

PROYECTO FINAL

INSTITUTO SUPERIOR TECNOLOGICO CENESTUR

PROGRAMACION ORIENTADA A OBJETOS

ING. YADIRA FRANCO

INTEGRANTES:

ERICK CASTILLO

LISETH NATO

II. CONTENIDO	
III. INTRODUCCION.....	3
IV. OBJETIVOS	3
4.1. Objetivo General	3
4.2. Objetivos Específicos.....	3
V. DESARROLLO	4
5.1. VARIABLES	4
5.2. DIAGRAMA UML.....	5
5.3. CREACION DE CARPETAS DENTRO DE NUESTRO PROYECTO.....	7
5.3.1. CARPETA MODELO.....	7
5.3.2. CARPETA VISTA.....	9
5.3.3. CARPETA PERSISTENCIA	29
5.3.4. CARPETA CONTROLADOR.....	34
5.3.5. MAIN.....	38
VI. PRUEBAS	39
6.1. INGRESO Y GUARDAR USUARIO	39
.....	39
6.2. FUNCIONES DE ADMINISTRADOR.....	40
6.3. MENU – VER INSCRIPCIONES	42
6.3.1. ELIMINAR INSCRIPCION.....	43
6.4. CURSOS DISPONIBLES	45

6.4.1. AGREGAR UN NUEVO CURSO.....	46
6.5. ELIMINAR CURSO	47
6.6. AGREGAR UN NUEVO ADMINISTRADOR - GESTIONAR ADMINISTRADOR	
49	
6.7. ELIMINAR EL NUEVO ADMINISTRADOR.....	50
VII. CREACION DE ARCHIVO .jar.....	51
VIII. CREACION DE ARCHIVO .exe.....	52
IX. CONCLUSIONES	54
X. ENLACES DE DESCARGA DEL PROYECTO.....	56
10.1. Enlace Github	56
10.2. Enlace Video	56
XI. BIBLIOGRAFIA.....	56

III. INTRODUCCION

El presente informe investigativo describe el desarrollo y análisis de un proyecto en programación orientada a objetos, el mismo que tiene como propósito principal implementar un sistema de gestión de inscripciones para cursos vacacionales este es un proceso fundamental para las distintas instituciones ya sean educativas o recreativas las cuales buscan organizar y administrar de una manera eficiente la participación de niños, jóvenes o adultos en las diferentes actividades que se realizan durante el periodo de descanso escolar .

La implementación de este sistema permitirá optimizar la recaudación de los datos, facilitar el seguimiento y el control de las inscripciones, horarios. Este proyecto tiene como finalidad desarrollar un sistema el cual permita la gestión de forma efectiva de las inscripciones a los diferentes cursos vacacionales, garantizando un control sencillo y seguro para los usuarios

IV. OBJETIVOS

4.1. Objetivo General

Desarrollar una base de datos basado en un sistema de inscripciones a cursos vacacionales utilizando el CRUD en Java Swing el cual gestiones de una manera eficiente las inscripciones a un curso vacacional, permitiendo registrar, consultar, modifica, y eliminar información de cursos, participantes y horarios asegurando una gestión segura del proceso.

4.2. Objetivos Específicos

- Diseñar una estructura de base de datos en MySQL para así poder almacenar la información de los participantes, horarios, cursos vacacionales al cual se inscribieron de una manera segura y eficiente.

- Desarrollar una interfaz gráfica mediante Java Swing la cual permita al usuario realizar operaciones CRUD sobre los registros de los participantes.
- Incluir funciones que permitan la consulta los datos ingresados, de esta manera se facilita la gestión de los mismos.
- Validar la correcta conexión y comunicación entre la interfaz gráfica y la base de datos.
- Realizar las respectivas pruebas para verificar el funcionamiento de las operaciones y su cumplimiento con los requisitos solicitados.

Al concluir el proyecto se espera contar con una herramienta funcional que represente un sistema real de información y mantenible.

V. DESARROLLO

Nuestro sistema tiene por finalidad proveer a los usuarios una manera eficaz de realizar las inscripciones a los diferentes cursos, así como también permite a los administradores el poder consultar los datos ingresados por los usuarios y así poder tener una verificación de la información ingresada dando con esto un servicio eficiente.

