

# 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

<b>Integrante</b>	<b>Comisión</b>	<b>Rol en el proyecto</b>
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. Elección del dominio y justificación

Para el desarrollo de la aplicación se eligió el dominio Usuario  $\rightarrow$  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  $1 \rightarrow 1$  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  $1 \rightarrow 1$  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

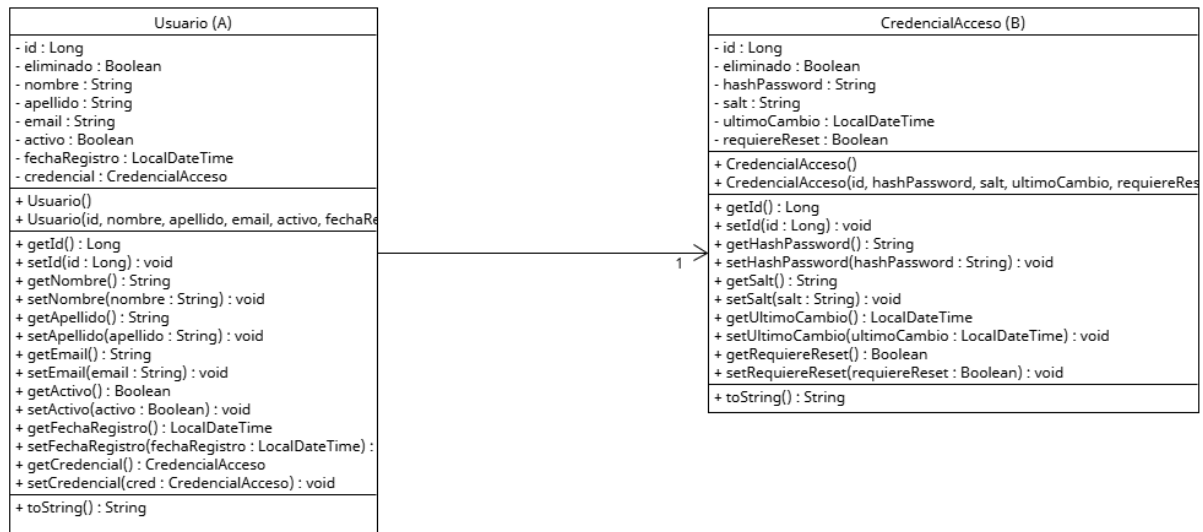
El diagrama refleja la estructura de las clases y su relación:

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  $1 \rightarrow 1$  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:

```
1 package Service;
2
3 import java.util.List;
4
5 public interface GenericService<T> {
6
7     T insertar(T entity) throws Exception;
8
9     T actualizar(T entity) throws Exception;
10
11     boolean eliminar(long id) throws Exception;
12
13     T getById(long id) throws Exception;
14
15     List<T> getAll() throws Exception;
16 }
```

### 1. Servicio de Credencial Acceso

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:

```

14  @Override
15  public CredencialAcceso insertar(CredencialAcceso cred) throws Exception {
16      if (cred == null) {
17          throw new IllegalArgumentException("La credencial no puede ser null.");
18      }
19
20      try (Connection conn = DatabaseConfig.getConnection()) {
21          conn.setAutoCommit(false);
22          try {
23              validar(cred);
24              CredencialAcceso creada = credDao.crear(cred, conn);
25              conn.commit();
26              return creada;
27          } catch (Exception e) {
28              conn.rollback();
29              throw new Exception("Error al insertar la credencial: " + e.getMessage(), e);
30          }
31      }
32  }

```

## 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.

```

// 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  $1 \rightarrow 1$ .

## 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:

```

80 private void crearUsuario() throws Exception {
81
82     Usuario u = new Usuario();
83
84     System.out.print("Username: ");
85     u.setUsername(sc.nextLine());
86
87     System.out.print("Email: ");
88     u.setEmail(sc.nextLine());
89
90     u.setActivo(true);
91     u.setFechaRegistro(LocalDate.now());
92     u.setEliminado(false);
93
94     // credencial asociada
95     CredencialAcceso c = new CredencialAcceso();
96     System.out.print("Hash password: ");
97     c.setHashPassword(sc.nextLine());
98     c.setSalt("salt-" + System.currentTimeMillis());
99     c.setUltimoCambio(LocalDate.now());
00     c.setRequiereReset(false);
01     c.setEliminado(false);
02
03     u.setCredencial(c);
04
05     usuarioService.insertar(u);
06
07     System.out.println("✓ Usuario creado con éxito");
08 }

```

### 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:

