



Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😊 

➤ Capa de Acceso a Datos

Plan formativo: Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



REPASO CLASE ANTERIOR

En la clase anterior trabajamos :

- ✓ Acceso a datos en una aplicación web dinámica
- ✓ Conexión de una aplicación a una base de datos mediante JDBC



LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

5.4

Start! 🏁

Capa de acceso a
datos

Conexión con Singleton

Singletoneando

Implementando un objeto
singleton para la conexión a la
base de datos
Obtención de datos (objeto
Statement)

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



Implementar Singleton para conectarse a la base de datos



Obtener datos con Statement



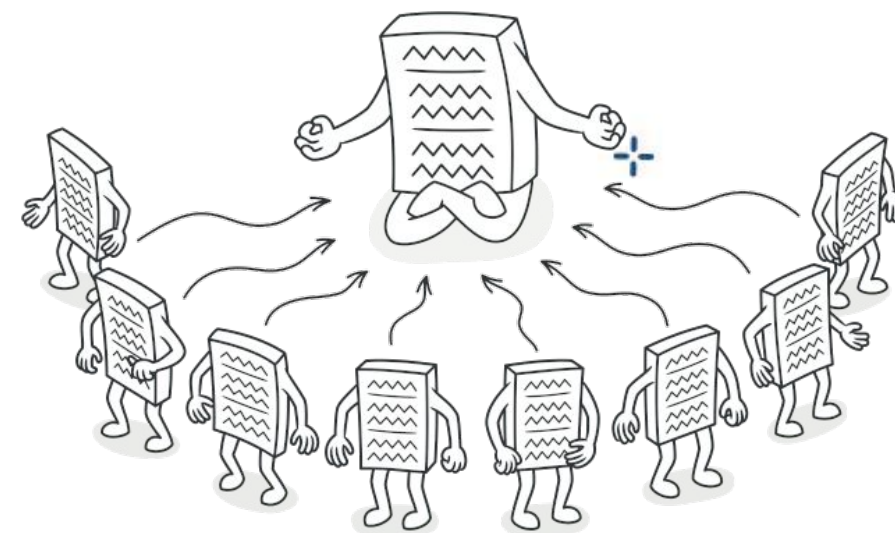
➤ Implementando Singleton para conectarse a la DB



Implementando Singleton

Singleton es un **patrón de diseño creacional** que nos permite asegurarnos de que una clase tenga **una única instancia**, a la vez que proporciona un punto de acceso global a dicha instancia.

Implementar un objeto singleton para la conexión a la base de datos es una práctica común para asegurar que solo exista una instancia de la conexión y reutilizarla en toda la aplicación. Esto puede mejorar el rendimiento y la eficiencia, evitando la creación excesiva de conexiones.






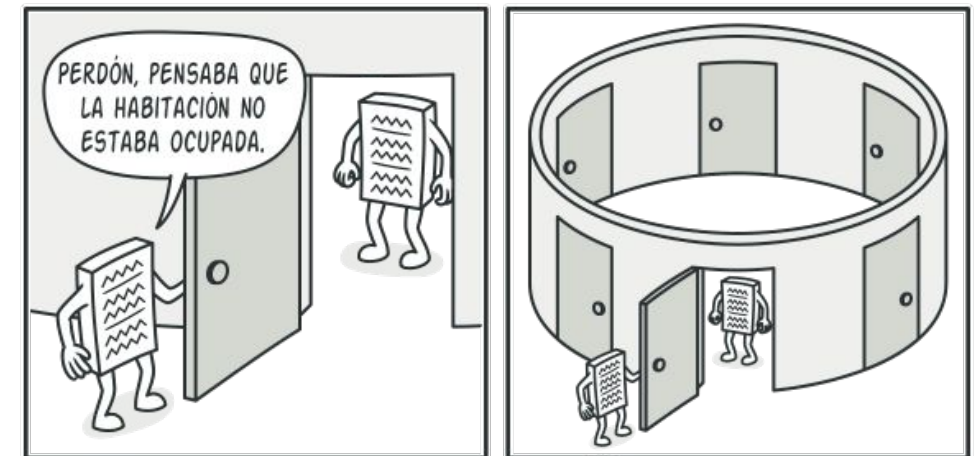
Implementando Singleton

Para comunicarnos con una base de datos utilizando JDBC, se debe en primer lugar establecer una conexión con la base de datos a través del driver JDBC apropiado. El API JDBC especifica la conexión en la interfaz `java.sql.Connection`.

