Trabajo Final Integrador
Aplicación Java con DAO y MySQL

Link al DRIVE
Link al VIDEO
Link al REPOSITORIO

Materia: Programación 2 – UTN
Curso: Tecnicatura Universitaria en Programación
Grupo 27
Fecha: 17 - 11 - 2025

Integrantes y Roles

Integrante	Comisión	Rol en el proyecto
Yamila Fernández	2	UML y DAO
Alicia Yasmin Ahumada	1	Entidades (genérico)
Camila Lihuen Rissi	4	Base de datos y scripts SQL
Rocio Milagros Ocete	3	Service y App Menú consola

1. 1. Elección del dominio y justificación

Para el desarrollo de la aplicación se eligió el dominio Usuario → CredencialAcceso.

- *Justificación:*
    - Este dominio permite modelar un escenario real de autenticación de usuarios, donde cada usuario posee exactamente una credencial de acceso.
    - La relación entre clases es 1→1 unidireccional, lo que refleja que un usuario conoce su credencial, pero la credencial no necesita referencia al usuario.
    - Permite implementar controles de seguridad como baja lógica, activación/desactivación de usuarios y forzar cambios de contraseña.
    - Es un dominio que permite aplicar el patrón DAO con operaciones transaccionales y mantener código limpio y modular.

2. Diseño y UML

2.1 Decisiones clave de diseño

Se eligió una relación 1→1 unidireccional entre Usuario y CredencialAcceso.

Cada usuario tiene exactamente una credencial asociada.

La credencial no referencia al usuario en el modelo de objetos para mantener la simplicidad.

La relación se implementa en la base de datos con una foreign key única en la tabla credencial_acceso (a_id), garantizando que cada credencial pertenece a un único usuario.

Se consideró la alternativa de PK compartida, pero se eligió la FK única por ser más flexible ante futuras extensiones (p. ej. agregar historial de credenciales).

2.2 Resumen del Diagrama UML

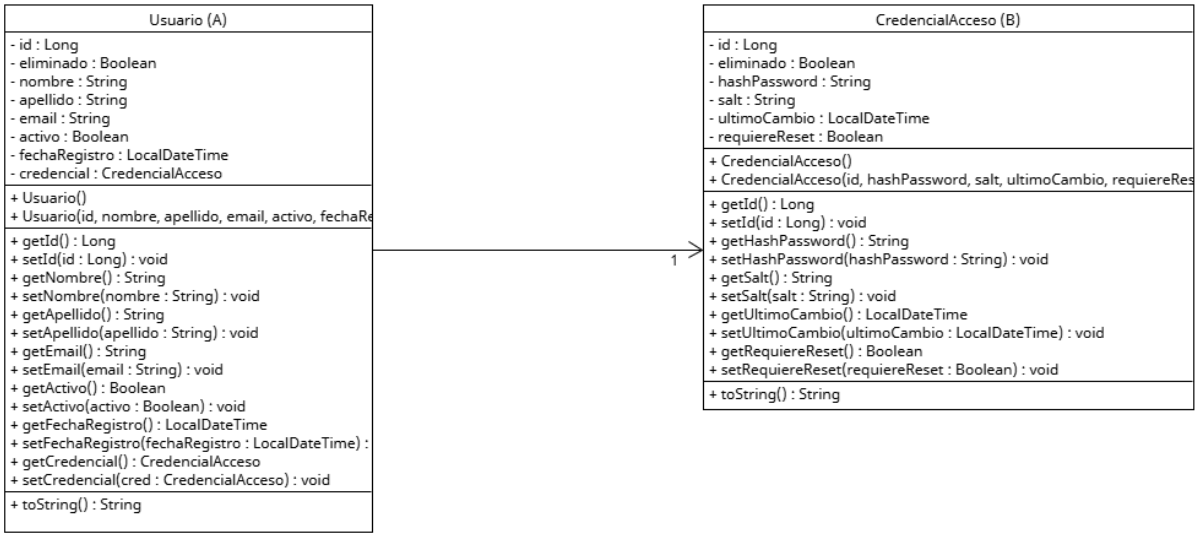
El diagrama refleja la estructura de las clases y su relación:

Usuario (Clase A): Contiene información personal (nombre, apellido, email), control de estado (activo, eliminado), fecha de registro y referencia a su CredencialAcceso.

CredencialAcceso (Clase B): Almacena información de autenticación (hashPassword, salt, ultimoCambio, requiereReset) y atributos de control (id, eliminado).

La relación 1→1 unidireccional indica que un usuario conoce su credencial, mientras que la credencial no necesita referenciar al usuario directamente en el modelo de objetos.

Ambas clases incluyen constructores, getters, setters y toString() para facilitar la manipulación de los objetos en la aplicación.



3. MySQL

El proyecto adopta una arquitectura por capas que permite mantener una clara separación de responsabilidades, mejorar la modularidad y asegurar un mantenimiento adecuado a medida que el sistema crece.

Estructura de la Base de Datos

La base de datos utilizada se denomina tfi_programacion2 y modela la relación uno a uno entre un usuario del sistema y su credencial de acceso, garantizando integridad y consistencia.

a) Tabla usuario

La tabla usuario almacena información general vinculada a cada persona registrada en el sistema. Contiene los siguientes campos principales:

- id como clave primaria autoincremental,
- username y email, ambos únicos para evitar duplicados,
- indicadores de estado como activo y eliminado,
- y fecha_registro, que permite mantener trazabilidad sobre la creación del usuario.

b) Tabla credencial_acceso

La tabla credencial_acceso concentra la información de autenticación asociada a cada usuario. Incluye:

- hash_password y salt, utilizados para almacenar contraseñas de forma segura
- el campo requiere_reset, que permite determinar si la credencial necesita ser modificada
- ultimo_cambio, para registrar el momento del último ajuste de contraseña
- y usuario_id, una clave foránea única que representa la relación uno a uno

c) Relación entre tablas

La relación entre usuario y credencial_acceso se implementa mediante una foreign key con restricción on delete cascade

Estructura del Proyecto en Java

El proyecto se organizó en paquetes que separan funcionalmente cada capa del sistema. Esta organización facilita el mantenimiento, el crecimiento de la aplicación y la reutilización de componentes.

a) Paquete config

Contiene la clase DatabaseConnection, responsable de establecer la comunicación con MySQL.
Aquí se gestiona:

- la carga del driver JDBC,
- la configuración de la conexión,
- y el manejo centralizado de accesos a la base de datos.

b)Paquete entities

Incluye las clases modelo que representan las entidades de la base de datos:

- Usuario, que almacena la información general y referencia a la credencial,
- CredencialAcceso, donde se mantiene todo lo vinculado a la seguridad y autenticación.

c) Paquete dao

La capa DAO constituye el puente directo con MySQL, utilizando JDBC junto con PreparedStatement, lo que previene inyecciones SQL y mejora el rendimiento.

- UsuarioDaoJdbc, que administra todos los CRUD sobre usuarios, incluyendo la obtención de la credencial mediante consultas combinadas
- CredencialDaoJdbc, donde se gestionan las credenciales y sus operaciones dentro de transacciones coordinadas

d) Paquete service

- la creación simultánea de usuario y credencial en una misma transacción
- el manejo de validaciones de integridad
- la ejecución de bajas lógicas
- y el aseguramiento de que todas las operaciones críticas se realicen con commit y rollback controlados

e) Paquete main

A través de AppMenu, el usuario puede interactuar con el sistema desde la consola.
El menú facilita operaciones como:

- creación de un usuario con su credencial
- listado general
- visualización completa por ID
- actualización de datos
- ejecución de bajas lógicas

Flujo General del Sistema

El funcionamiento del sistema sigue una secuencia bien definida:

1. El usuario selecciona una opción desde el menú principal
2. El menú delega la operación solicitada al servicio correspondiente
3. UsuarioService valida los datos, abre una conexión y establece una transacción
4. La operación se ejecuta mediante los DAOs, que interactúan con la base de datos
5. Si todas las acciones finalizan correctamente, la transacción se confirma (commit.
6. Ante cualquier error, se revierten los cambios (rollback)
7. La aplicación informa el resultado por consola mediante mensajes claros y comprensibles

4. DAO

En el punto 4 implementamos la capa de acceso a datos (DAO) para las entidades Usuario y CredencialAcceso utilizando JDBC en Java.

1. GenericDao<T>

Creamos la interfaz genérica GenericDao<T> con los métodos CRUD:

crear(T entity)

leer(long id)

leerTodos()

actualizar(T entity)

eliminar(long id)

Esta interfaz sirve como manual para cualquier DAO, asegurando consistencia y reutilización de código.

