

ACTIVIDAD DE APRENDIZAJE

GA7-220501096-AA2-EV01

Proyecto

CODIFICACIÓN DE MÓDULOS DEL SOFTWARE SEGÚN REQUERIMIENTOS DEL
PROYECTO

Realizado por

NATALY MSOCOSO GUZMÁN

SERVICIO NACIONAL DE APRENDIZAJE SENA

ANÁLISIS Y DESARROLLO DE SOFTWARE

2024

INTRODUCCIÓN

En el contexto de la programación y el desarrollo de software, una de las habilidades más fundamentales es la capacidad de interactuar con bases de datos. Este proyecto se centra en el desarrollo de un **módulo de gestión de datos** utilizando **Java con Maven** y **JDBC**, con el fin de realizar operaciones básicas sobre una base de datos. El objetivo es implementar un sistema que gestione información de empleados, permitiendo realizar operaciones de **inserción, consulta, actualización y eliminación** (conocidas como operaciones CRUD).

Durante el desarrollo de este proyecto, se siguieron las mejores prácticas y estándares de codificación, lo que incluye la correcta nomenclatura de clases, métodos, variables y paquetes, así como la estructuración adecuada del código y el uso de herramientas de versionamiento. Además, se prestó especial atención al diseño del sistema mediante la creación de diagramas de clases, diagramas de casos de uso y el uso de historias de usuario que guiaron el desarrollo del sistema.

Para el cumplimiento de los objetivos técnicos del proyecto, se hizo uso de herramientas de desarrollo como **Apache NetBeans IDE 23**, el **JDK 23** y **MySQL** como sistema gestor de base de datos. Adicionalmente, se implementó **Git** para el manejo y versionado del código, permitiendo así llevar un control adecuado de los cambios durante todo el ciclo de vida del proyecto.

El proceso comenzó con la configuración inicial del proyecto, la creación de la base de datos y las tablas necesarias, y la implementación de las clases Java para manejar la lógica de negocio y la conexión a la base de datos. Posteriormente, se definieron las operaciones CRUD dentro de la aplicación, lo que permitió gestionar la información de manera eficiente. Finalmente, se realizó una correcta integración con **Git** para el control de versiones y la publicación del proyecto en un repositorio remoto de **GitHub**.