La clase `DriverManager` permite obtener objetos `Connection`.

Para conectarse es necesario proporcionar:

- URL de conexión, que incluye:
 - Nombre del host donde está la base de datos.
 -  Nombre de la base de datos a usar.
- Nombre del usuario en la base de datos.
- Contraseña del usuario en la base de datos.





Implementando Singleton

Como vimos la clase anterior, una vez cargado el driver se debe establecer la conexión con la BD. Para ello se utiliza el siguiente método:

`Connection` conexion = `DriverManager.getConnection`(url, login, password);

El objeto `Connection` representa el contexto de una conexión con la base de datos, es decir:

- Permite obtener **objetos Statement para realizar consultas SQL**.
- Permite obtener metadatos acerca de la base de datos (nombres de tablas, etc.)
- Permite gestionar transacciones.

```
protected void conectarBase(){
    try {
        //registrar el Driver de MySQL
        Class.forName("com.mysql.jdbc.Driver");

        //URL de la DB
        String urlDataBase = "jdbc:mysql://localhost:3306/"+database+"?useSSL=false";

        //Establecemos la conexion!
        Connection conexion = DriverManager.getConnection(urlDataBase, user, password);
    } catch (Exception e) {
        System.out.println("Error en Conectar Base");
    }
}
```



Implementando Singleton

Veamos el ejemplo respetando el patrón Singleton:

Como podemos ver, creamos los objetos como atributos protegidos y las variables finales para que la clase EjemploSingleton tenga todas las instancias de clase necesarias.



```
public class EjemploSingleton {
    //OBJETO CONNECTION ---> Encargado de INICIAR - TENER - MANTENER la conexion
    protected Connection conexion = null;
    //OBJETO RESULTSET ---> Guardar todos los datos que llegan de la DB (las filas de la consulta)
    protected ResultSet resultado = null;
    //OBJETO STATEMENT ---> "Tiene" las consultas... es donde generamos las sentencias a ejecutar!
    protected Statement stmt = null;

    private final String user = "root";
    private final String password = "root";
    private final String database = "nombre_database";

    protected void conectarBase(){
        try {
            //registrar el Driver de MySQL
            Class.forName("com.mysql.jdbc.Driver");

            //URL de la DB
            String urlDataBase = "jdbc:mysql://localhost:3306/"+database+"?useSSL=false";

            //Establecemos la conexion!
            conexion = DriverManager.getConnection(urlDataBase, user, password);

        } catch (Exception e) {
            System.out.println("Error en Conectar Base");
        }
    }
}
```



Implementando Singleton

De manera consecutiva a la creación de una conexión, **es necesario desconectar la base**. Para esto, vamos a crear un método que verifique si hay alguna transacción abierta y en ese caso que la cierre.



```
protected void desconectarBase(){
    try {
        if (resultado!=null) {
            resultado.close();
        }
        if (stmt!=null) {
            stmt.close();
        }
        if (conexion!=null) {
            conexion.close();
        }
    } catch (Exception e) {
        System.out.println("Error en Desconectar Base!");
    }
}
```





Implementando Singleton



Una vez que tengamos la conexión con el objeto Connection encapsulado en un objeto singleton, la vamos a usar para crear un objeto **Statement**, **este objeto recibe la consulta para ejecutarla y enviársela a la base de datos.**



La información que recibimos de la base de datos va a ser capturada por el objeto ResultSet para después poder mostrar la información.