2. DAOs concretos

Implementamos UsuarioDAOImpl y CredencialAccesoDAOImpl.

Todos los métodos usan PreparedStatement para interactuar con la base de forma segura, evitando inyecciones SQL.

UsuarioDAOImpl maneja la relación 1→1 unidireccional con CredencialAcceso, creando o eliminando la credencial asociada automáticamente.

3. Uso de Connection y transacciones

Todos los métodos aceptan un Connection externa, permitiendo que varias operaciones se realicen en la misma transacción.

Se usa conn.setAutoCommit(false) para controlar commits y rollbacks:

Si alguna operación falla, se hace rollback y se deshacen los cambios.

Si todo está correcto, se hace commit y se guardan los cambios en la base.

5. Implementación de la capa service y manejo de transacciones

Para la capa de servicios implementamos el patrón Service, utilizando transacciones JDBC para garantizar la consistencia de los datos en operaciones que involucren más de una tabla. Se definió una interfaz genérica GenericService<T> que define los métodos básicos de CRUD:

```java
package Service;

import java.util.List;

public interface GenericService<T> {

    T insertar(T entity) throws Exception;

    T actualizar(T entity) throws Exception;

    boolean eliminar(long id) throws Exception;

    T getById(long id) throws Exception;

    List<T> getAll() throws Exception;
}
```

1.Servicio de CredencialAcceso

El servicio CredencialAccesoService implementa GenericService<CredencialAcceso> y se encarga de manejar:

- Validaciones de campos obligatorios (hashPassword y salt).
- Transacciones completas: se abre un Connection con autoCommit(false), se ejecuta la operación, y se realiza commit() si todo sale bien o rollback() ante cualquier error.
- Conexión a la base de datos usando DatabaseConfig.getConnection().

Ejemplo de método de inserción:

```java
@Override
public CredencialAcceso insertar(CredencialAcceso cred) throws Exception {
    if (cred == null) {
        throw new IllegalArgumentException("La credencial no puede ser null.");
    }

    try (Connection conn = DatabaseConfig.getConnection()) {
        conn.setAutoCommit(false);
        try {
            validar(cred);
            CredencialAcceso creada = credDao.crear(cred, conn);
            conn.commit();
            return creada;
        } catch (Exception e) {
            conn.rollback();
            throw new Exception("Error al insertar la credencial: " + e.getMessage(), e);
        }
    }
}
```

2. Servicio de Usuario con relación 1→1

El servicio UsuarioService implementa la lógica 1→1 unidireccional, asegurando que cada Usuario pueda asociarse a una sola CredencialAcceso.

- Primero se valida el usuario y luego se crea o actualiza la credencial.
- La transacción asegura que ambos registros (Usuario y Credencial) se persistan juntos.
- La operación de eliminación también se realiza de forma transaccional.

```java
// 1 → Primero crear credencial
if (usuario.getCredencial() != null) {
    credDao.crear(usuario.getCredencial(), conn);
}

// 2 → Después crear usuario
Usuario creado = usuarioDao.crear(usuario, conn);

conn.commit();
```

3. Validaciones implementadas

- UsuarioService: username y email obligatorios, email con formato válido.
- CredencialAccesoService: hashPassword y salt obligatorios.

- Prevención de inconsistencias en la relación 1→1.

6. Implementación del menú de consola (AppMenu)

Se desarrolló AppMenu para permitir CRUD completo de Usuario y Credencial mediante consola.

1. Menú principal

Permite elegir entre operaciones sobre usuarios o credenciales:

```
System.out.println("\n===== MENU PRINCIPAL =====");
System.out.println("1) Usuarios");
System.out.println("2) Credenciales");
System.out.println("0) Salir");
System.out.print("Elija opción: ");
```

2. Menú de usuarios

Incluye:

- Crear usuario con credencial asociada.
- Leer usuario por ID.
- Listar todos los usuarios.
- Actualizar y eliminar.
- Búsqueda por username.
  Fragmento de creación de usuario:

```java
    private void crearUsuario() throws Exception {

        Usuario u = new Usuario();

        System.out.print("Username: ");
        u.setUsername(sc.nextLine());

        System.out.print("Email: ");
        u.setEmail(sc.nextLine());

        u.setActivo(true);
        u.setFechaRegistro(LocalDateTime.now());
        u.setEliminado(false);

        // credencial asociada
        CredencialAcceso c = new CredencialAcceso();
        System.out.print("Hash password: ");
        c.setHashPassword(sc.nextLine());
        c.setSalt("salt-" + System.currentTimeMillis());
        c.setUltimoCambio(LocalDateTime.now());
        c.setRequiereReset(false);
        c.setEliminado(false);

        u.setCredencial(c);

        usuarioService.insertar(u);

        System.out.println("✓ Usuario creado con éxito");
    }
```

3. Menú de credenciales

Permite operaciones CRUD independientes sobre CredencialAcceso, con validaciones y manejo de errores:

- Crear credencial.
- Leer por ID.
- Listar todas.
- Actualizar y eliminar.

Ejemplo de actualización:

```java
private void actualizarCredencial() throws Exception {
    System.out.print("ID de la credencial a actualizar: ");
    long id = Long.parseLong(sc.nextLine());

    CredencialAcceso c = credService.getById(id);

    if (c == null) {
        System.out.println("Credencial no encontrada.");
        return;
    }

    System.out.print("Nuevo hash password: ");
    c.setHashPassword(sc.nextLine());
    c.setUltimoCambio(LocalDateTime.now());

    credService.actualizar(c);

    System.out.println("✓ Credencial actualizada.");
}
```

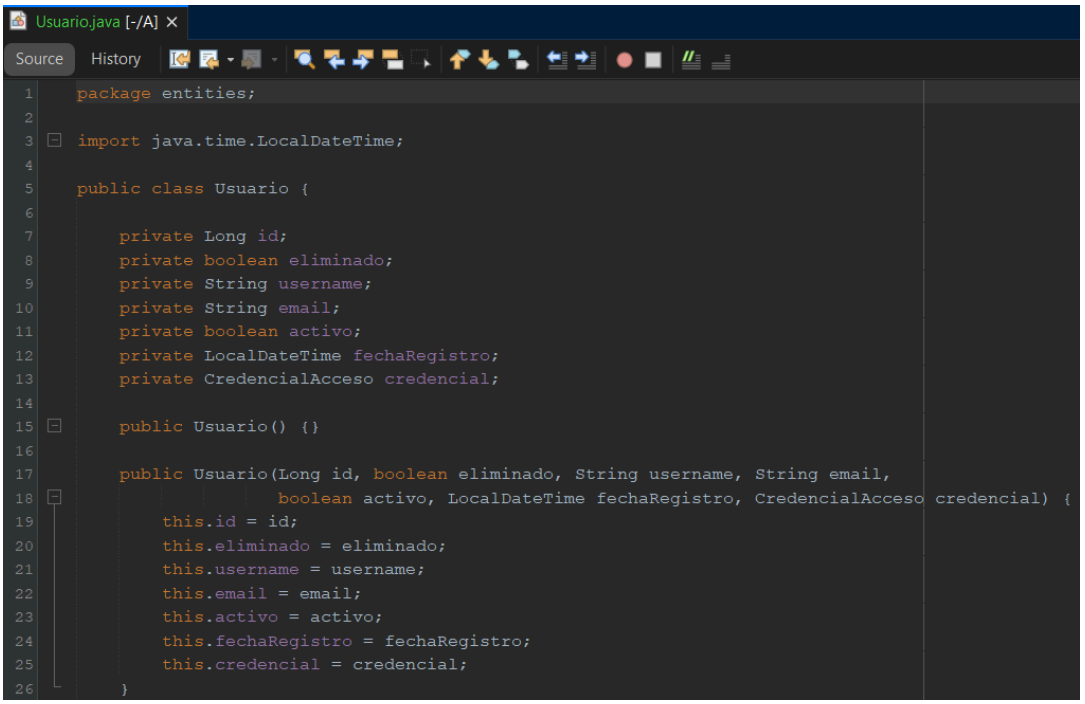
4. Manejo de errores y robustez

- Se manejan entradas inválidas (parseos numéricos, strings vacíos, formato de email).
- Se controlan IDs inexistentes.
- Se capturan errores de base de datos, violaciones de unicidad y errores de validación.
- Se muestran mensajes claros de éxito o error.