5.1. VARIABLES

Para la elaboración de este proyecto necesitaremos las siguientes variables para usuarios y administradores:

Variables usuarios:

- Nombre
- Apellidos
- Edad

- Teléfono
- Cursos (la variable es definida por el administrador)

Variable cursos:

- Id_curso
- Nombre_del_curso y horario

5.2. DIAGRAMA UML

Con las variables definidas realizaremos un diagrama para las conexiones entre los archivos de nuestro proyecto, ya que existen interconexiones entre archivos ya que se heredan clases y los mismo deben poder unirse con nuestro SQL, para de esta manera dar una persistencia al usuario final.

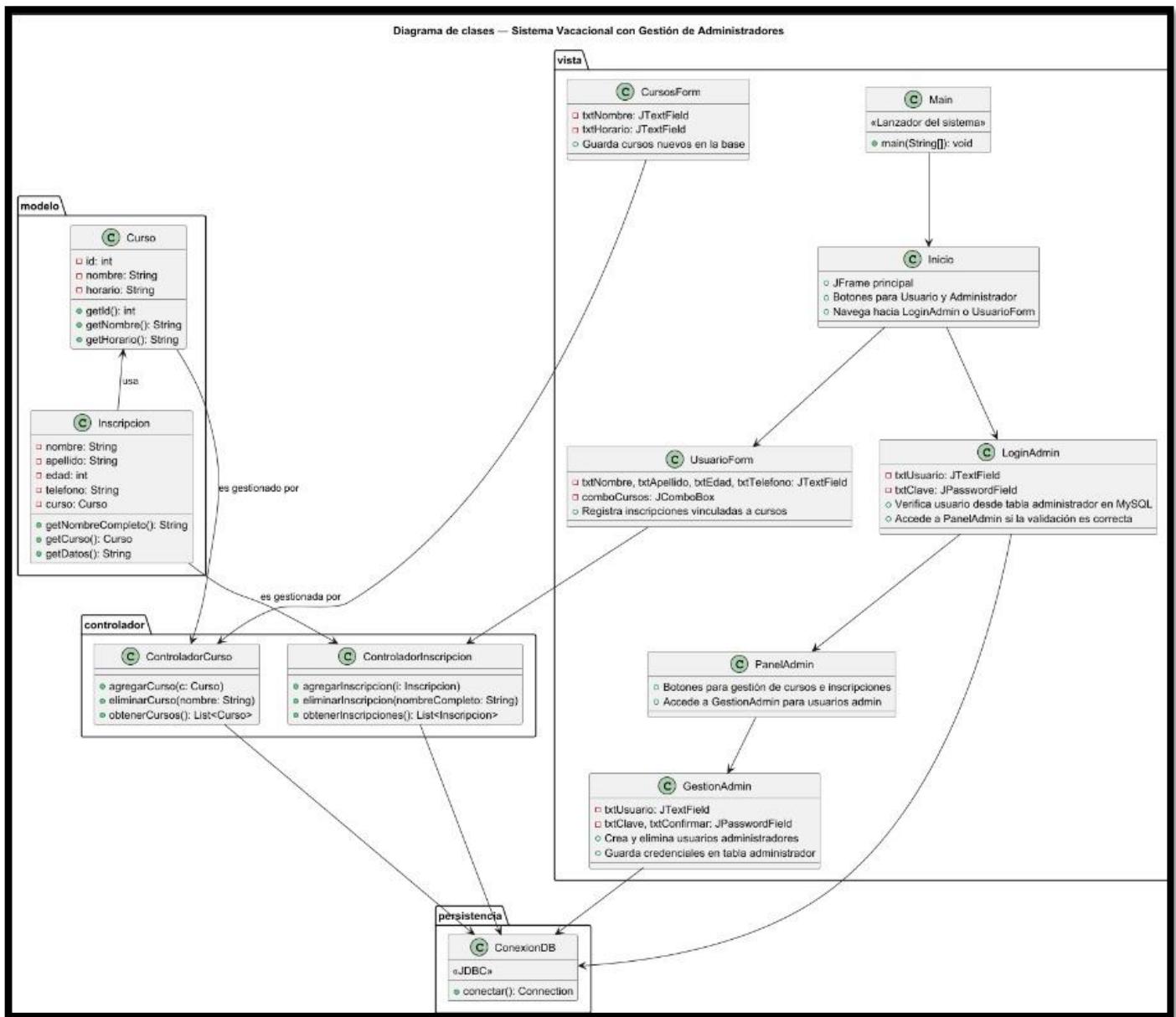


Imagen 1: Diagrama UML

5.3. CREACION DE CARPETAS DENTRO DE NUESTRO PROYECTO

5.3.1. CARPETA MODELO

5.3.1.1. Curso.java:

```
// Paquete al que pertenece la clase
package modelo;

// Clase que representa un curso académico
public class Curso {

    // Identificador único del curso
    private int id;

    // Nombre del curso
    private String nombre;

    // Horario en el que se imparte el curso
    private String horario;

    // Constructor sin ID (útil al crear un nuevo curso sin aún
    asignarle ID)
    public Curso(String nombre, String horario) {
        this.nombre = nombre;
        this.horario = horario;
    }

    // Constructor completo (útil al recuperar datos de una fuente
    con ID incluido)
    public Curso(int id, String nombre, String horario) {
        this.id = id;
        this.nombre = nombre;
        this.horario = horario;
    }

    // Método para obtener el ID del curso
    public int getId() {
        return id;
    }

    // Método para establecer/modificar el ID del curso
    public void setId(int id) {
        this.id = id;
    }

    // Método para obtener el nombre del curso
    public String getNombre() {
```

```
        return nombre;
    }

    // Método para obtener el horario del curso
    public String getHorario() {
        return horario;
    }

    // Método que define cómo se mostrará el curso como texto
    @Override
    public String toString() {
        // Ejemplo de salida: "Matemáticas (Lunes 10am)"
        return nombre + " (" + horario + ")";
    }
}
```

5.3.1.2. Incripcion.java:

```
// Indica que esta clase forma parte del paquete 'modelo'
package modelo;

// Clase que representa la inscripción de un estudiante en un curso
public class Incripcion {

    // Nombre del estudiante
    private String nombre;

    // Apellido del estudiante
    private String apellido;

    // Edad del estudiante
    private int edad;

    // Teléfono de contacto del estudiante
    private String telefono;

    // Curso en el que está inscrito el estudiante
    private Curso curso;

    // Constructor que inicializa todos los atributos de la
    // inscripción
    public Incripcion(String nombre, String apellido, int edad,