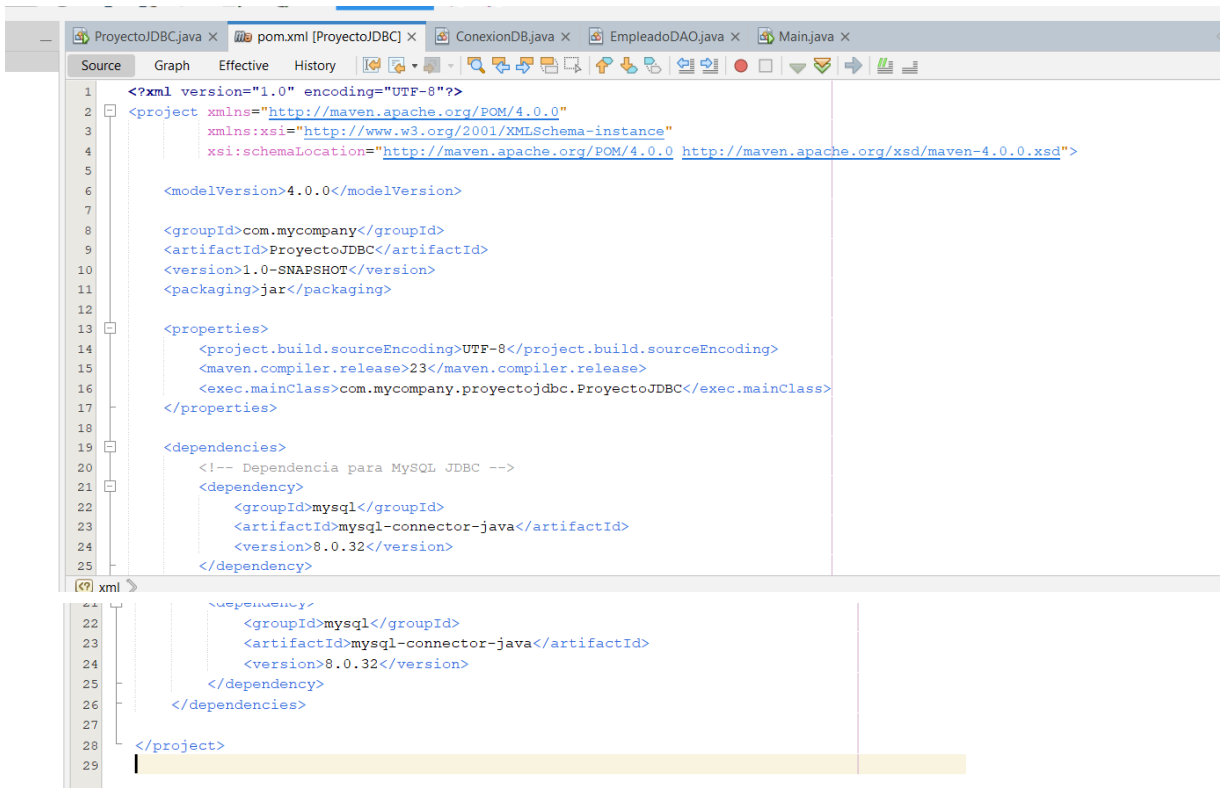
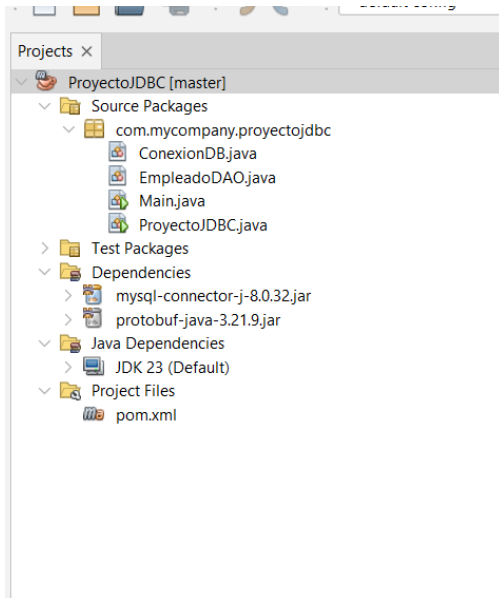
OBJETIVO

El objetivo principal de este proyecto es desarrollar un sistema de gestión de empleados en Java utilizando **JDBC** para realizar operaciones de **inserción, consulta, actualización y eliminación** de registros en una base de datos. Este sistema debe cumplir con los siguientes requisitos:

1. **Gestión de Base de Datos:** Crear y administrar una base de datos de empleados utilizando MySQL.
2. **Operaciones CRUD:** Implementar funcionalidades para insertar, consultar, actualizar y eliminar datos en la base de datos.
3. **Estándares de Codificación:** El código debe seguir las mejores prácticas de codificación, incluyendo el uso adecuado de nombres de clases, métodos, variables y paquetes.
4. **Uso de Maven:** Utilizar Maven para gestionar las dependencias del proyecto, como el conector de MySQL para JDBC.
5. **Herramientas de Versionamiento:** Usar **Git** y **GitHub** para el control de versiones del código y la gestión de cambios a lo largo del desarrollo del proyecto.
6. **Conexión y Configuración Correcta:** Asegurarse de que la conexión a la base de datos se realice correctamente y que las operaciones CRUD se lleven a cabo sin errores.

EJECUCIÓN

1. Creé un nuevo proyecto con Maven. Esto me permitió gestionar las dependencias del proyecto de manera más sencilla. En este caso, agregué la dependencia de MySQL JDBC en el archivo pom.xml, que Maven maneja automáticamente para conectar mi proyecto con la base de datos.