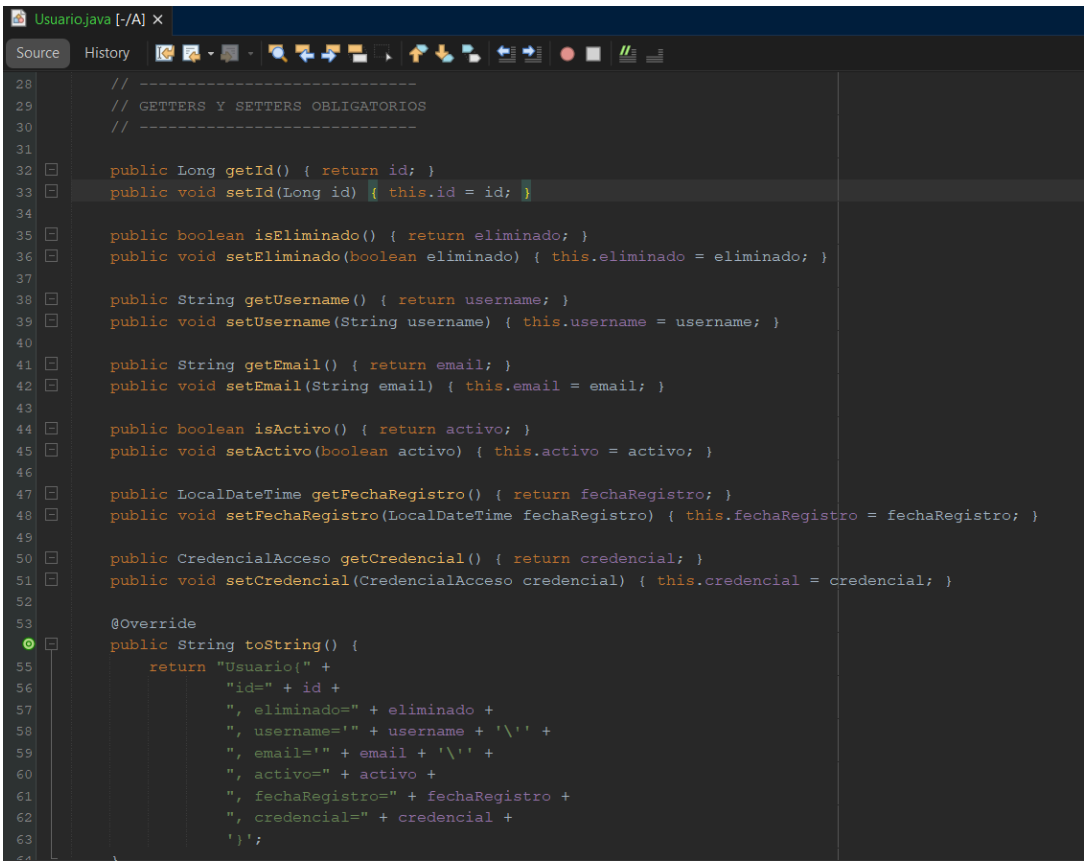
Capturas de pantalla

- Package ENTITIES

Usuario.JAVA




```java
package entities;

import java.time.LocalDateTime;

public class Usuario {

    private Long id;
    private boolean eliminado;
    private String username;
    private String email;
    private boolean activo;
    private LocalDateTime fechaRegistro;
    private CredencialAcceso credencial;

    public Usuario() {}

    public Usuario(Long id, boolean eliminado, String username, String email,
                   boolean activo, LocalDateTime fechaRegistro, CredencialAcceso credencial) {
        this.id = id;
        this.eliminado = eliminado;
        this.username = username;
        this.email = email;
        this.activo = activo;
        this.fechaRegistro = fechaRegistro;
        this.credencial = credencial;
    }
```




```java
    // ----------------------------
    // GETTERS Y SETTERS OBLIGATORIOS
    // ----------------------------

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public boolean isEliminado() { return eliminado; }
    public void setEliminado(boolean eliminado) { this.eliminado = eliminado; }

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

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

    public boolean isActivo() { return activo; }
    public void setActivo(boolean activo) { this.activo = activo; }

    public LocalDateTime getFechaRegistro() { return fechaRegistro; }
    public void setFechaRegistro(LocalDateTime fechaRegistro) { this.fechaRegistro = fechaRegistro; }

    public CredencialAcceso getCredencial() { return credencial; }
    public void setCredencial(CredencialAcceso credencial) { this.credencial = credencial; }

    @Override
    public String toString() {
        return "Usuario{" +
                "id=" + id +
                ", eliminado=" + eliminado +
                ", username='" + username + '\'' +
                ", email='" + email + '\'' +
                ", activo=" + activo +
                ", fechaRegistro=" + fechaRegistro +
                ", credencial=" + credencial +
                '}';
    }
```

CredencialAcceso.JAVA

CredencialAcceso.java [-/A]

Source History

```java
package entities;

import java.time.LocalDateTime;

public class CredencialAcceso {

    private Long id;
    private boolean eliminado;
    private String hashPassword;
    private String salt;
    private LocalDateTime ultimoCambio;
    private boolean requiereReset;
    private Long usuarioId; // Nueva propiedad para relacionar con Usuario

    public CredencialAcceso() {}

    public CredencialAcceso(Long id, boolean eliminado, String hashPassword, String salt,
                            LocalDateTime ultimoCambio, boolean requiereReset) {
        this.id = id;
        this.eliminado = eliminado;
        this.hashPassword = hashPassword;
        this.salt = salt;
        this.ultimoCambio = ultimoCambio;
        this.requiereReset = requiereReset;
    }
```

CredencialAcceso.java [-/A]

Source History

```java
    // ----------------------------
    // GETTERS Y SETTERS
    // ----------------------------
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public boolean isEliminado() { return eliminado; }
    public void setEliminado(boolean eliminado) { this.eliminado = eliminado; }

    public String getHashPassword() { return hashPassword; }
    public void setHashPassword(String hashPassword) { this.hashPassword = hashPassword; }

    public String getSalt() { return salt; }
    public void setSalt(String salt) { this.salt = salt; }

    public LocalDateTime getUltimoCambio() { return ultimoCambio; }
    public void setUltimoCambio(LocalDateTime ultimoCambio) { this.ultimoCambio = ultimoCambio; }

    public boolean isRequiereReset() { return requiereReset; }
    public void setRequiereReset(boolean requiereReset) { this.requiereReset = requiereReset; }

    public Long getUsuarioId() { return usuarioId; }
    public void setUsuarioId(Long usuarioId) { this.usuarioId = usuarioId; }

    @Override
    public String toString() {
        return "CredencialAcceso{" +
                "id=" + id +
                ", eliminado=" + eliminado +
                ", hashPassword='" + hashPassword + '\'' +
                ", salt='" + salt + '\'' +
                ", ultimoCambio=" + ultimoCambio +
                ", requiereReset=" + requiereReset +
                ", usuarioId=" + usuarioId +
                '}';
    }
}
```

- Package CONFIG

DatabaseConnection.JAVA

DatabaseConnection.java [-/A] ×

Source | History

```java
package config;

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

public class DatabaseConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/tfi_programacion2?useSSL=false&serverTimezone=UTC";
    private static final String USER = "root";
    private static final String PASS = "Cami123";

    static {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("Driver JDBC no encontrado: " + e.getMessage());
        }
    }

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

DatabaseConfig.JAVA

DatabaseConfig.java [-/A] ×

Source | History

```java
package config;

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

public class DatabaseConfig {

    private static final String URL = "jdbc:mysql://localhost:3307/tfi_programacion2?useSSL=false&serverTimezone=UTC";
    private static final String USER = "root";    // CAMBIAR SI TU MYSQL TIENE OTRO USUARIO
    private static final String PASS = "";        // CAMBIAR SI TENÉS CONTRASEÑA

