

Informe de práctica 02

Tema: Patron Singleton y Bases de Datos

Estudiante	Escuela	Asignatura
Jhonatan David Arias Quispe Jorge Luis Mamani Huarsaya jariasq@unsa.edu.pe, jmamanihuars@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de Programación 2 Semestre: II Código: 1701213

1 Actividades

1. Primero haremos conexión con MariaDB localmente.
2. Crearemos una base de datos fp2-23b que contendrá las tablas y datos proporcionados en clase.

2 Equipos, materiales y temas utilizados

- Sistema Operativo ArchCraft GNU Linux 64 bits Kernel
- Sistema Operativo Arch GNU Linux 64 bits Kernel
- NeoVim
- OpenJDK 64-Bit 20.0.1
- MariaDB Java Client 3.3.2
- Git 2.43.0
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.

3 URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/JhonatanDczel/fp2-23b.git>
- URL para la práctica 02 en el Repositorio GitHub.
- <https://github.com/JhonatanDczel/fp2-23b/tree/main/fase03/prac02>

4 Instalando el servidor y cliente MariaDB

- En esta ocasión, estamos utilizando un sistema operativo Linux y se ha decidido usar el servidor y cliente MariaDB de Oracle, podemos descargarlo desde los repositorios oficiales del sistema operativo.

Listing 1: Descargando el servidor y el cliente MariaDB

```
1 $ sudo pacman -S mariadb
```

- Después de haber instalado el servidor MariaDB, debemos realizar la activación del servidor para que se ejecute en segundo plano. Comúnmente todo sistema operativo basado en Linux utiliza el gestor de servidores **systemctl**. En Linux muchos lo conocemos como Daemons (demonios).

Listing 2: Descargando el servidor y el cliente MariaDB

```
2 $ sudo systemctl start mariadb.service
```

5 Descargando el driver para Java

- Para poder conectarse a la base de datos usando el lenguaje de programación Java y realizar consultas, tenemos que usar un .jar que actúe como puente para la conexión.
- Podemos instalar el driver JDBC para el servidor MariaDB. Por ejemplo en sistema operativo Arch Linux podemos instalarlo de la siguiente manera.

Listing 3: Descargando el servidor y el cliente MariaDB

```
3 $ sudo paru -S mariadb-jdbc
```

6 Peticiones al servidor MariaDB

- Por lo pronto solo queremos probar el funcionamiento del patrón singleton, de hecho lo que presentaremos es sencillo.

6.1 La clase Petición.java

- Esta clase es la que se encarga de realizar las peticiones al servidor, recibir los resultados e imprimirlos. Prácticamente crear objetos de esta clase es realizar más de una petición al servidor.
- El patrón de diseño **Singleton** nos permite crear solo una instancia de esta clase. Se logra esta funcionalidad haciendo privado su constructor y solo poder crear una instancia de esta clase llamando a un método estático que solo permita crear una instancia.

```
1 private static Petición instance;  
2  
3 private Petición() throws SQLException, ClassNotFoundException{}
```

- El constructor es privado y solo puede ser utilizado por cualquier método de la misma clase. En esta ocasión, siguiendo el patrón Singleton, solo el método `createInstance()` puede crear una instancia del método.

```
1 public static Peticion createInstance() throws SQLException, ClassNotFoundException{
2     if (instance == null) {
3         instance = new Peticion();
4     }
5     getData();
6     return instance;
7 }
```

- El campo instance almacena la única instancia de la clase que puede ser creada, esta instancia es verificada cada vez que se use el método createInstance(). Una vez que se verifica la existencia del objeto, agregamos el método getData() que se encarga de hacer la solicitud correspondiente.