```

247 private void actualizarCredencial() throws Exception {
248     System.out.print("ID de la credencial a actualizar: ");
249     long id = Long.parseLong(sc.nextLine());
250
251     CredencialAcceso c = credService.getById(id);
252
253     if (c == null) {
254         System.out.println("Credencial no encontrada.");
255         return;
256     }
257
258     System.out.print("Nuevo hash password: ");
259     c.setHashPassword(sc.nextLine());
260     c.setUltimoCambio(LocalDate.now());
261
262     credService.actualizar(c);
263
264     System.out.println("✓ Credencial actualizada.");
265 }

```

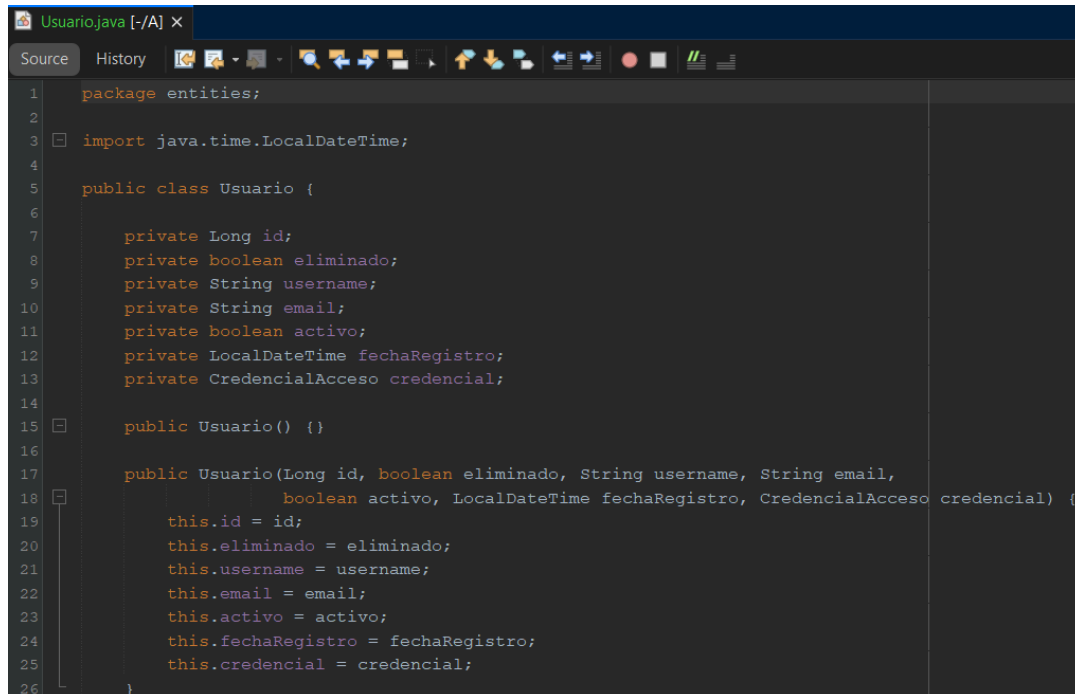
#### 4. Manejo de errores y robustez

- 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.

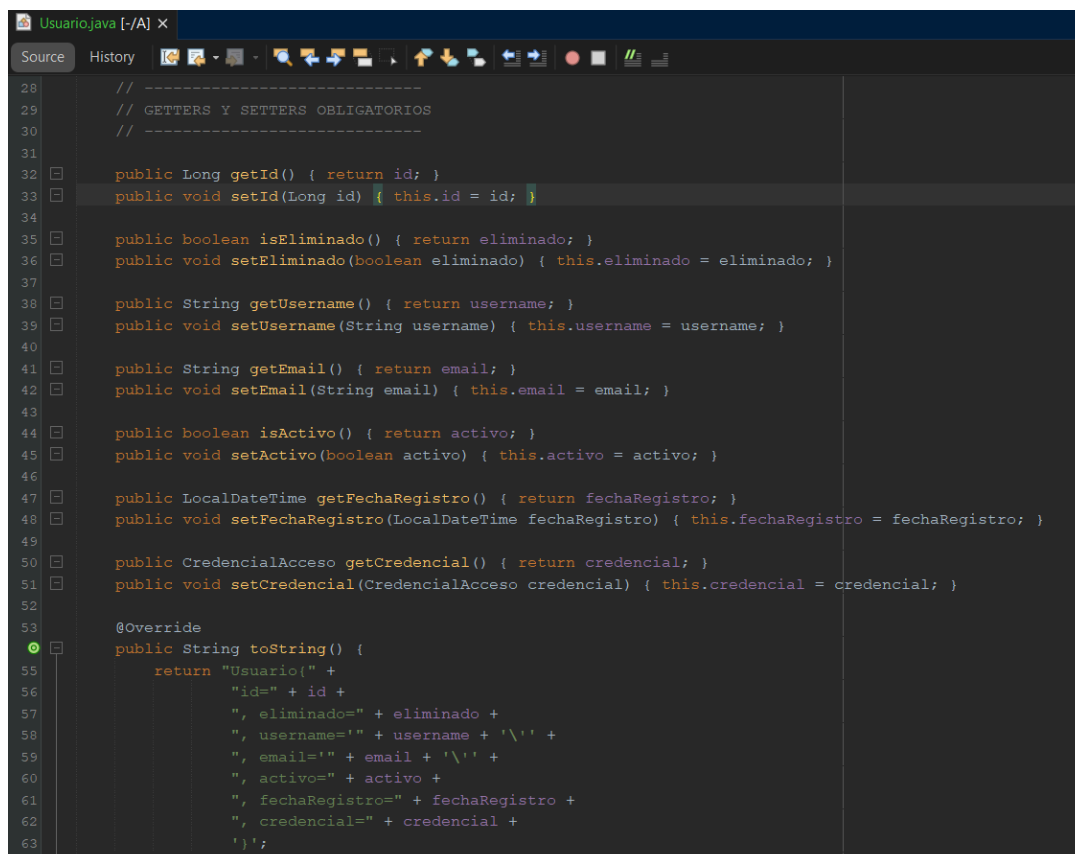
## Capturas de pantalla

- Package ENTITIES

## Usuario.JAVA



```
1 package entities;
2
3 import java.time.LocalDateTime;
4
5 public class Usuario {
6
7     private Long id;
8     private boolean eliminado;
9     private String username;
10    private String email;
11    private boolean activo;
12    private LocalDateTime fechaRegistro;
13    private CredencialAcceso credencial;
14
15    public Usuario() {}
16
17    public Usuario(Long id, boolean eliminado, String username, String email,
18        boolean activo, LocalDateTime fechaRegistro, CredencialAcceso credencial) {
19        this.id = id;
20        this.eliminado = eliminado;
21        this.username = username;
22        this.email = email;
23        this.activo = activo;
24        this.fechaRegistro = fechaRegistro;
25        this.credencial = credencial;
26    }
```



```
28 // -----
29 // GETTERS Y SETTERS OBLIGATORIOS
30 // -----
31
32 public Long getId() { return id; }
33 public void setId(Long id) { this.id = id; }
34
35 public boolean isEliminado() { return eliminado; }
36 public void setEliminado(boolean eliminado) { this.eliminado = eliminado; }
37
38 public String getUsername() { return username; }
39 public void setUsername(String username) { this.username = username; }
40
41 public String getEmail() { return email; }
42 public void setEmail(String email) { this.email = email; }
43
44 public boolean isActive() { return activo; }
45 public void setActive(boolean activo) { this.activo = activo; }
46
47 public LocalDateTime getFechaRegistro() { return fechaRegistro; }
48 public void setFechaRegistro(LocalDateTime fechaRegistro) { this.fechaRegistro = fechaRegistro; }
49
50 public CredencialAcceso getCredencial() { return credencial; }
51 public void setCredencial(CredencialAcceso credencial) { this.credencial = credencial; }
52
53 @Override
54 public String toString() {
55     return "Usuario{" +
56         "id=" + id +
57         ", eliminado=" + eliminado +
58         ", username='" + username + '\'' +
59         ", email='" + email + '\'' +
60         ", activo=" + activo +
61         ", fechaRegistro=" + fechaRegistro +
62         ", credencial=" + credencial +
63         '}';
64 }
```

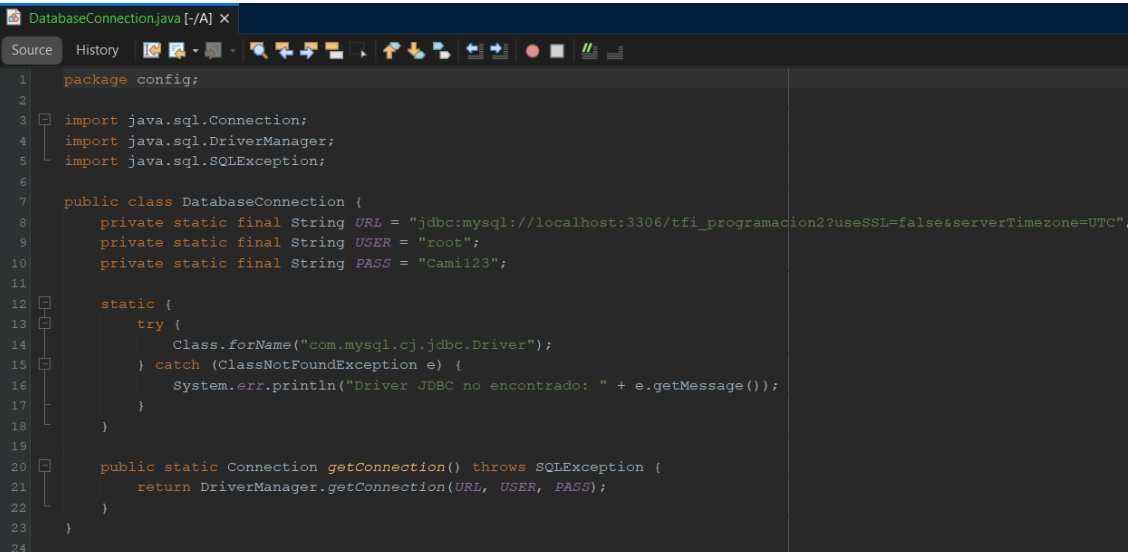
## CredencialAcceso.JAVA