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

- Package SERVICE

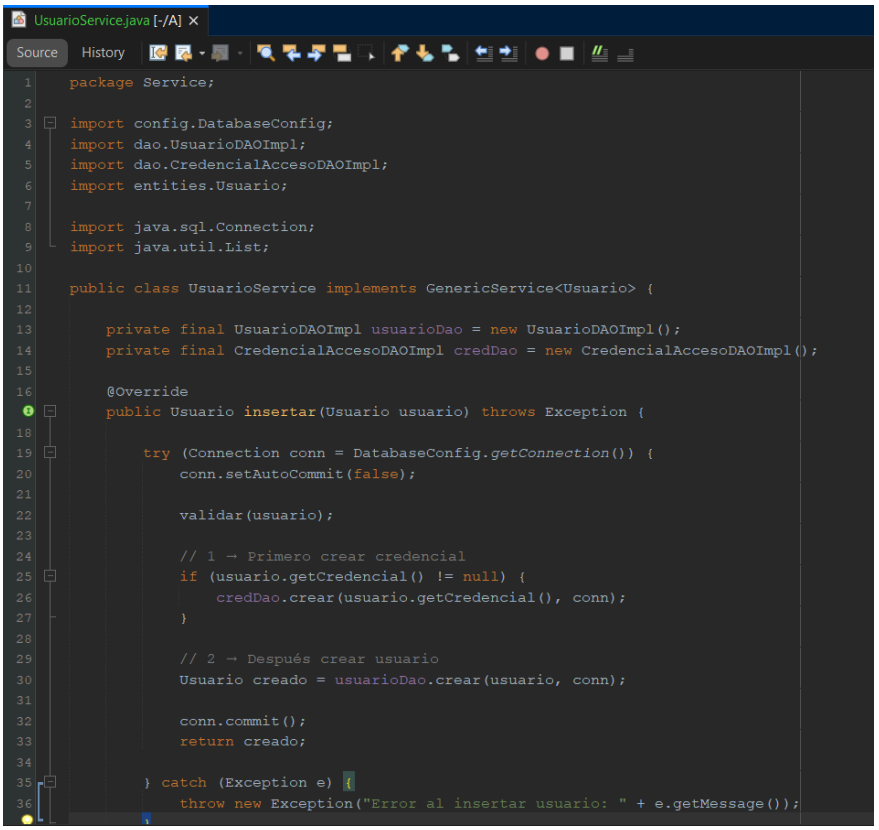
GenericService.JAVA

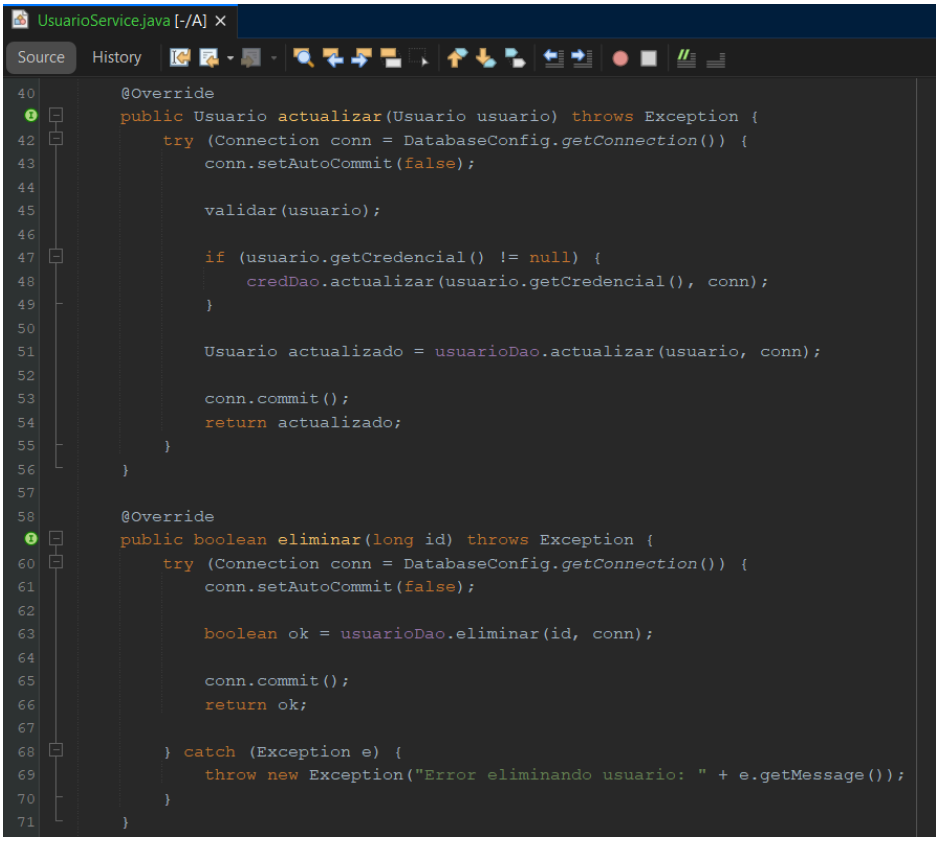
GenericService.java [-/A] ×

Source | History

```java
package Service;

import java.util.List;

public interface GenericService<T> {

    T insertar(T entity) throws Exception;

    T actualizar(T entity) throws Exception;

    boolean eliminar(long id) throws Exception;

    T getById(long id) throws Exception;

    List<T> getAll() throws Exception;
}
```

UsuarioService.JAVA





```java
    @Override
    public Usuario getById(long id) throws Exception {
        try (Connection conn = DatabaseConfig.getConnection()) {
            return usuarioDao.leer(id, conn);
        }
    }

    @Override
    public List<Usuario> getAll() throws Exception {
        try (Connection conn = DatabaseConfig.getConnection()) {
            return usuarioDao.leerTodos(conn);
        }
    }

    private void validar(Usuario u) throws Exception {
        if (u.getUsername() == null || u.getUsername().isBlank())
            throw new Exception("El username no puede estar vacío.");

        if (u.getEmail() == null || u.getEmail().isBlank())
            throw new Exception("El email es obligatorio.");

        if (!u.getEmail().contains("@"))
            throw new Exception("Formato de email inválido.");
    }
}

```

CredencialAccesoService.JAVA

CredencialAccesoService.java [-/A]

Source History

```java
package Service;

import config.DatabaseConfig;
import dao.CredencialAccesoDAOImpl;
import entities.CredencialAcceso;

import java.sql.Connection;
import java.util.List;

public class CredencialAccesoService implements GenericService<CredencialAcceso> {

    private final CredencialAccesoDAOImpl credDao = new CredencialAccesoDAOImpl();

    @Override
    public CredencialAcceso insertar(CredencialAcceso cred) throws Exception {
        if (cred == null) {
            throw new IllegalArgumentException("La credencial no puede ser null.");
        }

        try (Connection conn = DatabaseConfig.getConnection()) {
            conn.setAutoCommit(false);
            try {
                validar(cred);
                CredencialAcceso creada = credDao.crear(cred, conn);
                conn.commit();
                return creada;
            } catch (Exception e) {
                conn.rollback();
                throw new Exception("Error al insertar la credencial: " + e.getMessage(), e);
            }
        }
    }
```