7 Petición a la base de datos desde Java

- Usaremos el driver controlador para Mariadb
- Se inicia la conexión cargando JDBC Driver y creando un objeto Connection:

```
1 String url = "jdbc:mariadb://localhost/fp2_23b";
2 String user = "fp2_23b";
3 String password = "12345678";
4
5 Class.forName("org.mariadb.jdbc.Driver");
6
7 Connection connection = DriverManager.getConnection(url, user, password);
```

- Primero se registra el controlador de la base de datos MariaDB
- Y se establece la conexión a la base de datos usando el método getConnection de la clase DriverManager
- Ahora realizamos la petición con un objeto Statement y el método executeQuery:

```
1
2 Statement statement = connection.createStatement();
3 String query = "SELECT * FROM vets";
4 ResultSet resultSet = statement.executeQuery(query);
5
6 while (resultSet.next()) {
7     int id = resultSet.getInt("id");
8     String fn = resultSet.getString("first_name");
9     String ln = resultSet.getString("last_name");
10
11     System.out.println("ID: " + id + ", First Name: " + fn + " Last Name: " + ln);
12 }
13 System.out.println();
```

- Este código sencillo crea peticiones a la base de datos y muestra los datos por consola
- Posteriormente creamos una clase Peticion que se encargue de manejar las peticiones

8 Haciendo peticiones con múltiples objetos

- Ahora creamos otra clase Test desde la que instanciaremos objetos Peticion que harán las peticiones a la base de datos

- Como es una implementación inicial, instanciaremos un nuevo objeto cada vez que queramos hacer una petición

```
1 public class Test {  
2     public static void main(String[] args) throws SQLException, ClassNotFoundException {  
3         Peticion p0 = new Peticion();  
4         Peticion p1 = new Peticion();  
5         Peticion p2 = new Peticion();  
6         Peticion p3 = new Peticion();  
7  
8         System.out.println(p0.hashCode() + "\n" + p1.hashCode() + "\n" + p2.hashCode() + "\n" +  
9             p3.hashCode());  
10    }  
11 }
```

- Como se puede observar, crean 4 objetos Peticion que hacen las peticiones a la base de datos y muestran los resultados por consola
- Ademas imprimimos el hashCode de cada uno de los objetos
- Viendo el hashCode podemos deducir que todos estos objetos son distintos:

```
1  
2     ...  
3     ...  
4  
5     ID: 4, First Name: Rafael Last Name: Ortega  
6     ID: 5, First Name: Henry Last Name: Stevens  
7     ID: 6, First Name: Sharon Last Name: Jenkins  
8     ID: 7, First Name: Jorge Last Name: god  
9  
10    ID: 1, First Name: James Last Name: Carter  
11    ID: 2, First Name: Helen Last Name: Leary  
12    ID: 3, First Name: Linda Last Name: Douglas  
13    ID: 4, First Name: Rafael Last Name: Ortega  
14    ID: 5, First Name: Henry Last Name: Stevens  
15    ID: 6, First Name: Sharon Last Name: Jenkins  
16    ID: 7, First Name: Jorge Last Name: god  
17  
18    1684106402  
19    1204759365  
20    6952832174  
21    2048675102
```

- Esto, aunque en primer momento parece lo mas facil e intuitivo de hacer, trae complicaciones cuando hacemos peticiones a grande escala
- Si tuvieramos que crear un objeto por cada petición que hagamos y tuvieramos que hacer cientos de miles de peticiones resultaría ineficiente en espacio
- La solucion es usar un patrón de diseño bien conocido: el Singleton

9 Peticiones con un solo objeto: Patron Singleton

- Usando el patron de singleton, al ejemplo anteriormente mostrado, optimiza el espacio usado en peticiones grandes
- Al usar un unico objeto en cada petición, ahorramos el espacio que generaría instanciar un nuevo objeto por cada petición

- Luego de hacer la implementacion del patron singleton, podemos verificar si se está usando un solo objeto, imprimiendo el Hash como se muestra a continuacion:

```
1  ...
2
3  ID: 3, First Name: Linda Last Name: Douglas
4  ID: 4, First Name: Rafael Last Name: Ortega
5  ID: 5, First Name: Henry Last Name: Stevens
6  ID: 6, First Name: Sharon Last Name: Jenkins
7  ID: 7, First Name: Jorge Last Name: god
8
9  1684106402
10 1684106402
11 1684106402
12 1684106402
```

- Como vemos, el hash que se imprime en cada peticion es exactamente el mismo
- Esto nos indica que hicimos correctamente la implementacion de singleton para la clase Peticion