```
CredencialAcceso.java [-/A] x
Source History
1 package entities;
2
3 import java.time.LocalDateTime;
4
5 public class CredencialAcceso {
6
7     private Long id;
8     private boolean eliminado;
9     private String hashPassword;
10    private String salt;
11    private LocalDateTime ultimoCambio;
12    private boolean requiereReset;
13    private Long usuarioId; // Nueva propiedad para relacionar con Usuario
14
15    public CredencialAcceso() {}
16
17    public CredencialAcceso(Long id, boolean eliminado, String hashPassword, String salt,
18    LocalDateTime ultimoCambio, boolean requiereReset) {
19
20        this.id = id;
21        this.eliminado = eliminado;
22        this.hashPassword = hashPassword;
23        this.salt = salt;
24        this.ultimoCambio = ultimoCambio;
25        this.requiereReset = requiereReset;
26    }
27
28    // -----
29    // GETTERS Y SETTERS
30    // -----
31    public Long getId() { return id; }
32    public void setId(Long id) { this.id = id; }
33
34    public boolean isEliminado() { return eliminado; }
35    public void setEliminado(boolean eliminado) { this.eliminado = eliminado; }
36
37    public String getHashPassword() { return hashPassword; }
38    public void setHashPassword(String hashPassword) { this.hashPassword = hashPassword; }
39
40    public String getSalt() { return salt; }
41    public void setSalt(String salt) { this.salt = salt; }
42
43    public LocalDateTime getUltimoCambio() { return ultimoCambio; }
44    public void setUltimoCambio(LocalDateTime ultimoCambio) { this.ultimoCambio = ultimoCambio; }
45
46    public boolean isRequiereReset() { return requiereReset; }
47    public void setRequiereReset(boolean requiereReset) { this.requiereReset = requiereReset; }
48
49    public Long getUsuarioId() { return usuarioId; }
50    public void setUsuarioId(Long usuarioId) { this.usuarioId = usuarioId; }
51
52    @Override
53    public String toString() {
54        return "CredencialAcceso{" +
55            "id=" + id +
56            ", eliminado=" + eliminado +
57            ", hashPassword='" + hashPassword + '\'' +
58            ", salt='" + salt + '\'' +
59            ", ultimoCambio=" + ultimoCambio +
60            ", requiereReset=" + requiereReset +
61            ", usuarioId=" + usuarioId +
62            '}';
63    }
64 }
```

```
CredencialAcceso.java [-/A] x
Source History
27 // -----
28 // GETTERS Y SETTERS
29 // -----
30 public Long getId() { return id; }
31 public void setId(Long id) { this.id = id; }
32
33 public boolean isEliminado() { return eliminado; }
34 public void setEliminado(boolean eliminado) { this.eliminado = eliminado; }
35
36 public String getHashPassword() { return hashPassword; }
37 public void setHashPassword(String hashPassword) { this.hashPassword = hashPassword; }
38
39 public String getSalt() { return salt; }
40 public void setSalt(String salt) { this.salt = salt; }
41
42 public LocalDateTime getUltimoCambio() { return ultimoCambio; }
43 public void setUltimoCambio(LocalDateTime ultimoCambio) { this.ultimoCambio = ultimoCambio; }
44
45 public boolean isRequiereReset() { return requiereReset; }
46 public void setRequiereReset(boolean requiereReset) { this.requiereReset = requiereReset; }
47
48 public Long getUsuarioId() { return usuarioId; }
49 public void setUsuarioId(Long usuarioId) { this.usuarioId = usuarioId; }
50
51 @Override
52 public String toString() {
53     return "CredencialAcceso{" +
54         "id=" + id +
55         ", eliminado=" + eliminado +
56         ", hashPassword='" + hashPassword + '\'' +
57         ", salt='" + salt + '\'' +
58         ", ultimoCambio=" + ultimoCambio +
59         ", requiereReset=" + requiereReset +
60         ", usuarioId=" + usuarioId +
61         '}';
62 }
63 }
```

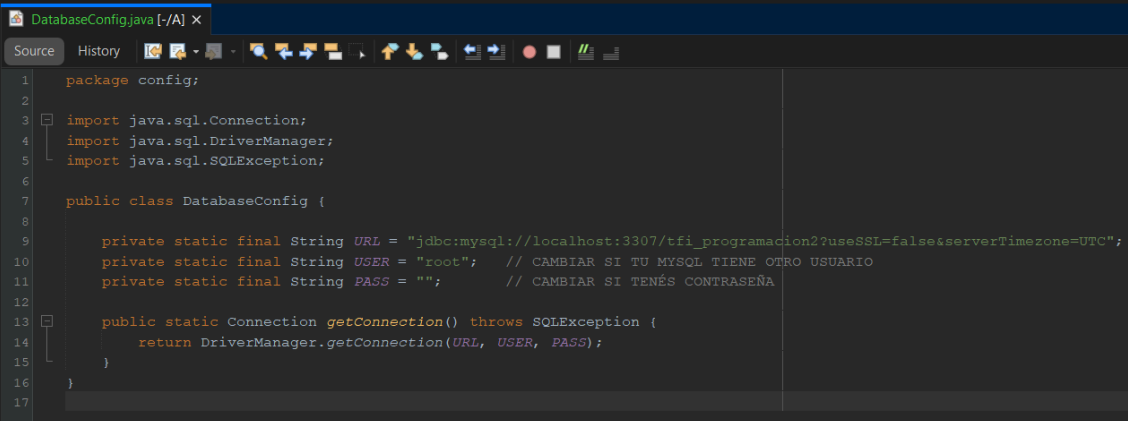
- Package CONFIG

## DatabaseConnection.JAVA