CredencialAccesoService.java [-/A] ×

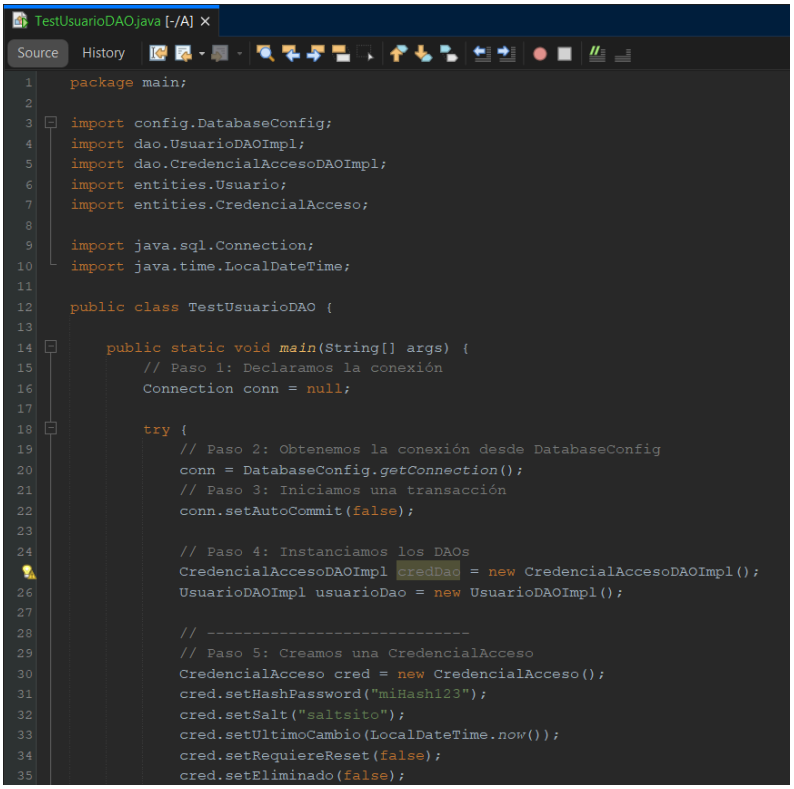
Source | History

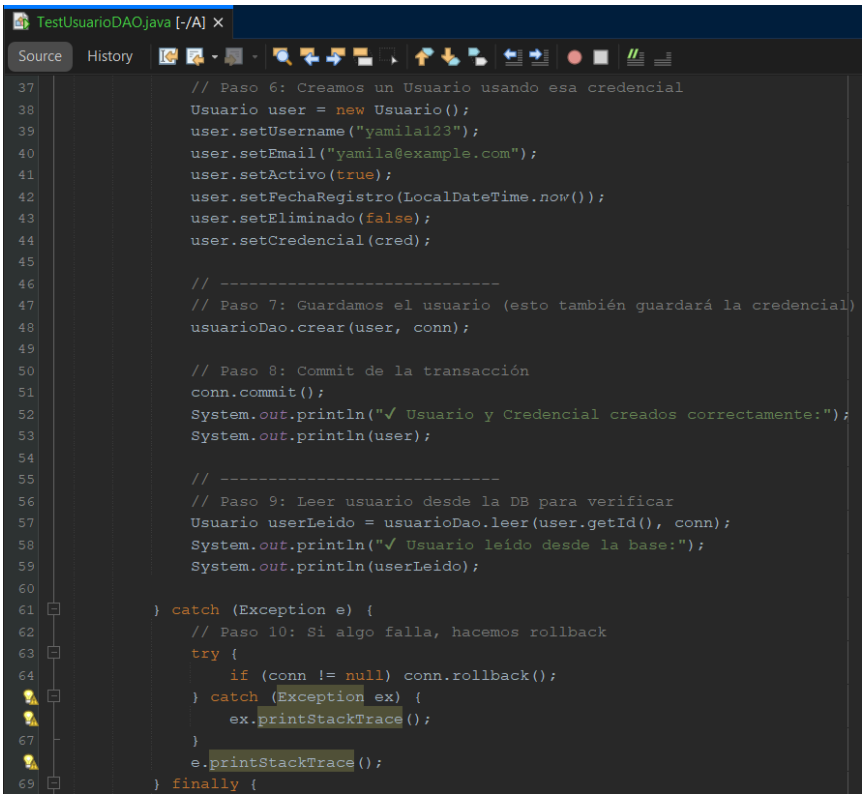
```java
    @Override
    public CredencialAcceso actualizar(CredencialAcceso cred) throws Exception {
        if (cred == null) {
            throw new IllegalArgumentException("La credencial no puede ser null.");
        }

        try (Connection conn = DatabaseConfig.getConnection()) {
            conn.setAutoCommit(false);
            try {
                validar(cred);
                CredencialAcceso updated = credDao.actualizar(cred, conn);
                conn.commit();
                return updated;
            } catch (Exception e) {
                conn.rollback();
                throw new Exception("Error al actualizar la credencial: " + e.getMessage(), e);
            }
        }
    }

    @Override
    public boolean eliminar(long id) throws Exception {
        try (Connection conn = DatabaseConfig.getConnection()) {
            conn.setAutoCommit(false);
            try {
                boolean ok = credDao.eliminar(id, conn);
                conn.commit();
                return ok;
            } catch (Exception e) {
                conn.rollback();
                throw new Exception("Error al eliminar la credencial: " + e.getMessage(), e);
            }
        }
    }
```

```java
    @Override
    public CredencialAcceso getById(long id) throws Exception {
        try (Connection conn = DatabaseConfig.getConnection()) {
            return credDao.leer(id, conn);
        }
    }

    @Override
    public List<CredencialAcceso> getAll() throws Exception {
        try (Connection conn = DatabaseConfig.getConnection()) {
            return credDao.leerTodos(conn);
        }
    }

    private void validar(CredencialAcceso cred) throws Exception {
        if (cred.getHashPassword() == null || cred.getHashPassword().isBlank())
            throw new Exception("La contraseña hash no puede ser vacía.");

        if (cred.getSalt() == null || cred.getSalt().isBlank())
            throw new Exception("El salt es obligatorio.");
    }
}
```

- Package MAIN

TestUsuarioDAO.JAVA





```
            // Paso 11: Cerramos la conexión
            try {
                if (conn != null) conn.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

TestConexion.Java

TestConexion.java [-/A]

Source History

```
package main;

import config.DatabaseConfig;
import java.sql.Connection;

public class TestConexion {
    public static void main(String[] args) {
        try (Connection conn = DatabaseConfig.getConnection()) {
            System.out.println("✓ Conexión exitosa a la base de datos!");
        } catch (Exception e) {
            System.out.println("✗ Error de conexión:");
            e.printStackTrace();
        }
    }
}
```

AppMenu.JAVA

AppMenu.java [-/A]

Source History

```
package main;

import entities.Usuario;
import entities.CredencialAcceso;
import Service.UsuarioService;
import Service.CredencialAccesoService;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Scanner;

public class AppMenu {

    private final Scanner sc = new Scanner(System.in);
    private final UsuarioService usuarioService = new UsuarioService();
    private final CredencialAccesoService credService = new CredencialAccesoService();

    public void iniciar() {

        String op;

        do {
            System.out.println("\n===== MENU PRINCIPAL =====");
            System.out.println("1) Usuarios");
            System.out.println("2) Credenciales");
            System.out.println("0) Salir");
            System.out.print("Elija opción: ");

            op = sc.nextLine().trim();

            switch (op) {
                case "1" -> menuUsuarios();
                case "2" -> menuCredenciales();
                case "0" -> System.out.println("¡Saliendo!");
                default -> System.out.println("Opción inválida.");
```

```
        } while (!op.equals("0"));
    }

    // ============================
    //           USUARIOS
    // ============================

    private void menuUsuarios() {
        String op;

        do {
            System.out.println("\n--- MENU USUARIOS ---");
            System.out.println("1) Crear usuario");
            System.out.println("2) Buscar por ID");
            System.out.println("3) Listar todos");
            System.out.println("4) Actualizar");
            System.out.println("5) Eliminar");
            System.out.println("6) Buscar por Username");
            System.out.println("0) Volver");
            System.out.print("Opción: ");

            op = sc.nextLine();

            try {
                switch (op) {
                    case "1" -> crearUsuario();
                    case "2" -> leerUsuario();
                    case "3" -> listarUsuarios();
                    case "4" -> actualizarUsuario();
                    case "5" -> eliminarUsuario();
                    case "6" -> buscarPorUsername();
                    case "0" -> {}
                    default -> System.out.println("Opción inválida.");
                }
```

```
            } catch (Exception e) {
                System.out.println("⚠ Error: " + e.getMessage());
            }

        } while (!op.equals("0"));
    }

    private void crearUsuario() throws Exception {

        Usuario u = new Usuario();

        System.out.print("Username: ");
        u.setUsername(sc.nextLine());

        System.out.print("Email: ");
        u.setEmail(sc.nextLine());

        u.setActivo(true);
        u.setFechaRegistro(LocalDateTime.now());
        u.setEliminado(false);

        // credencial asociada
        CredencialAcceso c = new CredencialAcceso();
        System.out.print("Hash password: ");
        c.setHashPassword(sc.nextLine());
        c.setSalt("salt-" + System.currentTimeMillis());
        c.setUltimoCambio(LocalDateTime.now());
        c.setRequiereReset(false);
        c.setEliminado(false);

        u.setCredencial(c);

        usuarioService.insertar(u);

        System.out.println("✓ Usuario creado con éxito");
    }
```

```java
    private void leerUsuario() throws Exception {
        System.out.print("ID: ");
        long id = Long.parseLong(sc.nextLine());

        Usuario u = usuarioService.getById(id);

        if (u == null)
            System.out.println("No existe ese usuario.");
        else
            System.out.println(u);
    }

    private void listarUsuarios() throws Exception {
        List<Usuario> lista = usuarioService.getAll();
        lista.forEach(System.out::println);
    }

    private void actualizarUsuario() throws Exception {

        System.out.print("ID del usuario a actualizar: ");
        long id = Long.parseLong(sc.nextLine());

        Usuario u = usuarioService.getById(id);

        if (u == null) {
            System.out.println("Usuario no encontrado.");
            return;
        }

        System.out.print("Nuevo email: ");
        u.setEmail(sc.nextLine());

        usuarioService.actualizar(u);

        System.out.println("✓ Usuario actualizado.");
    }
```

```java
    private void eliminarUsuario() throws Exception {

        System.out.print("ID del usuario: ");
        long id = Long.parseLong(sc.nextLine());

        if (usuarioService.eliminar(id))
            System.out.println("✓ Eliminado.");
        else
            System.out.println("No existe ese ID.");
    }

    private void buscarPorUsername() throws Exception {

        System.out.print("Username: ");
        String username = sc.nextLine();

        List<Usuario> all = usuarioService.getAll();

        all.stream()
                .filter(u -> u.getUsername().equalsIgnoreCase(username))
                .findFirst()
                .ifPresentOrElse(
                        System.out::println,
                        () -> System.out.println("No encontrado.")
                );
    }
```