› Obtención de datos: Statement



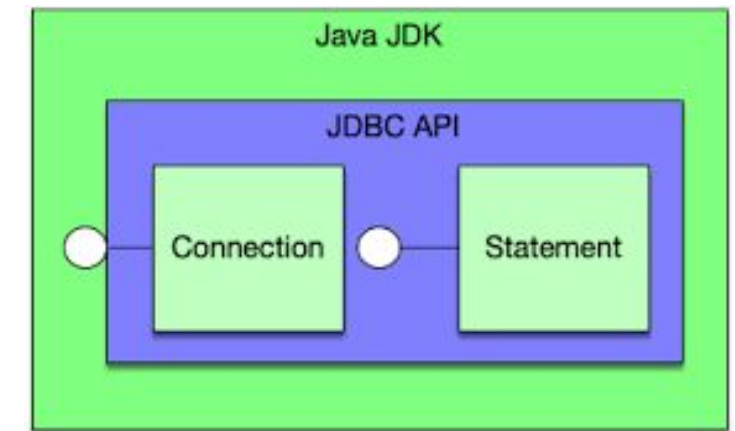
Statement



Una vez realizada la conexión a la BD, se puede utilizar para crear sentencias. Estas sentencias están encapsuladas en la clase **Statement**, y se pueden crear de la siguiente forma:

Statement stmt= conexion.**createStatement()**;

- Los objetos Statement **se obtienen a partir de un objeto Connection.**
- **Permiten realizar consultas SQL** en la base de datos.





Statement

Una vez obtenido el objeto Statement, se puede ejecutar sentencias utilizando su método `executeQuery()` al que se proporciona una cadena con la sentencia SQL que se quiere ejecutar:

`stmt.executeQuery(sentenciaSQL);`

Estas sentencias pueden utilizarse para consultas a la base de datos.



```
// Crear sentencia SQL
String sql = "SELECT * FROM products";

// Preparar sentencia
Statement stmt = conexion.createStatement();

// Ejecutar consulta
stmt.executeQuery(sql);
```




Statement

Las sentencias SQL se van a escribir entre comillas dobles ya que el objeto Statement recibe String, como dato para las sentencias.

`String sentenciaSQL = "SELECT nombre, apellido FROM alumnos";`



```
// Crear sentencia SQL
String sql = "SELECT * FROM products";

// Preparar sentencia
Statement stmt = conexion.createStatement();

// Ejecutar consulta
stmt.executeQuery(sql);
```



Statement



Como podemos ver las sentencias van a tener la **misma sintaxis que en MySQL**, la única **diferencia** se va a presentar a la hora de trabajar con datos de tipo **String** y de tipo **Date**.



Como estos datos suelen ir en comillas dobles en Java y en SQL, y nuestra sentencia ya está entre comillas dobles, **debemos poner los datos entre comillas simples** para diferenciarlos.

```
// sentencia SQL STRING
String sqlString = "SELECT * FROM products WHERE nombre = 'notebook'";

// sentencia SQL DATE
String sqlDate = "SELECT * FROM products WHERE vencimiento = '18-12-2022'";
```

LIVE CODING

Ejemplo en vivo

¡Singleton y Statement!

Vamos a crear una Clase Java que tenga encapsulada toda la conexión con el patrón de diseño Singleton. Esta clase debe tener métodos para conectarse, desconectarse y realizar una consulta (sentencia).

 **Tiempo: 20 minutos**

Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro...

Trabajo Integrador del Módulo 💪

Iremos completándolo progresivamente clase a clase.





Ejercicio N° 1

Singletoneando



Singletoneando

Manos a la obra: 🙌

Vamos a modificar el código que venimos trabajando para que respete el patrón Singleton.

Te invitamos a que generes algunos registros en tu BD Wallet para poder probar las consultas.

Consigna: 📝

- 1- Crear una clase que respete el patrón Singleton de conexión.
- 2- Adecuar el método `createConnection` al patrón de diseño Singleton. Además, debes crear el método `closeConnection` (para cerrar las conexiones existentes).

Tiempo 🕒: 30 minutos



○

¿Alguna consulta?

+



RESUMEN

¿Qué logramos en esta clase?

- ✓ **Comprender la implementación del patrón de diseño Singleton para la conexión.**
- ✓ **Aprender la implementación del objeto Statement.**



#WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 🙌🙌🙌

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
 - a. *Lectura Modulo 5, Lección 4: páginas 4 - 7*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

¡Muchas Gracias!

Nos vemos en la próxima clase 🙌



Momento: ✚

Time-out!

🕒 5 min.

