Calidad de los Sistemas Informáticos, 2018-2019
Práctica 1 – Conexión a la entidad

Objetivo

El objetivo de esta práctica es adquirir experiencia en la construcción de tablas en un sistema de gestión de bases de datos a partir de un análisis de requisitos, así como el uso y comprensión del código facilitado para la conexión y obtención de los datos de dicha tabla. De igual modo, se aprovecha la ocasión para generar un proyecto de Java.

El procedimiento para seguir será el siguiente:

1. **Selección de entidad a implementar** (p. 2), donde se elegirá una entidad de entre todas las obtenidas de los requisitos de información del análisis.
2. **Creación de la tabla correspondiente** (p. 2) en la base de datos.
3. **Creación básica en Eclipse del proyecto de la práctica** (p. 2).
4. **Creación de prueba unitaria para la clase `Data`** (p. 3), donde se desarrollarán varias técnicas para acceder a diferentes datos, validándose cada una mediante el método `assertEquals`.
5. **Implementación de métodos auxiliares para de conversión a BD** (p. 4), donde se implementarán los métodos de conversión de cadenas y de booleanos, que deberán ser invocados siempre que se trate de enviar cualquiera de los dos tipos de datos a la base de datos.
6. **Creación de pruebas unitarias para los métodos de conversión** (p. 4), validando éstos.

Requisitos previos

Para la realización de esta práctica se requiere:

- Eclipse con biblioteca de conexión al gestor de bases de datos a utilizar.
- Sistema de Gestión de Bases de Datos.
- Análisis de requisitos y, en concreto, una relación de los requisitos de información.
- Seminario de buenas prácticas de programación, cuyo seguimiento y cumplimiento se tendrá en cuenta a la hora de la evaluación de la actividad.

Criterios sintácticos de este documento

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

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

Procedimiento

1. Selección de entidad a implementar

1. Elegir de entre todas las entidades de los requisitos de información obtenidos en el análisis de requisitos (ver "Requisitos previos") una entidad que cumpla las siguientes características.
    a. Sea una de las entidades principales del sistema.
    b. Sea una entidad fuerte.
    c. Sea tipificable, es decir, que contenga un atributo denominado `tipo` (modelo conceptual) y/o relación con una entidad débil que lo tipifique (modelo lógico).
    d. Contenga al menos un atributo de tipo texto y uno de tipo numérico. No es necesario trabajar más que con dichos dos atributos. El resto pueden obviarse.
2. De no existir dicha entidad (que de ahora en adelante se denominará **$**`Entidad`**$**, en este documento), concebir una similar a partir de una ya existente, con objeto de ajustarse lo más posible a los pasos del proyecto.

2. Creación de la tabla correspondiente a la entidad

1. Crear, mediante el correspondiente editor de bases de datos:
    a. La base de datos **$**`Equipo`**$**, en función del nombre del equipo que desarrolla la práctica, evitando, por motivos de compatibilidad, caracteres especiales y sustituyéndolos por símbolos del alfabeto occidental básico. El cotejamiento por defecto será `utf8_general_ci` o similar[1].
    b. La tabla **$**`Entidad`**$** y su estructura, evitando incluir un número excesivo de campos y para flexibilizar los tiempos de la práctica[2], y con al menos los campos de tipo texto y número referidos en el punto 2. No debe incluirse por ahora el atributo de tipo referido en el punto 1.1.c.
2. Incluir un par de registros de ejemplo en la tabla, y que constituirán la información que se utilizará en esta práctica.
3. Exportar la base de datos a un archivo **$**`Equipo`**$**`.sql`.