```java
        // ============================
        //        CREDENCIALES
        // ============================

    private void menuCredenciales() {
        String op;

        do {
            System.out.println("\n--- MENU CREDENCIALES ---");
            System.out.println("1) Crear credencial");
            System.out.println("2) Buscar por ID");
            System.out.println("3) Listar todas");
            System.out.println("4) Actualizar");
            System.out.println("5) Eliminar");
            System.out.println("0) Volver");
            System.out.print("Opción: ");

            op = sc.nextLine();

            try {
                switch (op) {
                    case "1" -> crearCredencial();
                    case "2" -> leerCredencial();
                    case "3" -> listarCredenciales();
                    case "4" -> actualizarCredencial();
                    case "5" -> eliminarCredencial();
                    case "0" -> {}
                    default -> System.out.println("Opción inválida.");
                }
            } catch (Exception e) {
                System.out.println("⚠ Error: " + e.getMessage());
            }

        } while (!op.equals("0"));
    }
```

```java
    // ====================
    // Métodos CRUD de credenciales
    // ====================

    private void crearCredencial() throws Exception {
        CredencialAcceso c = new CredencialAcceso();

        System.out.print("Hash password: ");
        c.setHashPassword(sc.nextLine());

        c.setSalt("salt-" + System.currentTimeMillis());
        c.setUltimoCambio(LocalDateTime.now());
        c.setRequiereReset(false);
        c.setEliminado(false);

        credService.insertar(c);

        System.out.println("✓ Credencial creada con éxito, ID: " + c.getId());
    }

    private void leerCredencial() throws Exception {
        System.out.print("ID: ");
        long id = Long.parseLong(sc.nextLine());

        CredencialAcceso c = credService.getById(id);

        if (c == null)
            System.out.println("No existe esa credencial.");
        else
            System.out.println(c);
    }

    private void listarCredenciales() throws Exception {
        List<CredencialAcceso> lista = credService.getAll();
        lista.forEach(System.out::println);
    }

```

```java
private void actualizarCredencial() throws Exception {
    System.out.print("ID de la credencial a actualizar: ");
    long id = Long.parseLong(sc.nextLine());

    CredencialAcceso c = credService.getById(id);

    if (c == null) {
        System.out.println("Credencial no encontrada.");
        return;
    }

    System.out.print("Nuevo hash password: ");
    c.setHashPassword(sc.nextLine());
    c.setUltimoCambio(LocalDateTime.now());

    credService.actualizar(c);

    System.out.println("✓ Credencial actualizada.");
}

private void eliminarCredencial() throws Exception {
    System.out.print("ID de la credencial: ");
    long id = Long.parseLong(sc.nextLine());

    if (credService.eliminar(id))
        System.out.println("✓ Credencial eliminada.");
    else
        System.out.println("No existe esa ID.");
    }
}
```

Main.JAVA

Main.java [-/A] ×

Source History

```java
package main;

public class Main {
    public static void main(String[] args) {
        new AppMenu().iniciar();
    }
}
```

- Package DAO

UsuarioDAOImpl.JAVA

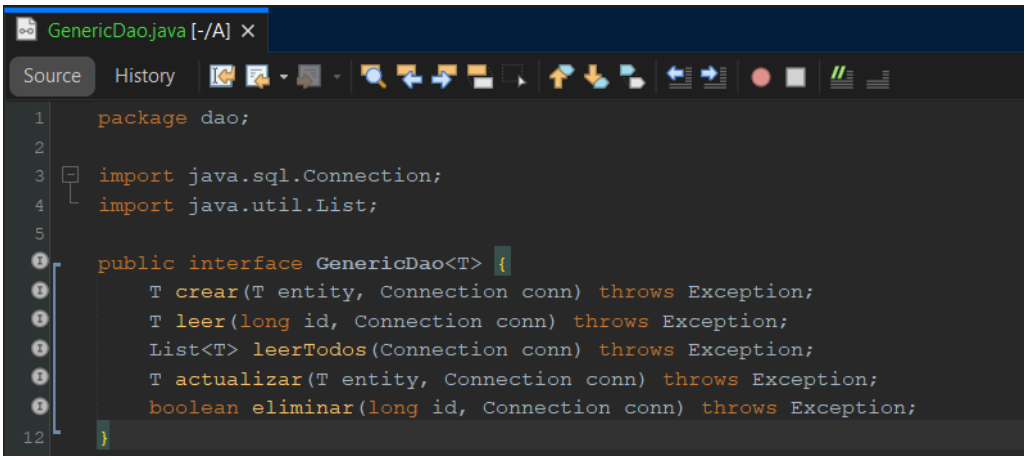
```java
package dao;

import entities.Usuario;
import entities.CredencialAcceso;
import java.sql.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class UsuarioDAOImpl implements GenericDao<Usuario> {

    private final CredencialAccesoDAOImpl credDao = new CredencialAccesoDAOImpl();

    @Override
    public Usuario crear(Usuario entity, Connection conn) throws Exception {
        // Primero crear la credencial si existe
        if (entity.getCredencial() != null) {
            credDao.crear(entity.getCredencial(), conn);
        }

        String sql = "INSERT INTO usuario (username, email, activo, fecha_registro, eliminado, credencial_id) VALUES (?, ?, ?, ?, ?, ?)";
        try (PreparedStatement ps = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
            ps.setString(1, entity.getUsername());
            ps.setString(2, entity.getEmail());
            ps.setBoolean(3, entity.isActivo());
            ps.setTimestamp(4, entity.getFechaRegistro() != null ? Timestamp.valueOf(entity.getFechaRegistro()) : null);
            ps.setBoolean(5, entity.isEliminado());
            ps.setObject(6, entity.getCredencial() != null ? entity.getCredencial().getId() : null);

            ps.executeUpdate();
            try (ResultSet rs = ps.getGeneratedKeys()) {
                if (rs.next()) {
                    entity.setId(rs.getLong(1));
                }
            }
        }
```

```java
        return entity;
    }

    @Override
    public Usuario leer(long id, Connection conn) throws Exception {
        String sql = "SELECT * FROM usuario WHERE id = ?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setLong(1, id);
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    return mapear(rs, conn);
                }
            }
        }
        return null;
    }

    @Override
    public List<Usuario> leerTodos(Connection conn) throws Exception {
        String sql = "SELECT * FROM usuario";
        List<Usuario> lista = new ArrayList<>();
        try (PreparedStatement ps = conn.prepareStatement(sql);
             ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                lista.add(mapear(rs, conn));
            }
        }
        return lista;
    }

    @Override
    public Usuario actualizar(Usuario entity, Connection conn) throws Exception {
        // Primero actualizar la credencial si existe
        if (entity.getCredencial() != null) {
            credDao.actualizar(entity.getCredencial(), conn);
        }
```

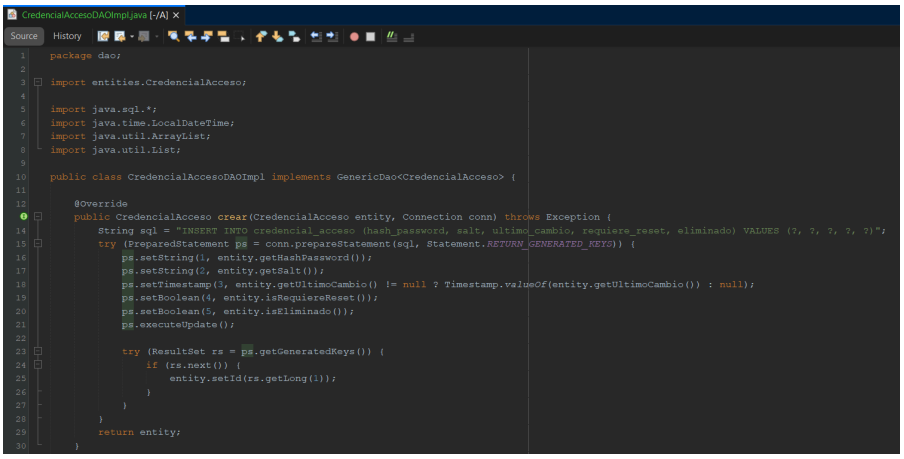
```
        String sql = "UPDATE usuario SET username = ?, email = ?, activo = ?, fecha_registro = ?, eliminado = ?, credencial_id = ? WHERE id = ?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setString(1, entity.getUsername());
            ps.setString(2, entity.getEmail());
            ps.setBoolean(3, entity.isActivo());
            ps.setTimestamp(4, entity.getFechaRegistro() != null ? Timestamp.valueOf(entity.getFechaRegistro()) : null);
            ps.setBoolean(5, entity.isEliminado());
            ps.setObject(6, entity.getCredencial() != null ? entity.getCredencial().getId() : null);
            ps.setLong(7, entity.getId());

            ps.executeUpdate();
        }
        return entity;
    }

    @Override
    public boolean eliminar(long id, Connection conn) throws Exception {
        Usuario usuario = leer(id, conn);
        if (usuario == null) return false;