2. Se gestiona la base de datos y la tabla

En este proyecto, decidí crear una base de datos llamada `empleadosdb` en MySQL. Luego, definí una tabla llamada `empleados` con los siguientes campos:

- `id` (int, clave primaria)
- `nombre` (varchar)
- `edad` (int)

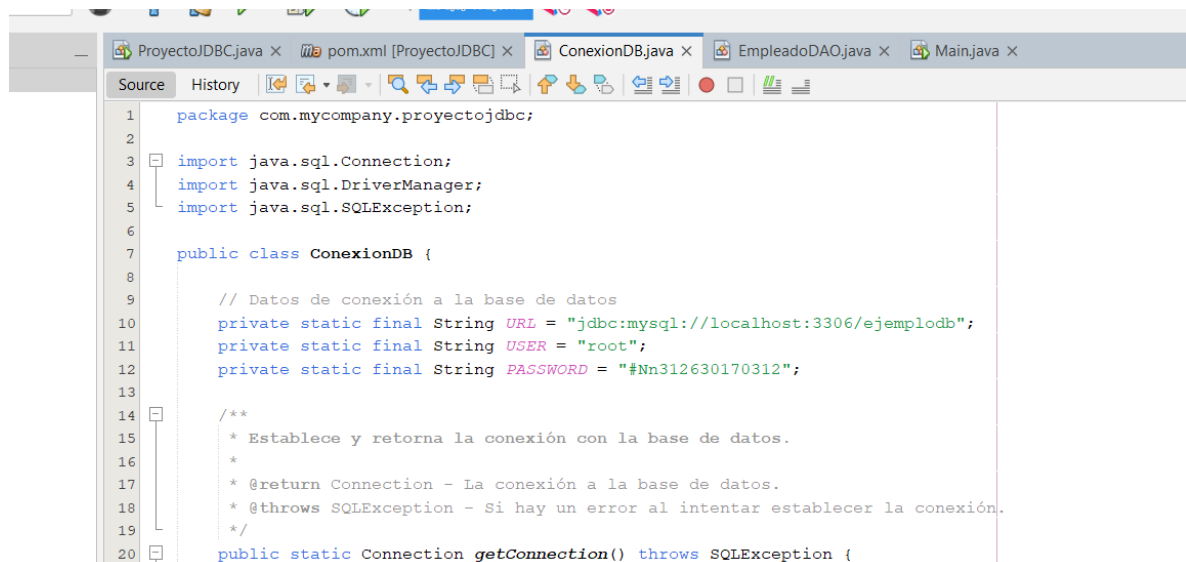
El código SQL para crear la base de datos y la tabla fue el siguiente



```
1 CREATE DATABASE ejemploDB;
2
3 USE ejemploDB;
4
5 CREATE TABLE empleados (
6     id INT AUTO_INCREMENT PRIMARY KEY,
7     nombre VARCHAR(100),
8     salario DOUBLE
9 );
```