3. Creación del proyecto de la práctica

1. Crear en Eclipse un proyecto Java llamado `es.uca.gii.csi18.`**$**`equipo`**$**, y que será el proyecto a desarrollar en la práctica[3].
    a. Durante la creación, incluir como biblioteca del proyecto el conector MySQL para Java, creado en la práctica anterior ("Instalación y configuración").
2. Arrastrar al nombre del proyecto el archivo resultado de la exportación de la base de datos generado en el punto 2.3 (cuando Eclipse pregunte, se indicará "como copia"). Dicho archivo deberá actualizarse en cada versión de la aplicación / base de datos a

[1] "`ci`" significa *case insensitive*. Se aconseja utilizar siempre este tipo de cotejamiento salvo que se quiera que una búsqueda como "pedro" no encuentre el valor "Pedro", al tenerse en cuenta las mayúsculas.
[2] Posteriormente podrán incluirse cuantos se deseen.
[3] Por convención, el nombre de los paquetes de Java no debe contener ninguna mayúscula.

entregar en la práctica, en su caso. El objetivo es tener la estructura de la base de datos dentro del proyecto, junto al código, por si es necesario eliminarla y crearla de nuevo, o continuar el desarrollo en otra máquina.

3. Crear bajo el nombre del proyecto o arrastrar al mismo (también como copia) el archivo `db.properties`, facilitado en el anexo. Modificar a continuación la información requerida de este archivo para el correcto acceso a la base de datos.
4. Con el botón derecho, crear en el paquete principal (`src`) un primer (sub)paquete `es.uca.gii.csi18.$equipo$.util`, y crear o añadir en él el archivo `Config.java`. Modificar la línea `package` de dicho archivo para adaptarlo al nombre del paquete recién creado.
5. Hacer lo mismo con un (sub)paquete `es.uca.gii.csi18.$equipo$.data`, y crear o añadir en él el archivo `Data.java`. Modificar la línea `package` y la línea correspondiente del bloque `import` de dicho archivo para adaptarlo al nombre del paquete recién creado.

4. Creación de prueba unitaria para la clase Data

1. Usando como base el código de la práctica anterior ("Instalación y configuración")[4], crear el (sub)paquete para pruebas `es.uca.gii.csi18.$equipo$.test`. Dicho paquete se empleará para ubicar las clases de pruebas unitarias y de integración.
2. Con el botón derecho sobre el paquete mencionado (*New > JUnit Test Case*), crear la clase de JUnit `DataTest.java`. Dicha clase debe incluir la invocación al método `Data.LoadDriver()` dentro del método `setUpBeforeClass`.
3. Renombrar el método `test` creado por defecto, denominándolo `testTableAccess`, cuya función será traer todos los registros de la tabla `$Entidad$` mediante una consulta y comprobar que el número de éstos se corresponde al de una consulta `SELECT COUNT` independiente previa. Asimismo e ilustrativamente, se deberá mostrar por pantalla los valores de cada registro[5], separados por espacios y cada registro en una línea. Los pasos son:
    a. Recuperar el número de elementos de una tabla mediante la consulta correspondiente y validar que el valor obtenido es lo que se espera, mediante el método de aserción correspondiente `assertEquals`.
    b. Recuperar todos los registros de una tabla mediante `SELECT *` (excepcionalmente; a partir de ahora y como norma, no se deberá emplear el asterisco, sino explícitamente todos los nombres de los campos que se deseen obtener). Recorrer el `ResultSet` para imprimir cada registro (todos sus campos en una línea) y aumentar un contador. Validar que el que el resultado del punto a) se corresponde al valor final del contador.
    c. El número de campos de un registro se obtiene con el método `getMetaData().getColumnCount()` del objeto de tipo `ResultSet`. Validar que el número de campos es el que se espera.
4. Ejecutar como cobertura con EclEmma y modificar hasta su correcto funcionamiento.

[4] Partiendo de la hipótesis varias veces constatada de que "los dedos tienen memoria" y que una revisión línea a línea se entiende mejor, se aconseja no copiar y pegar código, sino escribirlo.

[5] El método `getString` de la clase `ResultSet` admite cualquier tipo de dato básico, si no se desea otra cosa que imprimir el valor devuelto sin procesarlo.