```
1 package config;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class DatabaseConnection {
8     private static final String URL = "jdbc:mysql://localhost:3306/tfi_programacion2?useSSL=false&serverTimezone=UTC";
9     private static final String USER = "root";
10    private static final String PASS = "Camil23";
11
12    static {
13        try {
14            Class.forName("com.mysql.cj.jdbc.Driver");
15        } catch (ClassNotFoundException e) {
16            System.err.println("Driver JDBC no encontrado: " + e.getMessage());
17        }
18    }
19
20    public static Connection getConnection() throws SQLException {
21        return DriverManager.getConnection(URL, USER, PASS);
22    }
23 }
24
```

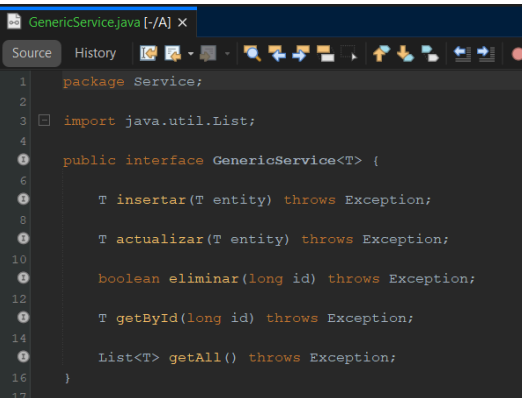
## DatabaseConfig.JAVA



```
1 package config;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class DatabaseConfig {
8
9     private static final String URL = "jdbc:mysql://localhost:3307/tfi_programacion2?useSSL=false&serverTimezone=UTC";
10    private static final String USER = "root"; // CAMBIAR SI TU MYSQL TIENE OTRO USUARIO
11    private static final String PASS = ""; // CAMBIAR SI TENÉS CONTRASEÑA
12
13    public static Connection getConnection() throws SQLException {
14        return DriverManager.getConnection(URL, USER, PASS);
15    }
16 }
17
```

- Package SERVICE

## GenericService.JAVA



```
1 package Service;
2
3 import java.util.List;
4
5 public interface GenericService<T> {
6
7     T insertar(T entity) throws Exception;
8
9     T actualizar(T entity) throws Exception;
10
11     boolean eliminar(long id) throws Exception;
12
13     T getById(long id) throws Exception;
14
15     List<T> getAll() throws Exception;
16 }
17
```

## UsuarioService.JAVA

```
1 package Service;
2
3 import config.DatabaseConfig;
4 import dao.UsuarioDAOImpl;
5 import dao.CredencialAccesoDAOImpl;
6 import entities.Usuario;
7
8 import java.sql.Connection;
9 import java.util.List;
10
11 public class UsuarioService implements GenericService<Usuario> {
12
13     private final UsuarioDAOImpl usuarioDao = new UsuarioDAOImpl();
14     private final CredencialAccesoDAOImpl credDao = new CredencialAccesoDAOImpl();
15
16     @Override
17     public Usuario insertar(Usuario usuario) throws Exception {
18
19         try (Connection conn = DatabaseConfig.getConnection()) {
20             conn.setAutoCommit(false);
21
22             validar(usuario);
23
24             // 1 -> Primero crear credencial
25             if (usuario.getCredencial() != null) {
26                 credDao.crear(usuario.getCredencial(), conn);
27             }
28
29             // 2 -> Después crear usuario
30             Usuario creado = usuarioDao.crear(usuario, conn);
31
32             conn.commit();
33             return creado;
34         } catch (Exception e) {
35             throw new Exception("Error al insertar usuario: " + e.getMessage());
36         }
37     }
38 }
```

```
40 @Override
41 public Usuario actualizar(Usuario usuario) throws Exception {
42     try (Connection conn = DatabaseConfig.getConnection()) {
43         conn.setAutoCommit(false);
44
45         validar(usuario);
46
47         if (usuario.getCredencial() != null) {
48             credDao.actualizar(usuario.getCredencial(), conn);
49         }
50
51         Usuario actualizado = usuarioDao.actualizar(usuario, conn);
52
53         conn.commit();
54         return actualizado;
55     }
56 }
57
58 @Override
59 public boolean eliminar(long id) throws Exception {
60     try (Connection conn = DatabaseConfig.getConnection()) {
61         conn.setAutoCommit(false);
62
63         boolean ok = usuarioDao.eliminar(id, conn);
64
65         conn.commit();
66         return ok;
67     } catch (Exception e) {
68         throw new Exception("Error eliminando usuario: " + e.getMessage());
69     }
70 }
71 }
```

```

73      @Override
74      public Usuario getById(long id) throws Exception {
75          try (Connection conn = DatabaseConfig.getConnection()) {
76              return usuarioDao.leer(id, conn);
77          }
78      }
79
80      @Override
81      public List<Usuario> getAll() throws Exception {
82          try (Connection conn = DatabaseConfig.getConnection()) {
83              return usuarioDao.leerTodos(conn);
84          }
85      }
86
87      private void validar(Usuario u) throws Exception {
88          if (u.getUsername() == null || u.getUsername().isBlank())
89              throw new Exception("El username no puede estar vacío.");
90
91          if (u.getEmail() == null || u.getEmail().isBlank())
92              throw new Exception("El email es obligatorio.");
93
94          if (!u.getEmail().contains("@"))
95              throw new Exception("Formato de email inválido.");
96      }
97  }
98

```

## CredencialAccesoService.JAVA

```

CredencialAccesoService.java [-/A] x
Source History
1  package Service;
2
3  import config.DatabaseConfig;
4  import dao.CredencialAccesoDAOImpl;
5  import entities.CredencialAcceso;
6
7  import java.sql.Connection;
8  import java.util.List;
9
10 public class CredencialAccesoService implements GenericService<CredencialAcceso> {
11
12     private final CredencialAccesoDAOImpl credDao = new CredencialAccesoDAOImpl();
13
14     @Override
15     public CredencialAcceso insertar(CredencialAcceso cred) throws Exception {
16         if (cred == null) {
17             throw new IllegalArgumentException("La credencial no puede ser null.");
18         }
19
20         try (Connection conn = DatabaseConfig.getConnection()) {
21             conn.setAutoCommit(false);
22             try {
23                 validar(cred);
24                 CredencialAcceso creada = credDao.crear(cred, conn);
25                 conn.commit();
26                 return creada;
27             } catch (Exception e) {
28                 conn.rollback();
29                 throw new Exception("Error al insertar la credencial: " + e.getMessage(), e);
30             }
31         }
32     }
33

```



