Calidad de los Sistemas Informáticos, 2018-2019
Práctica 2 – Clase de mapeo (1)

Objetivo

El objetivo de esta práctica es la creación de una clase de mapeo que represente la entidad construida como tabla en la práctica 1. En concreto, se creará la clase, el constructor, un método de creación / inserción de objeto / registro, y los métodos auxiliares necesarios.

El procedimiento para seguir será el siguiente:

1. Implementación de la clase, métodos de acceso y constructor (pág. 2)
2. Implementación métodos auxiliares (pág. 2)
3. Implementación del método de creación / inserción (pág. 2)
4. Deshabilitar pruebas innecesarias y que afectan al rendimiento (pág. 3)
5. Pruebas de construcción (pág. 3)
6. Pruebas de inserción (pág. 3)

Requisitos previos

Para la realización de esta práctica se requieren los resultados de la práctica 1.

Criterios sintácticos

- `Courier`: código fuente, nombres de archivos, paquetes, entidades, atributos, tablas y campos.
- *`Cursiva`*: términos en otro idioma.
- Negrita: contenido resaltado.
- **$**Entre dólar**$**: contenido no textual, generalmente a decidir por el desarrollador.

Instrucciones previas

- Salvo los métodos de tipo `get` y `set` (ver punto 2), todos los métodos realizados se comentarán, incluyendo descripción y excepciones. Para ello, en Eclipse se pulsará `shift+alt+J` sobre la cabecera del método.
- En aquellas partes del código en la que se consideren adecuadas precondiciones (generalmente, al inicio de cada método), deberá indicarse el comentario.

```
// TODO: Preconditions
```

Pablo de la Torre Moreno
Departamento de Ingeniería Informática
25 de septiembre de 2018

Procedimiento

1. Implementación de la clase, métodos de acceso y constructor

1. Crear la clase **$**Entidad**$**, dentro del (sub)paquete data.
2. Crear tantas variables privadas y métodos de acceso (get**$**Campo**$**() y set**$**Campo**$**(**$**tipo**$** **$**variable**$**)) como campos tenga la tabla **$**Entidad**$**.
3. Implementar un constructor para dicha clase y que contenga un parámetro **$**tipo**$** **$**clavePrimaria**$** (por ejemplo, String sDni o int iId). Dicho constructor deberá obtener el registro correspondiente a dicha clave primaria y rellenar las variables privadas.

2. Implementación métodos auxiliares

1. Sobreescribir en la clase **$**Entidad**$** el método String toString(), para devolver una cadena con información significativa. Se aconseja que el primer valor de la cadena sea la invocación a super.toString() y que, mediante separación por dos puntos, incluya algunos campos cortos de la tabla. Por ejemplo:

```
User@74341960:0000001A:Test User 1:true:2007-03-27 08:24:33
```

2. En el caso de que el identificador de la tabla sea autonumérico, se debe implementar un método estático en la clase Data llamado LastId(Connection con), que devuelve el último identificador insertado[1]. Para facilitar la portabilidad a otras bases de datos, la instrucción SQL por la cual se obtiene dicho identificador se incluirá en el archivo db.properties, con el nombre de variable jdbc.lastIdSentence. Su valor para MySQL es

```
SELECT LAST_INSERT_ID()
```

3. Implementación del método de creación / inserción

1. Implementar en la clase $Entidad$ un método estático public static $Entidad$ Create(**$**parámetros**$**) para la misma clase que reciba como parámetros toda la información requerida (valor de los campos que no se generan automáticamente) para crear un nuevo registro[2] en la tabla **$**Entidad**$**, devolviendo una instancia de dicha clase invocando a su constructor[3]. No debe olvidarse hacer pasar por Data.String2Sql toda cadena a enviar a la base de datos -aquí y en adelante-, con los parámetros correspondientes.

---

[1] El método LastId no debe cerrar la conexión, dado que ésta viene de otro método, que deberá de ser el encargado de manejarla. Si se cierra, estamos suponiendo que el método que invoca no va a seguir utilizándola, lo cual es una suposición peligrosa y un ejemplo de acoplamiento por control descendente. En general, un método sólo debe (y ha de hacerlo) destruir o inhabilitar las instancias que él crea.

[2] Se utiliza el método executeUpdate, en vez de executeQuery.

[3] En casos en los que el identificador sea autonumérico, es necesario invocar al método LastId(Connecion con) que se implementó en el punto 5.

Pruebas

4. Deshabilitar pruebas innecesarias y que afectan al rendimiento

1. Colocar la instrucción `@Disabled` o `@Ignore` (según versión de JUnit, ambas del paquete `org.junit`) sobre `@Test`, en el método `testTableAccess` de la clase `DataTest.java`. De lo contrario, cuando la base de datos contenga muchos registros su ejecución afectará al rendimiento.

5. Pruebas de construcción

1. Crear la clase de JUnit `$Entidad$Test.java` en el paquete `test`. Dicha clase debe incluir la invocación al método `Data.LoadDriver()` dentro del método `setUpBeforeClass`.
2. Renombrar el método `test` creado por defecto, denominándolo `testConstructor`, cuya función será obtener una instancia de la clase mediante la llamada a su constructor y comprobar que cada propiedad `get` se corresponde a su columna respectiva en la base de datos, incluyendo el campo que hace de clave primaria (para lo cual será necesario traer también el registro de la base de datos[4]). La instancia a probar queda a criterio del desarrollador. Se requiere un `Assert` para cada propiedad (método `get`).
3. Ejecutar como cobertura y modificar hasta su correcto funcionamiento.

6. Pruebas de inserción

1. Implementar en la misma clase otro método test:

   ```
   @Test
   public void testCreate()
   ```

   que cree un registro y compruebe que cada propiedad de la instancia devuelta se corresponde a la información original.
2. Ejecutar como cobertura y modificar hasta su correcto funcionamiento[5].

[4] No debe usarse SELECT *, sino indicarse uno a uno los campos a obtener.
[5] Al ejecutar una segunda vez con los mismos datos, debe de dar error. Leer la información generada por la excepción. Si ésta no aparece, consultar con el docente.