        String sql = "DELETE FROM usuario WHERE id = ?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setLong(1, id);
            boolean eliminado = ps.executeUpdate() > 0;

            // Eliminar la credencial asociada si existia
            if (eliminado && usuario.getCredencial() != null) {
                credDao.eliminar(usuario.getCredencial().getId(), conn);
            }
            return eliminado;
        }
    }
```

```
    /**
     * Convierte un ResultSet en un objeto Usuario
     */
    private Usuario mapear(ResultSet rs, Connection conn) throws Exception {
        LocalDateTime fecha = rs.getTimestamp("fecha_registro") != null
                ? rs.getTimestamp("fecha_registro").toLocalDateTime()
                : null;

        // Obtener la credencial asociada, si existe
        CredencialAcceso cred = null;
        long credId = rs.getLong("credencial_id");
        if (!rs.wasNull()) {
            cred = credDao.leer(credId, conn);
        }

        return new Usuario(
                rs.getLong("id"),
                rs.getBoolean("eliminado"),
                rs.getString("username"),
                rs.getString("email"),
                rs.getBoolean("activo"),
                fecha,
                cred
        );
    }
}
```

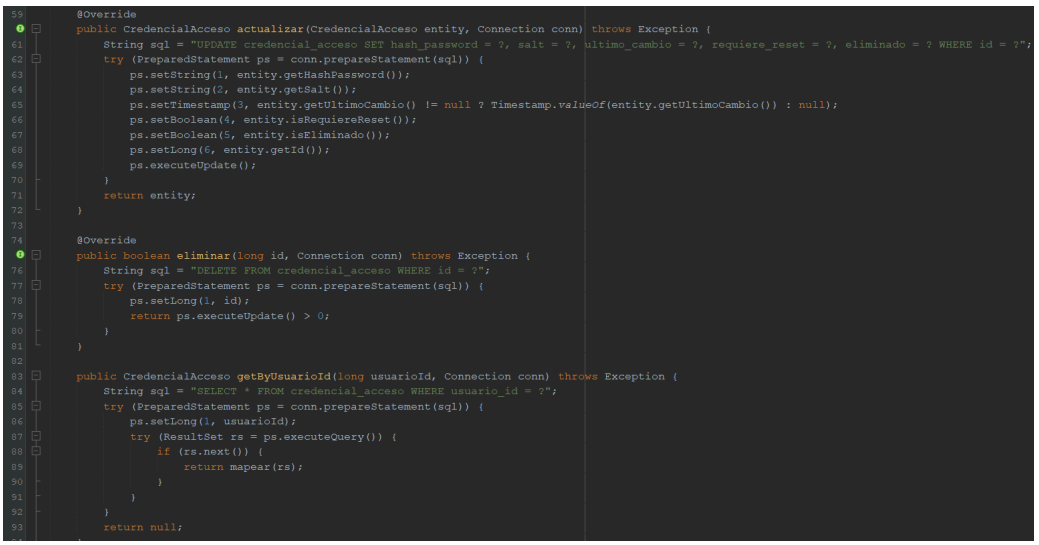
GenericDao.JAVA



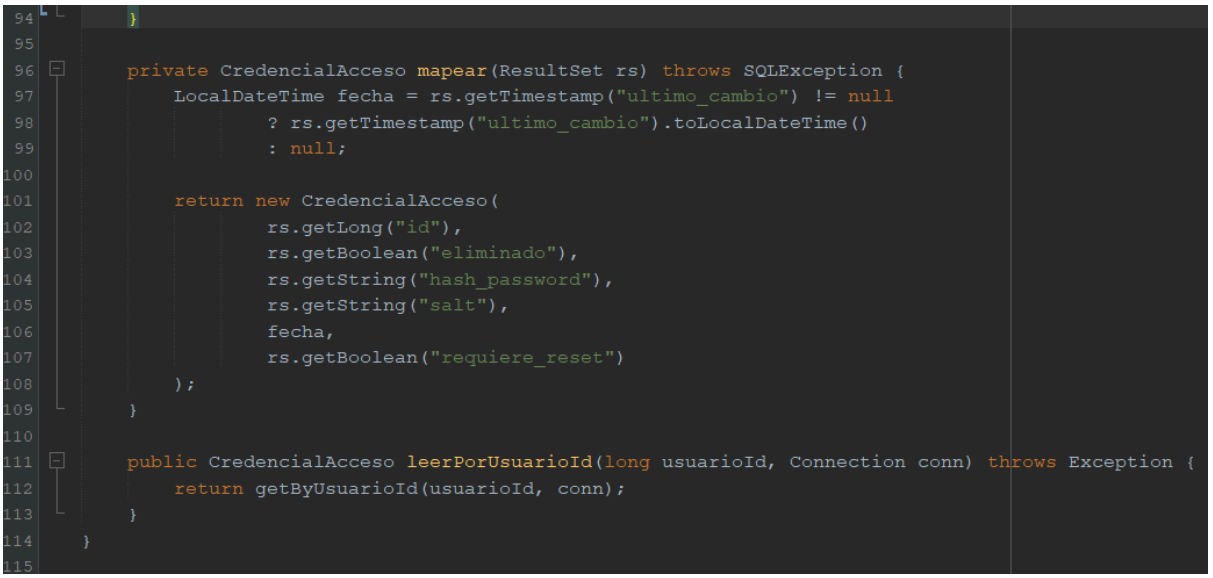
CredencialAccesoDAOImpl.JAVA




```java
    @Override
    public CredencialAcceso leer(long id, Connection conn) throws Exception {
        String sql = "SELECT * FROM credencial_acceso WHERE id = ?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setLong(1, id);
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    return mapear(rs);
                }
            }
        }
        return null;
    }

    @Override
    public List<CredencialAcceso> leerTodos(Connection conn) throws Exception {
        String sql = "SELECT * FROM credencial_acceso";
        List<CredencialAcceso> lista = new ArrayList<>();
        try (PreparedStatement ps = conn.prepareStatement(sql);
             ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                lista.add(mapear(rs));
            }
        }
        return lista;
    }
```



```java
    @Override
    public CredencialAcceso actualizar(CredencialAcceso entity, Connection conn) throws Exception {
        String sql = "UPDATE credencial_acceso SET hash_password = ?, salt = ?, ultimo_cambio = ?, requiere_reset = ?, eliminado = ? WHERE id = ?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setString(1, entity.getHashPassword());
            ps.setString(2, entity.getSalt());
            ps.setTimestamp(3, entity.getUltimoCambio() != null ? Timestamp.valueOf(entity.getUltimoCambio()) : null);
            ps.setBoolean(4, entity.isRequiereReset());
            ps.setBoolean(5, entity.isEliminado());
            ps.setLong(6, entity.getId());
            ps.executeUpdate();
        }
        return entity;
    }

    @Override
    public boolean eliminar(long id, Connection conn) throws Exception {
        String sql = "DELETE FROM credencial_acceso WHERE id = ?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setLong(1, id);
            return ps.executeUpdate() > 0;
        }
    }

    public CredencialAcceso getByUsuarioId(long usuarioId, Connection conn) throws Exception {
        String sql = "SELECT * FROM credencial_acceso WHERE usuario_id = ?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setLong(1, usuarioId);
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    return mapear(rs);
                }
            }
        }
        return null;
    }
```



```java
    }

    private CredencialAcceso mapear(ResultSet rs) throws SQLException {
        LocalDateTime fecha = rs.getTimestamp("ultimo_cambio") != null
                ? rs.getTimestamp("ultimo_cambio").toLocalDateTime()
                : null;

        return new CredencialAcceso(
                rs.getLong("id"),
                rs.getBoolean("eliminado"),
                rs.getString("hash_password"),
                rs.getString("salt"),
                fecha,
                rs.getBoolean("requiere_reset")
        );
    }

    public CredencialAcceso leerPorUsuarioId(long usuarioId, Connection conn) throws Exception {
        return getByUsuarioId(usuarioId, conn);
    }
}
```

Ejecución NETBEANS

MENÚ PRINCIPAL

```
===== MENU PRINCIPAL =====
1) Usuarios
2) Credenciales
0) Salir
Elija opci▯n:
```

CREAR USUARIO

```
===== MENU PRINCIPAL =====
1) Usuarios
2) Credenciales
0) Salir
Elija opci▯n: 1

--- MENU USUARIOS ---
1) Crear usuario
2) Buscar por ID
3) Listar todos
4) Actualizar
5) Eliminar
6) Buscar por Username
0) Volver
Opci▯n: 1
Username: EJEMPLO
Email: EJEMPLO@gmail.com
Hash password: 010101
? Usuario creado con ▯xito
```

BUSCAR POR ID