```

CredencialAccesoService.java [-/A] x
Source History
34 @Override
35 public CredencialAcceso actualizar(CredencialAcceso cred) throws Exception {
36     if (cred == null) {
37         throw new IllegalArgumentException("La credencial no puede ser null.");
38     }
39
40     try (Connection conn = DatabaseConfig.getConnection()) {
41         conn.setAutoCommit(false);
42         try {
43             validar(cred);
44             CredencialAcceso updated = credDao.actualizar(cred, conn);
45             conn.commit();
46             return updated;
47         } catch (Exception e) {
48             conn.rollback();
49             throw new Exception("Error al actualizar la credencial: " + e.getMessage(), e);
50         }
51     }
52 }
53
54 @Override
55 public boolean eliminar(long id) throws Exception {
56     try (Connection conn = DatabaseConfig.getConnection()) {
57         conn.setAutoCommit(false);
58         try {
59             boolean ok = credDao.eliminar(id, conn);
60             conn.commit();
61             return ok;
62         } catch (Exception e) {
63             conn.rollback();
64             throw new Exception("Error al eliminar la credencial: " + e.getMessage(), e);
65         }
66     }
67 }

```

```

69 @Override
70 public CredencialAcceso getById(long id) throws Exception {
71     try (Connection conn = DatabaseConfig.getConnection()) {
72         return credDao.leer(id, conn);
73     }
74 }
75
76 @Override
77 public List<CredencialAcceso> getAll() throws Exception {
78     try (Connection conn = DatabaseConfig.getConnection()) {
79         return credDao.leerTodos(conn);
80     }
81 }
82
83 private void validar(CredencialAcceso cred) throws Exception {
84     if (cred.getHashPassword() == null || cred.getHashPassword().isBlank())
85         throw new Exception("La contraseña hash no puede ser vacía.");
86
87     if (cred.getSalt() == null || cred.getSalt().isBlank())
88         throw new Exception("El salt es obligatorio.");
89 }
90 }
91

```

- Package MAIN

## TestUsuarioDAO.JAVA

```
TestUsuarioDAO.java [-/A] X
Source History
1 package main;
2
3 import config.DatabaseConfig;
4 import dao.UsuarioDAOImpl;
5 import dao.CredencialAccesoDAOImpl;
6 import entities.Usuario;
7 import entities.CredencialAcceso;
8
9 import java.sql.Connection;
10 import java.time.LocalDateTime;
11
12 public class TestUsuarioDAO {
13
14     public static void main(String[] args) {
15         // Paso 1: Declaramos la conexión
16         Connection conn = null;
17
18         try {
19             // Paso 2: Obtenemos la conexión desde DatabaseConfig
20             conn = DatabaseConfig.getConnection();
21             // Paso 3: Iniciamos una transacción
22             conn.setAutoCommit(false);
23
24             // Paso 4: Instanciamos los DAOs
25             CredencialAccesoDAOImpl credDao = new CredencialAccesoDAOImpl();
26             UsuarioDAOImpl usuarioDao = new UsuarioDAOImpl();
27
28             // -----
29             // Paso 5: Creamos una CredencialAcceso
30             CredencialAcceso cred = new CredencialAcceso();
31             cred.setHashPassword("miHash123");
32             cred.setSalt("saltsito");
33             cred.setUltimoCambio(LocalDateTime.now());
34             cred.setRequiereReset(false);
35             cred.setEliminado(false);
```

```
TestUsuarioDAO.java [-/A] X
Source History
37         // Paso 6: Creamos un Usuario usando esa credencial
38         Usuario user = new Usuario();
39         user.setUsername("yamila123");
40         user.setEmail("yamila@example.com");
41         user.setActivo(true);
42         user.setFechaRegistro(LocalDateTime.now());
43         user.setEliminado(false);
44         user.setCredencial(cred);
45
46         // -----
47         // Paso 7: Guardamos el usuario (esto también guardará la credencial)
48         usuarioDao.crear(user, conn);
49
50         // Paso 8: Commit de la transacción
51         conn.commit();
52         System.out.println("✓ Usuario y Credencial creados correctamente.");
53         System.out.println(user);
54
55         // -----
56         // Paso 9: Leer usuario desde la DB para verificar
57         Usuario userLeido = usuarioDao.leer(user.getId(), conn);
58         System.out.println("✓ Usuario leído desde la base.");
59         System.out.println(userLeido);
60
61     } catch (Exception e) {
62         // Paso 10: Si algo falla, hacemos rollback
63         try {
64             if (conn != null) conn.rollback();
65         } catch (Exception ex) {
66             ex.printStackTrace();
67         }
68         e.printStackTrace();
69     } finally {
```

```

70 // Paso 11: Cerramos la conexión
71 try {
72     if (conn != null) conn.close();
73 } catch (Exception e) {
74     e.printStackTrace();
75 }
76
77 }
78

```

## TestConexion.Java

```

TestConexion.java [-/A] x
Source History
1 package main;
2
3 import config.DatabaseConfig;
4 import java.sql.Connection;
5
6 public class TestConexion {
7     public static void main(String[] args) {
8         try (Connection conn = DatabaseConfig.getConnection()) {
9             System.out.println("✓ Conexión exitosa a la base de datos!");
10        } catch (Exception e) {
11            System.out.println("✗ Error de conexión:");
12            e.printStackTrace();
13        }
14    }
15 }

```

## AppMenu.JAVA

```

AppMenu.java [-/A] x
Source History
1 package main;
2
3 import entities.Usuario;
4 import entities.CredencialAcceso;
5 import Service.UsuarioService;
6 import Service.CredencialAccesoService;
7
8 import java.time.LocalDateTime;
9 import java.util.List;
10 import java.util.Scanner;
11
12 public class AppMenu {
13
14     private final Scanner sc = new Scanner(System.in);
15     private final UsuarioService usuarioService = new UsuarioService();
16     private final CredencialAccesoService credService = new CredencialAccesoService();
17
18     public void iniciar() {
19
20         String op;
21
22         do {
23             System.out.println("\n===== MENU PRINCIPAL =====");
24             System.out.println("1) Usuarios");
25             System.out.println("2) Credenciales");
26             System.out.println("0) Salir");
27             System.out.print("Elija opción: ");
28
29             op = sc.nextLine().trim();
30
31             switch (op) {
32                 case "1" -> menuUsuarios();
33                 case "2" -> menuCredenciales();
34                 case "0" -> System.out.println("¡Saliendo!");
35                 default -> System.out.println("Opción inválida.");
36             }
37         } while (op != "0");
38     }
39 }

```