3. Conexión a la Base de Datos con JDBC

Una parte clave del proyecto fue establecer la conexión con la base de datos. Para ello, creé la clase `ConexionDB.java`, que contiene un método `getConnection()` para manejar la conexión utilizando JDBC.



```
1 package com.mycompany.proyectojdbc;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class ConexionDB {
8
9     // Datos de conexión a la base de datos
10    private static final String URL = "jdbc:mysql://localhost:3306/empleadb";
11    private static final String USER = "root";
12    private static final String PASSWORD = "#Nn312630170312";
13
14    /**
15     * Establece y retorna la conexión con la base de datos.
16     *
17     * @return Connection - La conexión a la base de datos.
18     * @throws SQLException - Si hay un error al intentar establecer la conexión.
19     */
20    public static Connection getConnection() throws SQLException {
```

```

20 public static Connection getConnection() throws SQLException {
21     try {
22         // Cargar el driver JDBC para MySQL
23         Class.forName("com.mysql.cj.jdbc.Driver"); // Asegúrate de tener el driver de MySQL configurado en tu proyecto
24
25         // Establecer y devolver la conexión
26         return DriverManager.getConnection(URL, USER, PASSWORD);
27     } catch (ClassNotFoundException e) {
28         // Si no se encuentra el driver de MySQL, lanzar una excepción SQL
29         throw new SQLException("No se pudo encontrar el driver JDBC.", e);
30     }
31 }
32
33 }
34
35
36

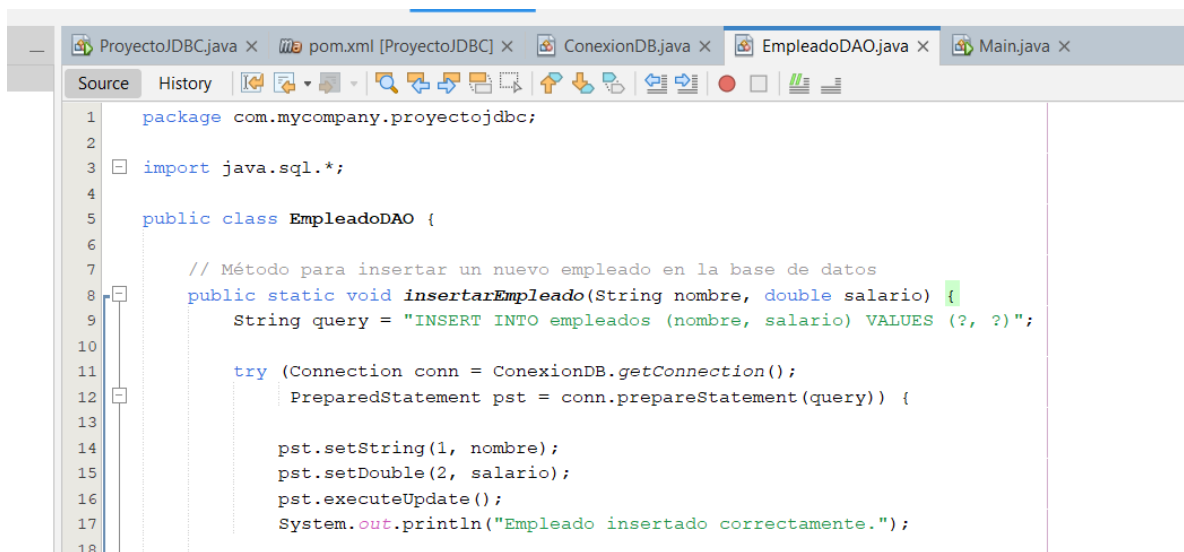
```

Este código maneja la conexión a la base de datos empleadosdb en MySQL. Usamos el DriverManager para conectar a la base de datos usando la URL, el usuario y la contraseña.

4. Implementar las Operaciones CRUD

En esta clase, implementamos los métodos para realizar las operaciones CRUD sobre la base de datos. Cada método utiliza una sentencia SQL diferente para realizar la operación correspondiente.

- Insertar Empleado: El método insertarEmpleado() permite agregar un nuevo empleado a la base de datos.



```

1 package com.mycompany.proyectojdbc;
2
3 import java.sql.*;
4
5 public class EmpleadoDAO {
6
7     // Método para insertar un nuevo empleado en la base de datos
8     public static void insertarEmpleado(String nombre, double salario) {
9         String query = "INSERT INTO empleados (nombre, salario) VALUES (?, ?)";
10
11         try (Connection conn = ConexionDB.getConnection();
12             PreparedStatement pst = conn.prepareStatement(query)) {
13
14             pst.setString(1, nombre);
15             pst.setDouble(2, salario);
16             pst.executeUpdate();
17             System.out.println("Empleado insertado correctamente.");
18         }
19     }
20 }

