**. En el proyecto anterior se crearon dos ficheros de mapeo: **Emp.hbm.xml** y **Depto.hbm.xml**, el primero asociado a la tabla Emp y el segundo asociado a la tabla Depto. Estos ficheros se guardan en el mismo directorio que las clases Java *Emp.java* y *Depto.java*.

Por ejemplo, la estructura del fichero Depto.hbm.xml es la siguiente:

```
<hibernate-mapping>
    <class catalog="empleados" name="ejemplo.Depto" optimistic-lock="none" table="depto">
        <id name="numdep" type="int">
            <column name="numdep"/>
            <generator class="assigned"/>
        </id>
        <many-to-one class="ejemplo.Emp" fetch="select" name="emp">
            <column name="numjefe"/>
        </many-to-one>
        <property name="nomdep" type="string">
            <column length="50" name="nomdep" not-null="true"/>
        </property>
        <property name="localidad" type="string">
            <column length="50" name="localidad" not-null="true"/>
        </property>
        <set fetch="select" inverse="true" lazy="true" name="emps" table="emp">
            <key>
                <column name="numdep"/>
            </key>
            <one-to-many class="ejemplo.Emp"/>
        </set>
    </class>
</hibernate-mapping>
```

Todos los ficheros de mapeo empiezan y acaban con la etiqueta <hibernate-mapping>.

La etiqueta <class> engloba a la clase con sus atributos, indicando siempre el mapeo a la tabla de la base de datos. En *name* se indica el nombre de la clase y en *table* el nombre de la tabla. En *catalog* se indica el nombre de la base de datos.

Dentro de *class* distinguimos la etiqueta <id>, en la cual se indica en *name* el campo que representa al atributo clave en la clase, en *column* su nombre en la tabla de la BD y en *type* el tipo de dato. El atributo *generator* indica la naturaleza del campo clave. Puede tomar, principalmente, los siguientes valores:

- **identity**: de tipo *auto_increment* en MySQL.
- **assigned**: su valor es asignado por la aplicación.
- **foreign**: usa el identificador de otro objeto asociado.

El resto de atributos se indican en las etiquetas <property>, asociando el nombre del campo de la clase con el nombre de la columna de la tabla y el tipo de dato. Los tipos de dato que se declaran y utilizan en los ficheros de mapeo no son tipos Java ni tampoco tipos SQL. Estos tipos se llaman "tipos de mapeo Hibernate" y son convertidores que pueden traducir tipos de datos Java a SQL y viceversa. Hibernate tratará de determinar el tipo correcto de conversión y de mapeo por sí mismo.

Hay otro tipo de etiquetas que mapean las relaciones entre las diferentes tablas de la base de datos. Las relaciones pueden ser:

- De uno a muchos (1:N): se representan con las etiquetas <many-to-one> y <one-to-many>. En el ejemplo, tenemos la etiqueta <many-to-one>, que indica que un departamento solo tendrá un jefe pero un empleado puede ser jefe de varios departamentos.
- De uno a uno (1:1): se representa con la etiqueta <one-to-one>. Es una relación 1:1.
- De muchos a muchos (N:M): se representa con la etiqueta <many-to-many>. Esta etiqueta no se suele utilizar, porque en un esquema relacional las relaciones N:M se descomponen en dos relaciones de uno a muchos con una nueva entidad.

También tenemos la etiqueta <set>, que especifica el tipo de colección que se utilizará para almacenar los empleados de un departamento. También indica cómo se puede obtener en la base de datos (consultando la tabla *emp* y la columna *numdep*).

2.2. Sesiones y estados de los objetos persistentes

Una clase para la que se establecen correspondencias mediante Hibernate se denomina "clase persistente", y una instancia de una clase persistente es un "objeto persistente".

La sesión (*Session*) es un componente esencial en Hibernate. Una sesión, junto con un gestor de entidades asociado, constituye un "contexto de persistencia". Un gestor de entidades (*javax.persistence.EntityManager)* lleva el control de los cambios que se realizan sobre objetos persistentes. Se puede obtener un *EntityManager* a partir de una *Session* y viceversa. Casi todo lo que se puede hacer una de estas API se puede hacer utilizando la otra, así que por simplicidad utilizaremos solo la interfaz *Session* de la API de Hibernate.