```

38         } while (!op.equals("0"));
39     }
40
41     // =====
42     //          USUARIOS
43     // =====
44
45     private void menuUsuarios() {
46         String op;
47
48         do {
49             System.out.println("\n--- MENU USUARIOS ---");
50             System.out.println("1) Crear usuario");
51             System.out.println("2) Buscar por ID");
52             System.out.println("3) Listar todos");
53             System.out.println("4) Actualizar");
54             System.out.println("5) Eliminar");
55             System.out.println("6) Buscar por Username");
56             System.out.println("0) Volver");
57             System.out.print("Opción: ");
58
59             op = sc.nextLine();
60
61             try {
62                 switch (op) {
63                     case "1" -> crearUsuario();
64                     case "2" -> leerUsuario();
65                     case "3" -> listarUsuarios();
66                     case "4" -> actualizarUsuario();
67                     case "5" -> eliminarUsuario();
68                     case "6" -> buscarPorUsername();
69                     case "0" -> {}
70                     default -> System.out.println("Opción inválida.");
71                 }

```

```

73             } catch (Exception e) {
74                 System.out.println("⚠ Error: " + e.getMessage());
75             }
76
77         } while (!op.equals("0"));
78     }
79
80     private void crearUsuario() throws Exception {
81
82         Usuario u = new Usuario();
83
84         System.out.print("Username: ");
85         u.setUsername(sc.nextLine());
86
87         System.out.print("Email: ");
88         u.setEmail(sc.nextLine());
89
90         u.setActivo(true);
91         u.setFechaRegistro(LocalDateDateTime.now());
92         u.setEliminado(false);
93
94         // credencial asociada
95         CredencialAcceso c = new CredencialAcceso();
96         System.out.print("Hash password: ");
97         c.setHashPassword(sc.nextLine());
98         c.setSalt("salt-" + System.currentTimeMillis());
99         c.setUltimoCambio(LocalDateDateTime.now());
100        c.setRequiereReset(false);
101        c.setEliminado(false);
102
103        u.setCredencial(c);
104
105        usuarioService.insertar(u);
106
107        System.out.println("✓ Usuario creado con éxito");
108    }

```

```

110 private void leerUsuario() throws Exception {
111     System.out.print("ID: ");
112     long id = Long.parseLong(sc.nextLine());
113
114     Usuario u = usuarioService.getById(id);
115
116     if (u == null)
117         System.out.println("No existe ese usuario.");
118     else
119         System.out.println(u);
120 }
121
122 private void listarUsuarios() throws Exception {
123     List<Usuario> lista = usuarioService.getAll();
124     lista.forEach(System.out::println);
125 }
126
127 private void actualizarUsuario() throws Exception {
128
129     System.out.print("ID del usuario a actualizar: ");
130     long id = Long.parseLong(sc.nextLine());
131
132     Usuario u = usuarioService.getById(id);
133
134     if (u == null) {
135         System.out.println("Usuario no encontrado.");
136         return;
137     }
138
139     System.out.print("Nuevo email: ");
140     u.setEmail(sc.nextLine());
141
142     usuarioService.actualizar(u);
143
144     System.out.println("✓ Usuario actualizado.");
145 }

```

```

147 private void eliminarUsuario() throws Exception {
148
149     System.out.print("ID del usuario: ");
150     long id = Long.parseLong(sc.nextLine());
151
152     if (usuarioService.eliminar(id))
153         System.out.println("✓ Eliminado.");
154     else
155         System.out.println("No existe ese ID.");
156 }
157
158 private void buscarPorUsername() throws Exception {
159
160     System.out.print("Username: ");
161     String username = sc.nextLine();
162
163     List<Usuario> all = usuarioService.getAll();
164
165     all.stream()
166         .filter(u -> u.getUsername().equalsIgnoreCase(username))
167         .findFirst()
168         .ifPresentOrElse(
169             System.out::println,
170             () -> System.out.println("No encontrado.")
171         );
172 }

```

```

174 // =====
175 // CREDENCIALES
176 // =====
177
178 private void menuCredenciales() {
179     String op;
180
181     do {
182         System.out.println("\n--- MENU CREDENCIALES ---");
183         System.out.println("1) Crear credencial");
184         System.out.println("2) Buscar por ID");
185         System.out.println("3) Listar todas");
186         System.out.println("4) Actualizar");
187         System.out.println("5) Eliminar");
188         System.out.println("0) Volver");
189         System.out.print("Opción: ");
190
191         op = sc.nextLine();
192
193         try {
194             switch (op) {
195                 case "1" -> crearCredencial();
196                 case "2" -> leerCredencial();
197                 case "3" -> listarCredenciales();
198                 case "4" -> actualizarCredencial();
199                 case "5" -> eliminarCredencial();
200                 case "0" -> {}
201                 default -> System.out.println("Opción inválida.");
202             }
203         } catch (Exception e) {
204             System.out.println("⚠ Error: " + e.getMessage());
205         }
206
207     } while (!op.equals("0"));
208 }

```

```

210 // =====
211 // Métodos CRUD de credenciales
212 // =====
213
214 private void crearCredencial() throws Exception {
215     CredencialAcceso c = new CredencialAcceso();
216
217     System.out.print("Hash password: ");
218     c.setHashPassword(sc.nextLine());
219
220     c.setSalt("salt-" + System.currentTimeMillis());
221     c.setUltimoCambio(LocalDateDateTime.now());
222     c.setRequiereReset(false);
223     c.setEliminado(false);
224
225     credService.insertar(c);
226
227     System.out.println("✓ Credencial creada con éxito, ID: " + c.getId());
228 }
229
230 private void leerCredencial() throws Exception {
231     System.out.print("ID: ");
232     long id = Long.parseLong(sc.nextLine());
233
234     CredencialAcceso c = credService.getById(id);
235
236     if (c == null)
237         System.out.println("No existe esa credencial.");
238     else
239         System.out.println(c);
240 }
241
242 private void listarCredenciales() throws Exception {
243     List<CredencialAcceso> lista = credService.getAll();
244     lista.forEach(System.out::println);
245 }
246