```

```

18
19 } catch (SQLException e) {
20     System.err.println("Error al insertar empleado: " + e.getMessage());
21 }
22 }
23
24 // Método para consultar un empleado por su ID
25 public static void consultarEmpleado(int id) {
26     String query = "SELECT * FROM empleados WHERE id = ?";
27
28     try (Connection conn = ConexionDB.getConnection();
29         PreparedStatement pst = conn.prepareStatement(query)) {
30
31         pst.setInt(1, id);
32         ResultSet rs = pst.executeQuery();
33
34         if (rs.next()) {
35             String nombre = rs.getString("nombre");
36             double salario = rs.getDouble("salario");
37             System.out.println("Empleado: " + nombre + ", Salario: " + salario);
38         } else {
39             System.out.println("Empleado no encontrado.");
40         }
41
42     } catch (SQLException e) {
43         System.err.println("Error al consultar empleado: " + e.getMessage());
44     }
45 }
46
47 // Método para actualizar los datos de un empleado
48 public static void actualizarEmpleado(int id, String nombre, double salario) {
49     String query = "UPDATE empleados SET nombre = ?, salario = ? WHERE id = ?";
50
51     try (Connection conn = ConexionDB.getConnection();
52         PreparedStatement pst = conn.prepareStatement(query)) {
53
54         pst.setString(1, nombre);
55         pst.setDouble(2, salario);
56         pst.setInt(3, id);
57         pst.executeUpdate();
58         System.out.println("Empleado actualizado correctamente.");
59
60     } catch (SQLException e) {
61         System.err.println("Error al actualizar empleado: " + e.getMessage());
62     }
63 }
64
65 // Método para eliminar un empleado de la base de datos
66 public static void eliminarEmpleado(int id) {
67     String query = "DELETE FROM empleados WHERE id = ?";
68
69     try (Connection conn = ConexionDB.getConnection();
70         PreparedStatement pst = conn.prepareStatement(query)) {
71
72         pst.setInt(1, id);
73         pst.executeUpdate();
74         System.out.println("Empleado eliminado correctamente.");

```

```

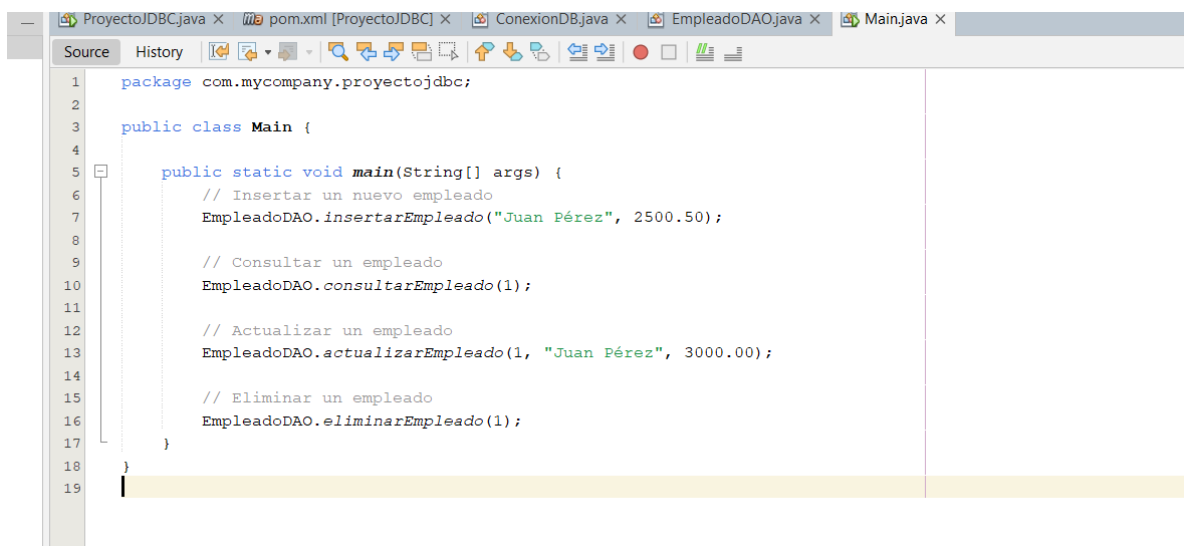
75
76     } catch (SQLException e) {
77         System.err.println("Error al eliminar empleado: " + e.getMessage());
78     }
79 }
80 }
81

```

- **Consultar Empleado:** El método `consultarEmpleado()` permite consultar la información de un empleado dado su id.
- **Actualizar Empleado:** El método `actualizarEmpleado()` permite actualizar los datos de un empleado específico.
- **Eliminar Empleado:** El método `eliminarEmpleado()` permite eliminar un empleado por su id.

