PATRON REPOSITORY

El patrón Repository es un patrón de diseño que se utiliza en el contexto de la arquitectura de software para separar la lógica de acceso a datos de la lógica de negocio. Actúa como una capa intermedia entre la capa de lógica de negocio y la capa de persistencia (base de datos o cualquier otro almacenamiento). Su principal objetivo es encapsular la lógica de acceso a datos, proporcionando una interfaz para realizar operaciones CRUD (Crear, Leer, Actualizar, Borrar).

Beneficios del Patrón Repository:

- **Desacoplamiento: Separa la lógica de negocio de la lógica de acceso a datos, facilitando cambios en la base de datos sin afectar la lógica de negocio.**
- **Interfaz Uniforme: Proporciona una interfaz consistente para acceder a los datos, independientemente de cómo se almacenan.**
- **Facilita la Prueba: Permite el uso de mocks en pruebas unitarias para simular el comportamiento del repositorio.**

Ejemplo de un programa Java que usa patrón repository

1. Configuración de la Base de Datos

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConfig {
    private static final String URL =
"jdbc:mysql://localhost:3306/testdb";
    private static final String USER = "root";
    private static final String PASSWORD = "password";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

2. DTOs

```java
public class UserDTO {
    private int id;
    private String name;
    private String email;

    public UserDTO(int id, String name, String email) {
        this.id = id;
        this.name = name;
        this.email = email;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

3. Repositorio

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class UserRepository {
    public UserDTO findById(int id) throws SQLException {
        String query = "SELECT * FROM users WHERE id = " + id;
        try (Connection connection = DatabaseConfig.getConnection();
             Statement statement = connection.createStatement();
             ResultSet resultSet = statement.executeQuery(query)) {
            if (resultSet.next()) {
                return new UserDTO(
                        resultSet.getInt("id"),
                        resultSet.getString("name"),
                        resultSet.getString("email")
                );
            } else {
                return null;
            }
        }
    }

    public void save(UserDTO user) throws SQLException {
        String query = "INSERT INTO users (name, email) VALUES ('" +
user.getName() + "', '" + user.getEmail() + "')";
        try (Connection connection = DatabaseConfig.getConnection();
             Statement statement = connection.createStatement()) {
            statement.executeUpdate(query);
        }
    }
}
```

4. Servicio

```
public class UserService {
    private UserRepository userRepository = new UserRepository();

    public UserDTO getUserById(int id) throws SQLException {
        return userRepository.findById(id);
    }

    public void createUser(String name, String email) throws
SQLException, InvalidUserDataException {
        if (!UserValidator.validateName(name) ||
!UserValidator.validateEmail(email)) {
            throw new InvalidUserDataException("Invalid user data");
        }
        UserDTO user = new UserDTO(0, name, email);
        userRepository.save(user);
    }
}
```

5. Validador

```
public class UserValidator {
    public static boolean validateName(String name) {
        return name != null && !name.trim().isEmpty();
    }

    public static boolean validateEmail(String email) {
        return email != null && email.contains("@");
    }
}
```

6. Excepciones

```
public class InvalidUserDataException extends Exception {
    public InvalidUserDataException(String message) {
        super(message);
    }
}
```

7. Clase Principal

```java
import java.sql.SQLException;

public class Main {
    public static void main(String[] args) {
        UserService userService = new UserService();

        try {
            // Crear un nuevo usuario
            userService.createUser("John Doe", "john.doe@example.com");

            // Obtener el usuario por ID
            UserDTO user = userService.getUserById(1);
            if (user != null) {
                System.out.println("User ID: " + user.getId());
                System.out.println("User Name: " + user.getName());
                System.out.println("User Email: " + user.getEmail());
            } else {
                System.out.println("User not found.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (InvalidUserDataException e) {
            System.err.println(e.getMessage());
        }
    }
}
```