Diseño y Desarrollo de Sistemas de Información

Mapeo Objeto Relacional (JPA - Hibernate)

[Parte 2/2]

Mapeo de relaciones "uno a muchos"

- En general, en Hibernate las relaciones se definen de forma bidireccional

- En una relación "uno a muchos", un objeto de clase A tendrá un conjunto de objetos de clase B, y un objeto de clase B tendrá un campo que referencie a un objeto de clase A

Monitor.java

```
OneToMany (mappedBy = "monitorresponsable")
private Set<Actividad> actividadesResponsable = new HashSet<Actividad>();
```

Actividad.java

```
@JoinColumn (name = "MONITORRESPONSABLE", referencedColumnName = "CODMONITOR")
@ManyToOne
private Monitor monitorresponsable;
```

Mapeo de relaciones "uno a muchos"

```
Session sesion = HibernateUtil.getSessionFactory().openSession();
Transaction transaction = sesion.beginTransaction();

Monitor monitor = sesion.get(Monitor.class, "M010");
if (monitor == null) {
    System.out.println("Monitor no existe en la BD");
} else {
    Actividad actividad = new Actividad("AC99", "Nueva Actividad");
    actividad.setMonitorresponsable(monitor);
    sesion.save(actividad);
}
transaction.commit();
```

Creación de una nueva actividad asignándole un monitor responsable

```
Session sesion = HibernateUtil.getSessionFactory().openSession();
Transaction transaction = sesion.beginTransaction();

Monitor monitor = sesion.get(Monitor.class, "M003");
if (monitor == null) {
    System.out.println("Monitor no existe en la BD");
} else {
     Actividad actividad = sesion.get(Actividad.class, "AC01");
     actividad.setMonitorresponsable(monitor);
     sesion.save(actividad);
}
transaction.commit();
```

Actualización del responsable de una actividad

Mapeo de relaciones "uno a muchos"

Listado del nombre de las actividades junto con el nombre y correo electrónico del monitor responsable

```
Session sesion = HibernateUtil.getSessionFactory().openSession();
Transaction transaction = sesion.beginTransaction();

Query consulta = sesion.createNativeQuery("SELECT * FROM ACTIVIDAD", Actividad.class);
List<Actividad> actividades = consulta.list();
for (Actividad actividad : actividades) {
    System.out.println(actividad.getNombre() + "\t"
        + actividad.getMonitorresponsable().getNombre() + "\t"
        + actividad.getMonitorresponsable().getCorreo());
    }
transaction.commit();
```

Listado del nombre de los monitores y las actividades de las que son responsables

```
Session sesion = HibernateUtil.getSessionFactory().openSession();
Transaction transaction = sesion.beginTransaction();

Query consulta = sesion.createNativeQuery("SELECT * FROM MONITOR", Monitor.class);
List<Monitor> monitores = consulta.list();
for (Monitor monitor : monitores) {
   System.out.println(monitor.getNombre());
   Set<Actividad> actividades = monitor.getActividadesResponsable();
   for (Actividad actividad : actividades ) {
        System.out.println(actividad.getNombre());
         }
    }
transaction.commit();
```

Mapeo de relaciones "muchos a muchos"

- En una relación "muchos a muchos", un objeto de clase A tendrá un conjunto de objetos de clase B y viceversa
- Al persistir cualquier objeto también se persiste su conjunto de objetos

Socio.java

```
@ManyToMany (mappedBy = "socios", cascade = CascadeType.PERSIST, fetch = FetchType.EAGER)
  private Set<Actividad> actividades = new HashSet<Actividad>();
```

Actividad.java

```
@JoinTable (name = "REALIZA", joinColumns = {
        @JoinColumn (name = "IDACTIVIDAD", referencedColumnName = "IDACTIVIDAD")}, inverseJoinColumns = {
        @JoinColumn (name = "NUMEROSOCIO", referencedColumnName = "NUMEROSOCIO")})
@ManyToMany (cascade = CascadeType.PERSIST, fetch = FetchType.EAGER)
private Set<Socio> socios = new HashSet<Socio>();
```

Mapeo de relaciones "muchos a muchos"

- En este tipo de mapeo es necesario mantener la consistencia en ambas direcciones
- Para ello, una buena práctica es implementar las 2 operaciones (inserción, borrado y actualización) en una única operación

```
Session sesion = HibernateUtil.getSessionFactory().openSession();
Transaction transaction = sesion.beginTransaction();

Socio socioNuevo = new Socio("S999", "Nuevo Socio", "11222333F","A");
Actividad actividad = sesion.get(Actividad.class, "AC01");
actividad.addSocio(socioNuevo);
sesion.save(socioNuevo);   // es necesario porque es un nuevo socio en la BD
sesion.save(actividad);
transaction.commit();
```

Crea un nuevo socio, le asigna la actividad "AC01" y lo inserta en la BD

Método de la clase **Actividad**

```
public void addSocio(Socio socio) {
      socios.add(socio);
      socio.getActividades().add(this);
}
```

Al ejecutarse este código se insertará una tupla en la tabla SOCIO con los datos del nuevo socio y la tupla ("S999", "AC01") en la tabla REALIZA

Mapeo de relaciones "muchos a muchos"

```
Session sesion = HibernateUtil.getSessionFactory().openSession();
Transaction transaction = sesion.beginTransaction();

Socio socio = sesion.get(Socio.class, "S999");
Actividad actividad = sesion.get(Actividad.class, "AC03");

actividad.addSocio(socio);

// una vez asignadas las relaciones en los dos sentidos, se puede realizar
// una operación save() de cualquiera de los dos objetos para que
// se almacene la tupla en la tabla intermedia

sesion.save(actividad);     // también podríamos haber usado sesion.save(socio)
transaction.commit();
```

Asigna una actividad a un socio

Mapeo de relaciones "muchos a muchos"

```
Session sesion = HibernateUtil.getSessionFactory().openSession();
Transaction transaction = sesion.beginTransaction();

Socio socio = sesion.get(Socio.class, "S999");
Actividad actividad = sesion.get(Actividad.class, "AC03");
actividad.eliminaSocio(socio);
sesion.save(actividad);
transaction.commit();
```

Elimina una actividad de un socio y, de camino, un socio de una actividad

Al ejecutarse este código se eliminará la tupla ("S999", "AC03") de la tabla REALIZA

Método de la clase **Actividad**

```
public void eliminaSocio(Socio socio) {
        socios.remove(socio);
        socio.getActividades().remove(this);
}
```

Mapeo de relaciones "muchos a muchos"

Listado del nombre de los socios junto con el nombre de las actividades que realiza

```
Session sesion = HibernateUtil.getSessionFactory().openSession();
Transaction transaction = sesion.beginTransaction();

Query consulta = sesion.createNativeQuery("SELECT * FROM SOCIO S", Socio.class);
List<Socio> socios = consulta.list();
for (Socio socio : socios) {
   System.out.println(socio.getNombre());
   Set<Actividad> actividades = socio.getActividades();
   for (Actividad actividad : actividades ) {
        System.out.println(actividad.getNombre());
     }
 }
transaction.commit();
```