String telefono, Curso curso) {
        this.nombre = nombre;
        this.apellido = apellido;
```

```
        this.edad = edad;
        this.telefono = telefono;
        this.curso = curso;
    }

    // Métodos getter para acceder a los atributos individuales
    public String getNombre() { return nombre; }
    public String getApellido() { return apellido; }
    public int getEdad() { return edad; }
    public String getTelefono() { return telefono; }
    public Curso getCurso() { return curso; }

    // Devuelve el nombre completo del estudiante
    public String getNombreCompleto() {
        return nombre + " " + apellido;
    }

    // Devuelve un resumen con todos los datos relevantes de la
    inscripción
    public String getDatos() {
        // Ejemplo: "Ana Pérez - Edad: 22 - Tel: 0999999999 - Curso:
        Matemáticas"
        return getNombreCompleto() + " - Edad: " + edad + " - Tel: "
        + telefono + " - Curso: " + curso.getNombre();
    }
}
```

5.3.2. CARPETA VISTA

La pantalla principal de nuestro sistema contiene 2 botones, ingreso usuario e ingreso administrador

5.3.2.1. Inicio.form:



Imagen 2: Método Visual - Inicio.java

5.3.2.2. Inicio.java:

```
// Paquete que contiene las clases de la interfaz gráfica
package vista;

// Importación de clases necesarias para construir la interfaz
import javax.swing.*;

// Clase principal de la ventana de inicio del sistema
public class Inicio extends JFrame {

    // Botón para el acceso del administrador
    private JButton btnAdmin;

    // Botón para el acceso del usuario
    private JButton btnUsuario;

    // Panel principal que contiene todos los componentes visuales
    private JPanel rootPanel;

    // Constructor de la clase, configura la ventana al iniciarse
    public Inicio() {

        // Título de la ventana
        setTitle("Bienvenido al Curso Vacacional");
    }
}
```

```
// Se establece el panel principal como contenido de la
ventana
    setContentPane(rootPanel); // rootPanel normalmente se genera
desde un archivo .form si usas GUI builder

    // Tamaño de la ventana
    setSize(400, 200);

    // Ubica la ventana en el centro de la pantalla
    setLocationRelativeTo(null);

    // Indica que el programa debe cerrarse al cerrar esta
ventana
    setDefaultCloseOperation(EXIT_ON_CLOSE);

    // Acción al presionar el botón "Admin": abre ventana de
login y cierra la actual
    btnAdmin.addActionListener(e -> {
        new LoginAdmin().setVisible(true); // abre ventana para
administrador
        dispose(); // cierra la ventana actual
    });

    // Acción al presionar el botón "Usuario": abre formulario de
inscripción y cierra la actual
    btnUsuario.addActionListener(e -> {
        new UsuarioForm().setVisible(true); // abre ventana para
usuarios
        dispose(); // cierra la ventana actual
    });
}
```