5. Probar el Proyecto en el Método main

En el método principal `Main.java`, creé una serie de pruebas para insertar, consultar, actualizar y eliminar empleados. Aquí, llamé a los métodos de `EmpleadoDAO` para interactuar con la base de datos.



```

1 package com.mycompany.proyectojdbc;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         // Insertar un nuevo empleado
7         EmpleadoDAO.insertarEmpleado("Juan Pérez", 2500.50);
8
9         // Consultar un empleado
10        EmpleadoDAO.consultarEmpleado(1);
11
12        // Actualizar un empleado
13        EmpleadoDAO.actualizarEmpleado(1, "Juan Pérez", 3000.00);
14
15        // Eliminar un empleado
16        EmpleadoDAO.eliminarEmpleado(1);
17    }
18 }
19

```

6. Control de Versiones con Git

En la carpeta del proyecto, ejecuté el siguiente comando para inicializar el repositorio

- `git init`

Agregué los archivos del proyecto al repositorio local

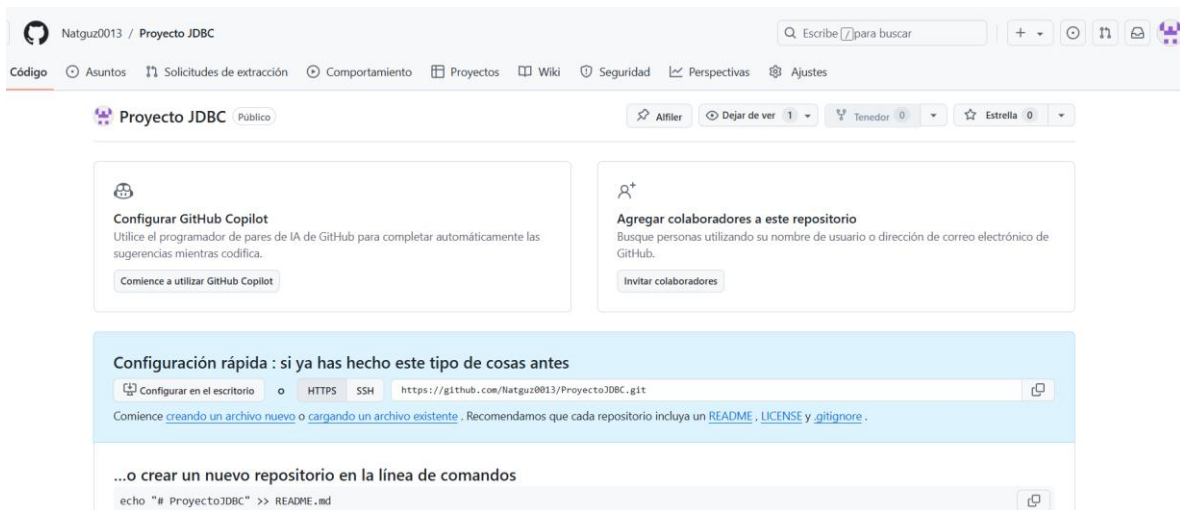
- `git add`

Realicé el primer commit de los archivos

- `git commit -m "Primer commit del proyecto"`

Creé un repositorio en **GitHub**, lo conecté con mi repositorio local y subí los archivos

- `git remote add origin https://github.com/Natguz0013/ProyectoJDBC.git`
- `git push -u origin master`



ENLACE DE PROYECTO GITHUB

<https://github.com/Natguz0013/ProyectoJDBC>

CONCLUSIÓN

Este proyecto me permitió aprender y aplicar JDBC para realizar operaciones CRUD en una base de datos MySQL utilizando Java. Además, integré herramientas de control de versiones como Git para gestionar el código y asegurarme de que el desarrollo fuera organizado. El uso de Maven para gestionar dependencias y la creación de un repositorio en GitHub aseguraron un flujo de trabajo eficiente y una correcta documentación del proceso de desarrollo.

Este proyecto puede ser ampliado y mejorado con funcionalidades adicionales, como validaciones de entrada, gestión de excepciones más detallada, o un diseño de interfaz de usuario para interactuar con la base de datos de manera más amigable.