Informe de práctica 0222

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 Programacion 2 Semestre: II Código: 1701213

1. Actividades

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

2. Equipos, materiales y temas utilizados

- Sistema Operativo ArchCraft GNU Linux 64 bits Kernell
- Sistema Operativo Arch GNU Linux 64 bits Kernell
- 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 ocasion, 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. j

Listing 1: Descargando el servidor y el cliente MariaDB

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

- Despues de haber instalado el servidor MariaDB, debemos realizar la activacion del servidor para que se ejecute en segundo plano. Comunmente 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

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

5. Descargando el driver para Java

- Para poder conectarse a la base de datos usando el lenguaje de programacion Java y realizar consultas, tenemos que usar un .jar que actue como puente para la conexion.
- Podemos instalar el driver JDBC par 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

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

6. Peticiones al servidor MariaDB

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

6.0.1. La clase Peticion.java

- Esta clase es la que se encarga de realiza las peticiones la servidor, recibir los resultados e imprimirlos. Praticamente crear objetos de esta clase es realizar mas de una peticion al servidor.
- El patron de diseno **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 clse llamando a un metodo estatico que solo permita crear una instancia.

```
private static Peticion instance;

private Peticion() throws SQLException, ClassNotFoundException{}
```

- El campo instance almacena la unica instancia de la clase que puede ser creada.
- El constructor es privador y solo puede ser utilizado por algun metodo de la misma clase

6.0.2. Variables importantes

```
String url = "jdbc:mariadb://localhost/fp2_23b";
String user = "fp2_23b";
String password = "12345678";
```

- Como se observa, estos son las variables que almacenan los parámetros necesarios para la conexión a la base de datos, como la URL de la base de datos, el nombre de usuario y la contraseña.

7. Peticion a la base de datos desde Java

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

```
String url = "jdbc:mariadb://localhost/fp2_23b";
String user = "fp2_23b";
String password = "12345678";

Class.forName("org.mariadb.jdbc.Driver");

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 metodo getConnection de la clase DriverManager
- Ahora realizamos la peticion con un objeto Statement y el metodo executeQuery:

```

Statement statement = connection.createStatement();
String query = "SELECT * FROM vets";
ResultSet resultSet = statement.executeQuery(query);

while (resultSet.next()) {
  int id = resultSet.getInt("id");
  String fn = resultSet.getString("first_name");
  String ln = resultSet.getString("last_name");

  System.out.println("ID: " + id + ", First Name: " + fn + " Last Name: " + ln);
}
System.out.println();
```

- Este codigo 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 multiples objetos

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

- Como es una implementación inicial, instanciaremos un nuevo objeto cada vez que querramos hacer una peticion

```java
public class Test {
  public static void main(String[] args) throws SQLException, ClassNotFoundException {
      Peticion p0 = new Peticion();
      Peticion p1 = new Peticion();
      Peticion p2 = new Peticion();
      Peticion p3 = new Peticion();

    System.out.println(p0.hashCode() + "\n" + p1.hashCode() + "\n" + p2.hashCode() + "\n" +
        p3.hashCode());
  }
}
```

- 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:

```
...
...

ID: 4, First Name: Rafael Last Name: Ortega
ID: 5, First Name: Henry Last Name: Stevens
ID: 6, First Name: Sharon Last Name: Jenkins
ID: 7, First Name: Jorge Last Name: god

ID: 1, First Name: James Last Name: Carter
ID: 2, First Name: Helen Last Name: Leary
ID: 3, First Name: Linda Last Name: Douglas
ID: 4, First Name: Rafael Last Name: Ortega
ID: 5, First Name: Henry Last Name: Stevens
ID: 6, First Name: Sharon Last Name: Jenkins
ID: 7, First Name: Jorge Last Name: god

1684106402
1204759365
6952832174
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 peticion 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