5.3.2.3. Resultado visual:

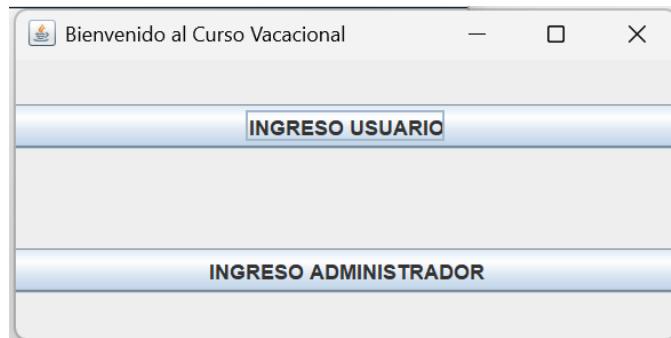


Imagen 3: Resultado Inicio.java

5.3.2.4. **UsuarioForm.form:**

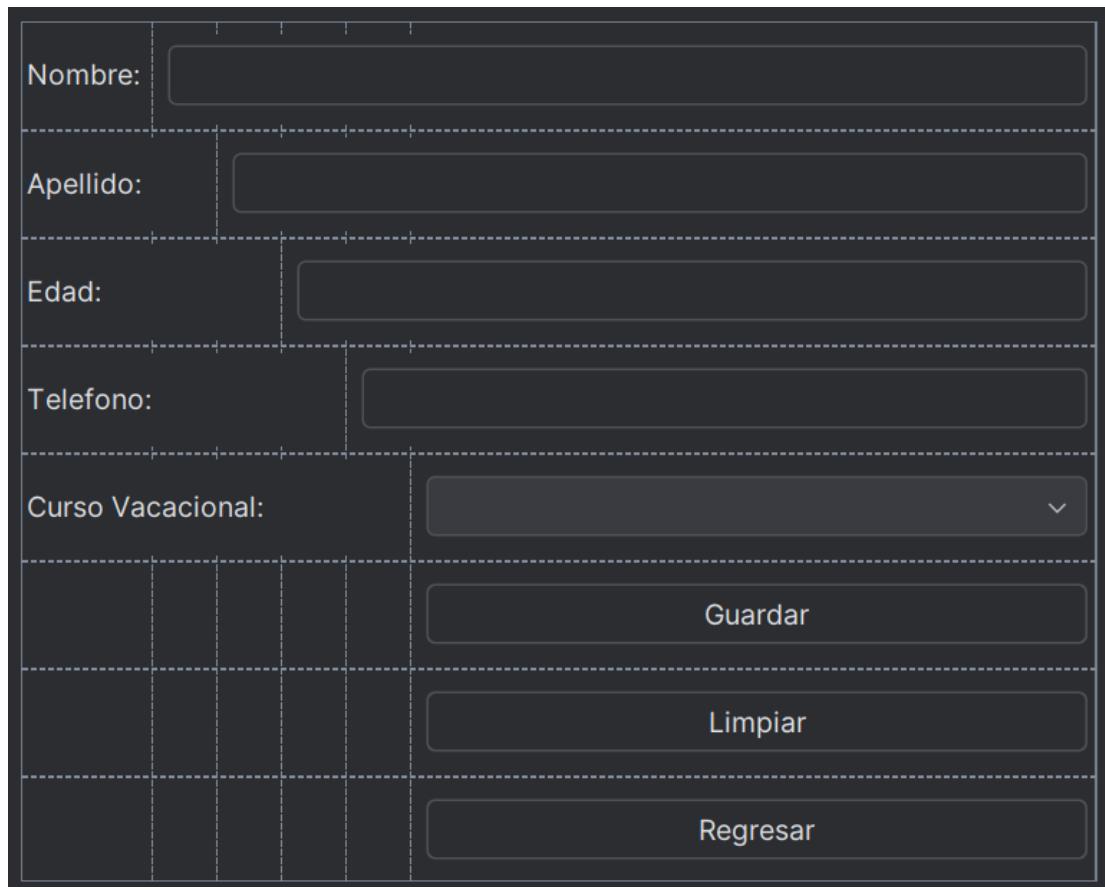


Imagen 4: Método Visual – Formulario Usuario

Curso vacacional lo define el administrador

5.3.2.5. **UsuarioForm.java:**

```
// Este archivo forma parte del paquete 'vista'  
package vista;  
  
// Importaciones necesarias para usar la lógica de control y  
// elementos visuales  
import controlador.ControladorCurso;  
import controlador.ControladorInscripcion;  
import modelo.Curso;  
import modelo.Inscripcion;  
  
import javax.swing.*;
```

```
import java.util.ArrayList;

// Ventana del formulario de inscripción para usuarios
public class UsuarioForm extends JFrame {

    // Campos de texto para capturar los datos del usuario
    private JTextField txtNombre;
    private JTextField txtApellido;
    private JTextField txtEdad;
    private JTextField txtTelefono;

    // ComboBox para seleccionar el curso disponible
    private JComboBox<Curso> comboCursos;

    // Botones de acción
    private JButton btnGuardar;
    private JButton btnLimpiar;
    private JButton btnRegresar;

    // Panel principal del formulario
    private JPanel rootPanel;

    // Constructor del formulario
    public UsuarioForm() {
        setTitle("Formulario de Inscripción"); // Título de la
        ventana
        setContentPane(rootPanel); // Establece el
        panel principal
        setSize(400, 350); // Tamaño de la
        ventana
        setLocationRelativeTo(null); // Centra la
        ventana en la pantalla
        setDefaultCloseOperation(EXIT_ON_CLOSE); // Cierra la
        aplicación al cerrar esta ventana

        inicializarCursos(); // Carga los
        cursos en el ComboBox

        // Acción al presionar 'Guardar': intenta registrar al
        usuario
        btnGuardar.addActionListener(e -> guardarInscripcion());

        // Acción al presionar 'Limpiar': borra los campos del
        formulario
        btnLimpiar.addActionListener(e -> limpiarCampos());