5. Implementación de métodos auxiliares para de conversión a BD

1. Implementar en la clase `Data` los métodos apropiados para convertir un tipo de dato de Java en un tipo de datos SQL válido. Dichos métodos deberán invocarse siempre que se desee enviar dicho tipo de dato a la base de datos. Los más utilizados son:
    a. Transformación de cadena, que convierte las comillas simples del parámetro s en dos comillas simples (ojo, no una comilla doble, sino dos simples), para evitar que SQL la interprete como cierre de cadena. Por otro lado, se puede indicar añadir automáticamente comillas simples en los extremos de la cadena, para ahorrar la concatenación de aquéllas en la generación de la consulta. Finalmente, se permite añadir tantos por cientos (el comodín de SQL) en los extremos (siempre dentro de las comillas simples, en su caso), para generar cadenas de búsqueda.

```
public static String String2Sql(
      String s, boolean bAddQuotes, boolean bAddWildcards)
```

Ejemplos (se indica tras la flecha el contenido de la cadena que devuelve):

```
Data.String2Sql("hola", false, false) → hola
Data.String2Sql("hola", true, false) → 'hola'
Data.String2Sql("hola", false, true) → %hola%
Data.String2Sql("hola", true, true) → '%hola%'
Data.String2Sql("O'Connell", false, false) → O''Connell
Data.String2Sql("O'Connell", true, false) → 'O''Connell'
Data.String2Sql("'Smith '", false, true) → %''Smith ''%
Data.String2Sql("'Smith '", true, false) → '''Smith '''
Data.String2Sql("'Smith '", true, true) → '%''Smith ''%'
```

    b. Transformación de booleano, devolviendo 0 ó 1 en función de si el parámetro es `false` o `true`, respectivamente.

```
public static int Boolean2Sql(boolean b)
```

6. Creación de pruebas unitarias para los métodos de conversión

1. Incluir en la clase `DataTest` los métodos `testString2Sql` y `testBoolean2Sql`, respectivamente para los dos anteriores. Incluir en el primero aserciones para comprobar todos los ejemplos facilitados y, en el segundo, dos aserciones de prueba para los dos valores booleanos.
2. Ejecutar como cobertura con EclEmma y modificar hasta su correcto funcionamiento.

Anexo A: Código fuente

Se incluye a continuación el código fuente de los archivos facilitados. Lo que aparece entre **$** se refiere a información personalizada por el desarrollador, excluyendo dicho carácter. Por ejemplo, **$**`contraseña`**$** se sustituiría por `12345`, en su caso.

db.properties

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost/$equipo$
jdbc.username=$usuario MySQL$
jdbc.password=$contraseña$
```

Config.java

```
package es.uca.gii.csi18.$equipo$.util;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

public class Config {

	public static Properties Properties(String sFile)
		throws IOException {

		InputStream inputStream = null;

		try {

			inputStream = new FileInputStream(sFile);
			Properties result = new Properties();
			result.load(inputStream);
			return result;

		}

		finally { if (inputStream != null) inputStream.close(); }
	}
}
```

Data.java

```
package es.uca.gii.csi18.$equipo$.data;

import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;

import es.uca.gii.csi18.$equipo$.util.Config;

public class Data {

	public static String getPropertiesUrl() {
		return "./src/db.properties";

	}

	public static Connection Connection() throws Exception {

		try {

			Properties properties =
				Config.Properties(getPropertiesUrl());

			return DriverManager.getConnection(
				properties.getProperty("jdbc.url"),
				properties.getProperty("jdbc.username"),
				properties.getProperty("jdbc.password"));

		}
		catch (Exception ee) { throw ee; }
	}


	public static void LoadDriver() throws
		InstantiationException, IllegalAccessException,
		ClassNotFoundException, IOException {

		Class.forName(Config.Properties(Data.getPropertiesUrl()
			).getProperty("jdbc.driverClassName")).newInstance();
	}
}
```