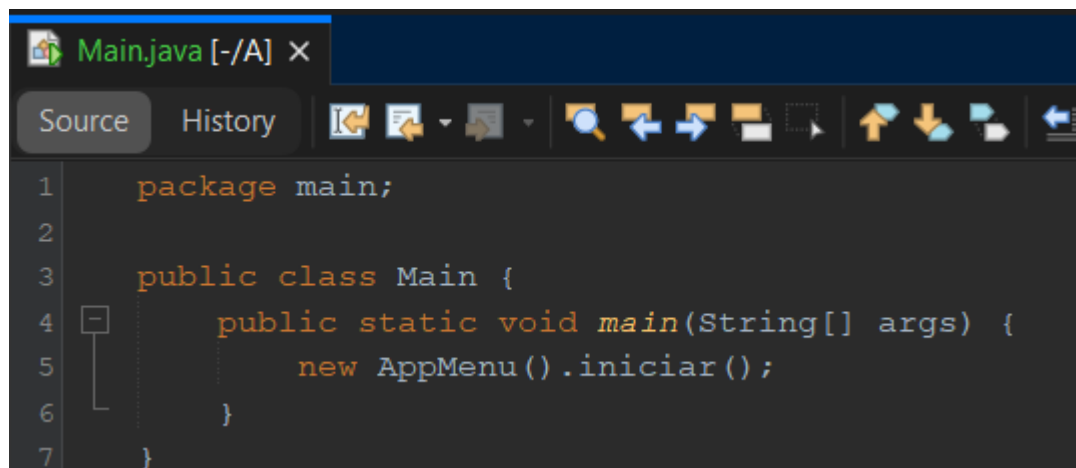
```

```

247 private void actualizarCredencial() throws Exception {
248     System.out.print("ID de la credencial a actualizar: ");
249     long id = Long.parseLong(sc.nextLine());
250
251     CredencialAcceso c = credService.getById(id);
252
253     if (c == null) {
254         System.out.println("Credencial no encontrada.");
255         return;
256     }
257
258     System.out.print("Nuevo hash password: ");
259     c.setHashPassword(sc.nextLine());
260     c.setUltimoCambio(LocalDate.now());
261
262     credService.actualizar(c);
263
264     System.out.println("✓ Credencial actualizada.");
265 }
266
267 private void eliminarCredencial() throws Exception {
268     System.out.print("ID de la credencial: ");
269     long id = Long.parseLong(sc.nextLine());
270
271     if (credService.eliminar(id))
272         System.out.println("✓ Credencial eliminada.");
273     else
274         System.out.println("No existe esa ID.");
275 }
276 }

```

Main.JAVA



```

Main.java [-/A] ×
Source History
1 package main;
2
3 public class Main {
4     public static void main(String[] args) {
5         new AppMenu().iniciar();
6     }
7 }

```

## - Package DAO

### UsuarioDAOImpl.JAVA

```
1 package dao;
2
3 import entities.Usuario;
4 import entities.CredencialAcceso;
5 import java.sql.*;
6 import java.time.LocalDateTime;
7 import java.util.ArrayList;
8 import java.util.List;
9
10 public class UsuarioDAOImpl implements GenericDao<Usuario> {
11
12     private final CredencialAccesoDAOImpl credDao = new CredencialAccesoDAOImpl();
13
14     @Override
15     public Usuario crear(Usuario entity, Connection conn) throws Exception {
16         // Primero crear la credencial si existe
17         if (entity.getCredencial() != null) {
18             credDao.crear(entity.getCredencial(), conn);
19         }
20
21         String sql = "INSERT INTO usuario (username, email, activo, fecha_registro, eliminado, credencial_id) VALUES (?, ?, ?, ?, ?, ?)";
22         try (PreparedStatement ps = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
23             ps.setString(1, entity.getUsername());
24             ps.setString(2, entity.getEmail());
25             ps.setBoolean(3, entity.isActivo());
26             ps.setTimestamp(4, entity.getFechaRegistro() != null ? Timestamp.valueOf(entity.getFechaRegistro()) : null);
27             ps.setBoolean(5, entity.isEliminado());
28             ps.setObject(6, entity.getCredencial() != null ? entity.getCredencial().getId() : null);
29
30             ps.executeUpdate();
31             try (ResultSet rs = ps.getGeneratedKeys()) {
32                 if (rs.next()) {
33                     entity.setId(rs.getLong(1));
34                 }
35             }
36         }
37     }
38 }
```

```
37     return entity;
38 }
39
40 @Override
41 public Usuario leer(long id, Connection conn) throws Exception {
42     String sql = "SELECT * FROM usuario WHERE id = ?";
43     try (PreparedStatement ps = conn.prepareStatement(sql)) {
44         ps.setLong(1, id);
45         try (ResultSet rs = ps.executeQuery()) {
46             if (rs.next()) {
47                 return mapear(rs, conn);
48             }
49         }
50     }
51     return null;
52 }
53
54 @Override
55 public List<Usuario> leerTodos(Connection conn) throws Exception {
56     String sql = "SELECT * FROM usuario";
57     List<Usuario> lista = new ArrayList<>();
58     try (PreparedStatement ps = conn.prepareStatement(sql)) {
59         ResultSet rs = ps.executeQuery();
60         while (rs.next()) {
61             lista.add(mapear(rs, conn));
62         }
63     }
64     return lista;
65 }
66
67 @Override
68 public Usuario actualizar(Usuario entity, Connection conn) throws Exception {
69     // Primero actualizar la credencial si existe
70     if (entity.getCredencial() != null) {
71         credDao.actualizar(entity.getCredencial(), conn);
72     }
73 }
```



```

74 String sql = "UPDATE usuario SET username = ?, email = ?, activo = ?, fecha_registro = ?, eliminado = ?, credencial_id = ? WHERE id = ?";
75 try (PreparedStatement ps = conn.prepareStatement(sql)) {
76     ps.setString(1, entity.getUsername());
77     ps.setString(2, entity.getEmail());
78     ps.setBoolean(3, entity.isActivo());
79     ps.setTimestamp(4, entity.getFechaRegistro() != null ? Timestamp.valueOf(entity.getFechaRegistro()) : null);
80     ps.setBoolean(5, entity.isEliminado());
81     ps.setObject(6, entity.getCredencial() != null ? entity.getCredencial().getId() : null);
82     ps.setLong(7, entity.getId());
83
84     ps.executeUpdate();
85 }
86 return entity;
87 }
88
89 @Override
90 public boolean eliminar(long id, Connection conn) throws Exception {
91     Usuario usuario = leer(id, conn);
92     if (usuario == null) return false;
93
94     String sql = "DELETE FROM usuario WHERE id = ?";
95     try (PreparedStatement ps = conn.prepareStatement(sql)) {
96         ps.setLong(1, id);
97         boolean eliminado = ps.executeUpdate() > 0;
98
99         // Eliminar la credencial asociada si existia
100         if (eliminado && usuario.getCredencial() != null) {
101             credDao.eliminar(usuario.getCredencial().getId(), conn);
102         }
103         return eliminado;
104     }
105 }

```

```

107 /**
108  * Convierte un ResultSet en un objeto Usuario
109  */
110 private Usuario mapear(ResultSet rs, Connection conn) throws Exception {
111     LocalDateTime fecha = rs.getTimestamp("fecha_registro") != null
112         ? rs.getTimestamp("fecha_registro").toLocalDateTime()
113         : null;
114
115     // Obtener la credencial asociada, si existe
116     CredencialAcceso cred = null;
117     long credId = rs.getLong("credencial_id");
118     if (!rs.wasNull()) {
119         cred = credDao.leer(credId, conn);
120     }
121
122     return new Usuario(
123         rs.getLong("id"),
124         rs.getBoolean("eliminado"),
125         rs.getString("username"),
126         rs.getString("email"),
127         rs.getBoolean("activo"),
128         fecha,
129         cred
130     );
131 }
132 }

```

## GenericDao.JAVA

