Ejercicios de Mapeado de clases persistentes

Índice

1 Ejercicio 1: Creación y mapeado de una tabla....2
2 Ejercicio 2: Cambiando el estado de los objetos....2
3 Ejercicio 3: Tablas y colecciones....3
4 Ejercicio OPCIONAL....3

Se os proporciona el fichero **hib-sesion2.doc** como plantilla para que contestéis a las preguntas formuladas en los ejercicios.

Vamos a utilizar el proyecto `hib-sesion2`, que se encuentra en el fichero **sesion02-ejercicios.zip**. En dicho proyecto realizaremos las modificaciones que se piden en los ejercicios propuestos.

1. Ejercicio 1: Creación y mapeado de una tabla.

Crea el fichero de mapeado `Event.hbm.xml`, a partir de la clase *Event*. La nueva tabla de BD se llamará `EVENTS`, con los campos: `EVENT_ID`, `EVENT_DATE`, `EVENT_NAME` y `EVENT_SITE`. Para probar el mapeado realizado añade el código necesario para crear y almacenar un evento en el método *public void createAndStoreEvent(String theName, Date theDate, String lugar)* (y asócialo al botón `STORE`, en el fichero *InterfaceUsuario.java*), de forma que se active cuando el usuario introduzca los datos por pantalla en la interfaz de usuario. Pulsando el botón `LIST` se muestra en pantalla un listado de los eventos almacenados.

El atributo *date* puedes asociarlo con el tipo Hibernate *timestamp*.

2. Ejercicio 2: Cambiando el estado de los objetos

A partir del código resultante del ejercicio anterior, se pide realizar las modificaciones necesarias para:

- Añadir el método *public void createInitialEvents()*, que crear filas en la tabla *EVENTS* con los siguientes eventos: ("Congreso","10/10/07","Barcelona"), ("Vacaciones","20/10/07","París"), ("Cumple","14/10/07", "París"), ("Aniversario","07/10/07", "Barcelona"), y ("Cena","28/10/07", "Barcelona"). Ten en cuenta que **cada vez que iniciemos el programa** tenemos que incializar la tabla de datos con los eventos anteriores. Puedes utilizar el método privado *Date obtenerFecha(String fecha)* que se proporciona.
- Añadir el método *public void findEventsBySite(String lugar)* que muestra por pantalla todos los eventos que se celebren en la ciudad que se pasa por parámetro. Por ejemplo, si realizamos la llamada *findEventsBySite("Barcelona")* el resultado será:

  Eventos que tienen lugar en Barcelona:

  ID: 1 Evento: Congreso Fecha: 10/10/2007

  ID: 4 Evento: Aniversario Fecha: 07/10/2007

ID: 5 Evento: Cena Fecha: 28/10/2007

- Añadir el método *public void changeEventsSite(String oldSitio, String nuevoSitio)* que cambia en la base de datos el lugar de celebración de todos los eventos desde *oldSitio* a *nuevoSitio*. Prueba a cambiar los eventos de Barcelona a Alicante, y luego vuélvelos a cambiar a Barcelona. Nota: Para comparar dos elementos utiliza el método `equals` en vez de `==`.
- Añadir el método *public Event makeEventTransient(Long eventId)* que borra en la base de datos el evento cuyo identificador de clave primaria es *eventId*. Para que el usuario pueda cambiar de idea, añadir también el método *public void undoMakeEventTransient(Event theEvent)* que vuelve a insertar en la BD el evento que se pasa por parámetro (y que será el que acabamos de borrar).

3. Ejercicio 3: Tablas y colecciones

A partir de la clase java *Person* crea el fichero de mapeado *Person.hbm.xml*. La nueva tabla tendrá el nombre `PERSON`. Las propiedades de la clase *Person* se *mapearán* como `P_EDAD`, `P_NOMBRE`, `P_APELLIDOS`, y `P_TELS`. Utiliza el nombre de `TELEFONOS_PERSONA` para la tabla que almacena la colección de teléfonos. Utiliza el valor del atributo `lazy` a `false`. Para probar el fichero de mapeado crea un fichero *PersonManager.java* (al igual que hicimos con *EventManager.java*), y crea los métodos:

- *public void createAndStorePerson(int edad, String nombre, String apellidos, Set telefonos)*: para crear y almacenar una nueva persona en la base de datos
- *public void createInitialPersons()*. Crea filas inciales en la tabla *PERSON* (cada vez que ejecutamos el programa) con los siguientes datos: (10,"Pepa","Flores", "1111","2222"),(20,"Maria","Flores", "1111","2222"), (45,"Juan","Flores", "1111","2222"), (30,"Pepe","Flores", "1111","2222"), (56,"Marta","Flores", "1111","2222").
- *private List listPersons()*: obtiene una lista de las personas almacenadas en la base de datos.
- *public void printPersons()*: imprime en pantalla **todos** los datos de las personas almacenadas en la base de datos con la información que devuelve el método *listPersons*.
- *public void addTelefonosToPerson(Long personId, String tel1, String tel2)*: añade dos teléfonos nuevos a una persona cuyo identificador en la base de datos es *personId*.

4. Ejercicio OPCIONAL

Una vez implementado en el ejercicio anterior los métodos *listPersons* y *printPersons*, prueba a poner el atributo `lazy` en el fichero de mapeado con valor `true` (quitando el

atributo se consigue el mismo efecto, ya que éste es su valor por defecto). ¿Puedes explicar lo que ocurre al ejecutar de nuevo el programa?. Realiza los cambios necesarios en *PersonManager.java* para que el programa funcione correctamente con el atributo `lazy` en su opción por defecto.