```
--- MENU USUARIOS ---
1) Crear usuario
2) Buscar por ID
3) Listar todos
4) Actualizar
5) Eliminar
6) Buscar por Username
0) Volver
Opci▯n: 2
ID: 1
Usuario{id=1, eliminado=false, username='mili', email='mili@gmail.com', activo=true, fechaRegistro=2025-11-19
```

LISTAR TODOS

```
--- MENU USUARIOS ---
1) Crear usuario
2) Buscar por ID
3) Listar todos
4) Actualizar
5) Eliminar
6) Buscar por Username
0) Volver
Opci▯n: 3
Usuario{id=1, eliminado=false, username='mili', email='mili@gmail.com', activo=true, fechaRegistro=2025-11-19
Usuario{id=2, eliminado=false, username='rocio', email='rocio_actualizado@gmail.com', activo=true, fechaRegis
Usuario{id=4, eliminado=false, username='EJEMPLO', email='EJEMPLO@gmail.com', activo=true, fechaRegistro=2025
```

ACTUALIZAR

```
--- MENU USUARIOS ---
1) Crear usuario
2) Buscar por ID
3) Listar todos
4) Actualizar
5) Eliminar
6) Buscar por Username
0) Volver
Opci▯n: 4
ID del usuario a actualizar: 1
Nuevo email: Milagritos@gmail.com
? Usuario actualizado.
```

ELIMINAR

```
--- MENU USUARIOS ---
1) Crear usuario
2) Buscar por ID
3) Listar todos
4) Actualizar
5) Eliminar
6) Buscar por Username
0) Volver
Opci▯n: 5
ID del usuario: 1
? Eliminado.
```

BUSCAR POR USERNAME

```
--- MENU USUARIOS ---
1) Crear usuario
2) Buscar por ID
3) Listar todos
4) Actualizar
5) Eliminar
6) Buscar por Username
0) Volver
Opci◆n: 6
Username: EJEMPLO
Usuario{id=4, eliminado=false, username='EJEMPLO', email='EJEMPLO@gmail.com', activo=true, fechaRegistro=2025
```

VOLVER

```
--- MENU USUARIOS ---
1) Crear usuario
2) Buscar por ID
3) Listar todos
4) Actualizar
5) Eliminar
6) Buscar por Username
0) Volver
Opci◆n: 0

===== MENU PRINCIPAL =====
1) Usuarios
2) Credenciales
0) Salir
Elija opci◆n:
```

CREAR CREDENCIAL

```
--- MENU CREDENCIALES ---
1) Crear credencial
2) Buscar por ID
3) Listar todas
4) Actualizar
5) Eliminar
0) Volver
Opci◆n: 1
Hash password: 121212
? Credencial creada con ◆xito, ID: 12
```

BUSCAR POR ID

```
--- MENU CREDENCIALES ---
1) Crear credencial
2) Buscar por ID
3) Listar todas
4) Actualizar
5) Eliminar
0) Volver
Opci◆n: 2
ID: 3
CredencialAcceso{id=3, eliminado=false, hashPassword='020202', salt='salt-1763533091622', ultimoCambio=2025-1
```

LISTAR TODAS

```
--- MENU CREDENCIALES ---
1) Crear credencial
2) Buscar por ID
3) Listar todas
4) Actualizar
5) Eliminar
0) Volver
Opci◆n: 3
CredencialAcceso{id=1, eliminado=false, hashPassword='010101', salt='salt-1763533074968', ultimoCambio=2025-1
CredencialAcceso{id=3, eliminado=false, hashPassword='020202', salt='salt-1763533091622', ultimoCambio=2025-1
CredencialAcceso{id=4, eliminado=false, hashPassword='020202', salt='salt-1763533091622', ultimoCambio=2025-1
CredencialAcceso{id=5, eliminado=false, hashPassword='030303', salt='salt-1763533105019', ultimoCambio=2025-1
CredencialAcceso{id=8, eliminado=false, hashPassword='050505', salt='salt-1763533250467', ultimoCambio=2025-1
CredencialAcceso{id=9, eliminado=false, hashPassword='090909', salt='salt-1763533256162', ultimoCambio=2025-1
CredencialAcceso{id=10, eliminado=false, hashPassword='010101', salt='salt-1763537663026', ultimoCambio=2025-
CredencialAcceso{id=11, eliminado=false, hashPassword='010101', salt='salt-1763537663026', ultimoCambio=2025-
CredencialAcceso{id=12, eliminado=false, hashPassword='121212', salt='salt-1763538098333', ultimoCambio=2025-
```

ACTUALIZAR

```
--- MENU CREDENCIALES ---
1) Crear credencial
2) Buscar por ID
3) Listar todas
4) Actualizar
5) Eliminar
0) Volver
Opci◆n: 4
ID de la credencial a actualizar: 3
Nuevo hash password: 212121
? Credencial actualizada.
```

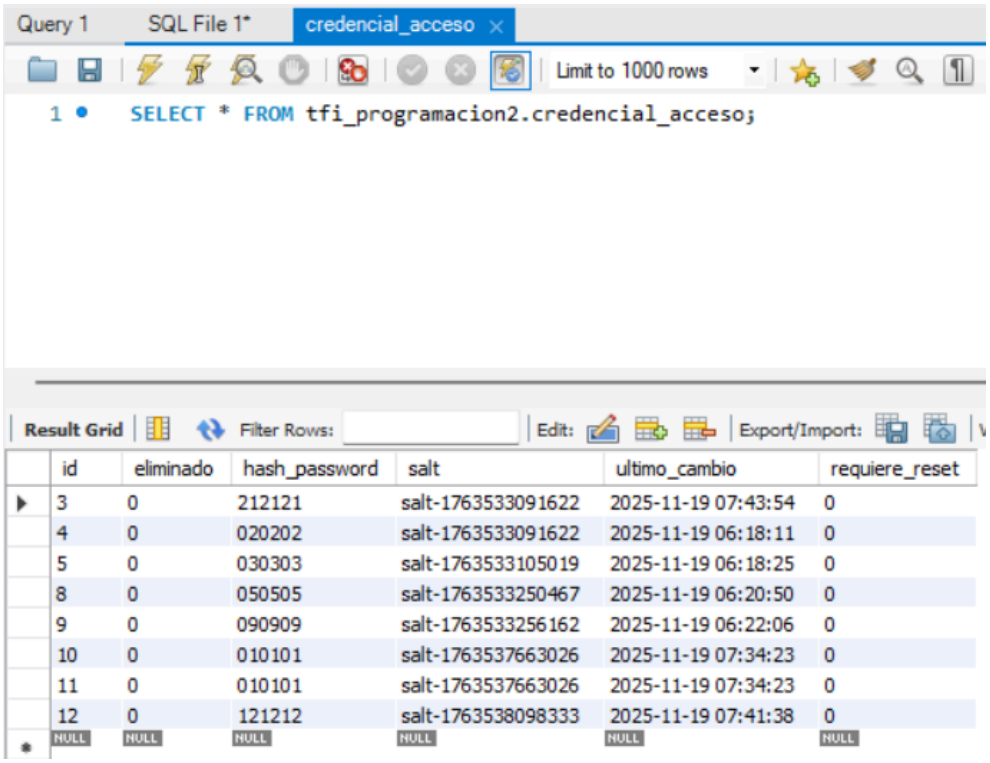
ELIMINAR

```
--- MENU CREDENCIALES ---
1) Crear credencial
2) Buscar por ID
3) Listar todas
4) Actualizar
5) Eliminar
0) Volver
Opci?n: 5
ID de la credencial: 1
? Credencial eliminada.
```

SALIR

```
===== MENU PRINCIPAL =====
1) Usuarios
2) Credenciales
0) Salir
Elija opci?n: 0
?Saliendo!
```

Tabla CREDENCIAL_ACCESO



id	eliminado	hash_password	salt	ultimo_cambio	requiere_reset
3	0	212121	salt-1763533091622	2025-11-19 07:43:54	0
4	0	020202	salt-1763533091622	2025-11-19 06:18:11	0
5	0	030303	salt-1763533105019	2025-11-19 06:18:25	0
8	0	050505	salt-1763533250467	2025-11-19 06:20:50	0
9	0	090909	salt-1763533256162	2025-11-19 06:22:06	0
10	0	010101	salt-1763537663026	2025-11-19 07:34:23	0
11	0	010101	salt-1763537663026	2025-11-19 07:34:23	0
12	0	121212	salt-1763538098333	2025-11-19 07:41:38	0
NULL	NULL	NULL	NULL	NULL	NULL

Tabla USUARIO