```
GenericDao.java [-/A] X
Source History
1 package dao;
2
3 import java.sql.Connection;
4 import java.util.List;
5
6 public interface GenericDao<T> {
7     T crear(T entity, Connection conn) throws Exception;
8     T leer(long id, Connection conn) throws Exception;
9     List<T> leerTodos(Connection conn) throws Exception;
10    T actualizar(T entity, Connection conn) throws Exception;
11    boolean eliminar(long id, Connection conn) throws Exception;
12 }
```

## CredencialAccesoDAOImpl.JAVA

```
CredencialAccesoDAOImpl.java [-/A] X
Source History
1 package dao;
2
3 import entidades.CredencialAcceso;
4
5 import java.sql.*;
6 import java.time.LocalDateTime;
7 import java.util.ArrayList;
8 import java.util.List;
9
10 public class CredencialAccesoDAOImpl implements GenericDao<CredencialAcceso> {
11
12     @Override
13     public CredencialAcceso crear(CredencialAcceso entity, Connection conn) throws Exception {
14         String sql = "INSERT INTO credencial_acceso (hash_password, salt, ultimo_cambio, requiere_reset, eliminado) VALUES (?, ?, ?, ?, ?)";
15         try (PreparedStatement ps = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
16             ps.setString(1, entity.getHashPassword());
17             ps.setString(2, entity.getSalto());
18             ps.setTimestamp(3, entity.getUltimoCambio() != null ? Timestamp.valueOf(entity.getUltimoCambio()) : null);
19             ps.setBoolean(4, entity.isRequiereReset());
20             ps.setBoolean(5, entity.isEliminado());
21             ps.executeUpdate();
22
23             try (ResultSet rs = ps.getGeneratedKeys()) {
24                 if (rs.next()) {
25                     entity.setId(rs.getLong(1));
26                 }
27             }
28             return entity;
29         }
30     }
```

```
32     @Override
33     public CredencialAcceso leer(long id, Connection conn) throws Exception {
34         String sql = "SELECT * FROM credencial_acceso WHERE id = ?";
35         try (PreparedStatement ps = conn.prepareStatement(sql)) {
36             ps.setLong(1, id);
37             try (ResultSet rs = ps.executeQuery()) {
38                 if (rs.next()) {
39                     return mapear(rs);
40                 }
41             }
42         }
43         return null;
44     }
45
46     @Override
47     public List<CredencialAcceso> leerTodos(Connection conn) throws Exception {
48         String sql = "SELECT * FROM credencial_acceso";
49         List<CredencialAcceso> lista = new ArrayList<>();
50         try (PreparedStatement ps = conn.prepareStatement(sql)) {
51             ResultSet rs = ps.executeQuery();
52             while (rs.next()) {
53                 lista.add(mapear(rs));
54             }
55         }
56         return lista;
57     }
```

```

100 @Override
101 public CredencialAcceso actualizar(CredencialAcceso entity, Connection conn) throws Exception {
102     String sql = "UPDATE credencial_acceso SET hash_password = ?, salt = ?, ultimo_cambio = ?, requiere_reset = ?, eliminado = ? WHERE id = ?";
103     try (PreparedStatement ps = conn.prepareStatement(sql)) {
104         ps.setString(1, entity.getHashPassword());
105         ps.setString(2, entity.getSalt());
106         ps.setTimestamp(3, entity.getUltimoCambio() != null ? Timestamp.valueOf(entity.getUltimoCambio()) : null);
107         ps.setBoolean(4, entity.isRequiereReset());
108         ps.setBoolean(5, entity.isEliminado());
109         ps.setLong(6, entity.getId());
110         ps.executeUpdate();
111     }
112     return entity;
113 }
114
115 @Override
116 public boolean eliminar(long id, Connection conn) throws Exception {
117     String sql = "DELETE FROM credencial_acceso WHERE id = ?";
118     try (PreparedStatement ps = conn.prepareStatement(sql)) {
119         ps.setLong(1, id);
120         return ps.executeUpdate() > 0;
121     }
122 }
123
124 public CredencialAcceso getByUsuarioId(long usuarioId, Connection conn) throws Exception {
125     String sql = "SELECT * FROM credencial_acceso WHERE usuario_id = ?";
126     try (PreparedStatement ps = conn.prepareStatement(sql)) {
127         ps.setLong(1, usuarioId);
128         try (ResultSet rs = ps.executeQuery()) {
129             if (rs.next()) {
130                 return mapear(rs);
131             }
132         }
133     }
134     return null;
135 }

```

```

136 }
137
138 private CredencialAcceso mapear(ResultSet rs) throws SQLException {
139     LocalDateTime fecha = rs.getTimestamp("ultimo_cambio") != null
140         ? rs.getTimestamp("ultimo_cambio").toLocalDateTime()
141         : null;
142
143     return new CredencialAcceso(
144         rs.getLong("id"),
145         rs.getBoolean("eliminado"),
146         rs.getString("hash_password"),
147         rs.getString("salt"),
148         fecha,
149         rs.getBoolean("requiere_reset")
150     );
151 }
152
153 public CredencialAcceso leerPorUsuarioId(long usuarioId, Connection conn) throws Exception {
154     return getByUsuarioId(usuarioId, conn);
155 }
156
157 }
158

```

## Ejecución NETBEANS

### MENÚ PRINCIPAL

```
===== MENU PRINCIPAL =====  
1) Usuarios  
2) Credenciales  
0) Salir  
Elija opción:
```

### CREAR USUARIO

```
===== MENU PRINCIPAL =====  
1) Usuarios  
2) Credenciales  
0) Salir  
Elija opción: 1  
  
--- MENU USUARIOS ---  
1) Crear usuario  
2) Buscar por ID  
3) Listar todos  
4) Actualizar  
5) Eliminar  
6) Buscar por Username  
0) Volver  
Opción: 1  
Username: EJEMPLO  
Email: EJEMPLO@gmail.com  
Hash password: 010101  
? Usuario creado con éxito
```

### 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  
Opción: 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
Opción: 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
Opción: 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
Opción: 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
Opción: 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
Opción: 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
Opción: 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
Opción: 5
ID de la credencial: 1
? Credencial eliminada.
```

## SALIR

```
===== MENU PRINCIPAL =====
1) Usuarios
2) Credenciales
0) Salir
Elija opción: 0
Saliendo!
```

## Tabla CREDENCIAL\_ACCESO

Query 1 SQL File 1\* credencial\_acceso x

Limit to 1000 rows

1 • SELECT \* FROM tfi\_programacion2.credencial\_acceso;

Result Grid Filter Rows: Edit: Export/Import:

	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

The screenshot shows the SQL query editor with the following query entered:

```
1 • SELECT * FROM tfi_programacion2.usuario;
```