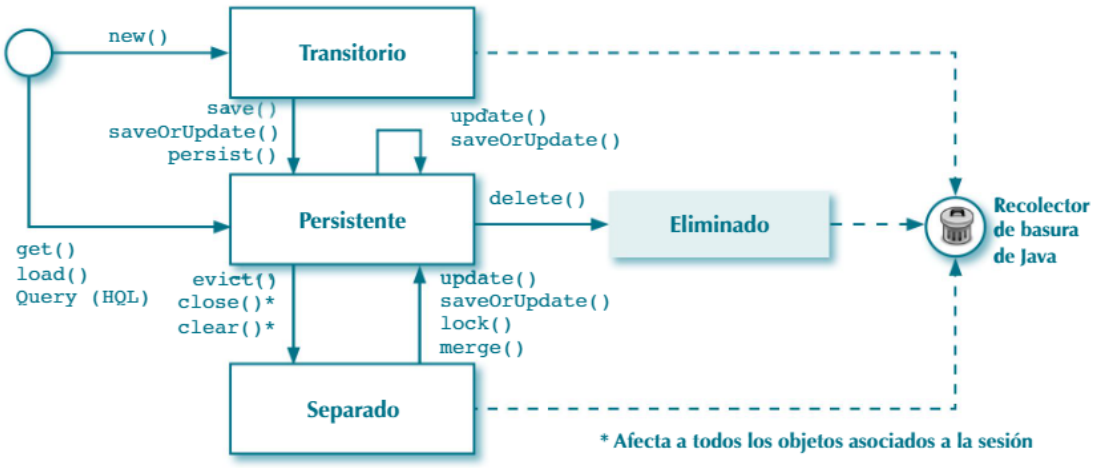
Los cambios que se realizan sobre objetos persistentes se reflejan en la base de datos. Los objetos persistentes pueden estar en diferentes estados:

1. Transitorio (*transient*): el objeto se acaba de crear y no está asociado con ningún contexto de persistencia. No está guardado en la base de datos.

2. Persistente (*persistent*): el objeto tiene un identificador asociado y está guardado en la base de datos. Cualquier cambio que se realice sobre el objeto se reflejará en la base de datos una vez que se cierre la sesión.
3. Separado (*detached*): el objeto sigue teniendo un identificador asociado, pero ya no está asociado con una sesión, bien porque se cerró la sesión o porque se desvinculó de ella. Mientras un objeto está separado, podemos hacer cambios sobre él pero éstos no se reflectirán en la base de datos. Para que se muestren los cambios, tenemos que convertirlo de nuevo en persistente.
4. Eliminado (*removed*): el objeto tiene un identificador y está asociado a un contexto, pero está pendiente de ser borrado de la base de datos.

Un objeto puede cambiar de estado en función de las operaciones que se realicen sobre él dentro de una sesión. Los objetos en estado transitorio y separado, una vez que ya no están asociados a un contexto de persistencia, pueden ser eliminados por el *garbage collector* de Java desde el momento en que no exista ninguna referencia a ellos.

Un diagrama del flujo de estados de un objeto es el siguiente:



2.3. Operaciones con objetos persistentes

Para almacenar un objeto persistente podemos utilizar los siguientes métodos de la interfaz *Session* (https://docs.jboss.org/hibernate/orm/5.6/javadocs/org/hibernate/Session.html):

- **save(objeto)**: almacena el objeto en la base de datos y devuelve el identificador. Si ya existe un objeto con el mismo identificador generará una excepción.
- **saveOrUpdate(objeto)**: hace lo mismo, pero si el objeto ya existe lo actualiza.

Para obtener un objeto:

- **get(clase, identificador)**: devuelve el objeto de esa clase y con ese identificador. Si no existe devuelve null.

- **load(clase, identificador)**: hace lo mismo, pero si el objeto no existe lanza una excepción de tipo ObjectNotFoundException. Se suele utilizar solo si tenemos seguridad de que el objeto existe.

Para borrar un objeto:

- **delete(objeto)**: borra el objeto de la base de datos. No devuelve ningún valor.

Para actualizar un objeto:

- **update(objeto)**: actualiza los datos del objeto. No devuelve ningún valor.

EJERCICIOS PROPUESTOS

1. Crea una clase *AñadirEmpleado*, en la que añadas un nuevo empleado en el departamento 10 (creado en el ejemplo anterior). El nuevo empleado tendrá los siguientes datos:
    - Número: 9999
    - Nombre: ANTONIO
    - Puesto: INVESTIGADOR
    - Sueldo: 3500€

2. Crea una clase *VisualizarDepartamento,* en la que visualices los datos del departamento 3 y el número y nombre de sus empleados.

3. Crea una clase *ActualizarDepartamento*, en la que actualices el departamento 10, de forma que ahora el empleado 9999 sea su jefe. Además, debes borrar el departamento 4.