        // Acción al presionar 'Regresar': vuelve a la ventana de
        inicio
        btnRegresar.addActionListener(e -> {
            new Inicio().setVisible(true);
        });
    }
}
```

```
        dispose(); // Cierra la ventana actual
    });
}

// Método para cargar los cursos en el ComboBox desde el
controlador
private void inicializarCursos() {
    comboCursos.removeAllItems(); // Limpia el ComboBox
    ArrayList<Curso> cursos = ControladorCurso.obtenerCursos();
    // Obtiene la lista de cursos disponibles
    for (Curso : cursos) {
        comboCursos.addItem(curso); // Agrega cada curso al ComboBox
    }
}

// Método que intenta registrar al usuario en un curso
private void guardarInscripcion() {
    try {
        // Obtiene los datos ingresados por el usuario
        String nombre = txtNombre.getText().trim();
        String apellido = txtApellido.getText().trim();
        int edad = Integer.parseInt(txtEdad.getText().trim());
        String telefono = txtTelefono.getText().trim();
        Curso = (Curso) comboCursos.getSelectedItem();

        // Validación de campos
        if (nombre.isEmpty() || apellido.isEmpty() || telefono.isEmpty() || curso == null) {
            JOptionPane.showMessageDialog(this, "Completa todos los campos.");
            return;
        }

        // Crea una inscripción y la guarda a través del
        controlador
        Inscripcion = new Inscripcion(nombre, apellido, edad,
        telefono, curso);
        ControladorInscripcion.agregarInscripcion(inscripcion);

        // Muestra mensaje de éxito
        JOptionPane.showMessageDialog(this, "Inscripción
        guardada.");

        // Limpia los campos después de guardar
        limpiarCampos();
    } catch (Exception ex) {
        // Muestra error si la edad no es válida (por ejemplo, si
        no es un número)
```

```
        JOptionPane.showMessageDialog(this, "Edad inválida.");
    }

    // Método para limpiar todos los campos del formulario
    private void limpiarCampos() {
        txtNombre.setText("");
        txtApellido.setText("");
        txtEdad.setText("");
        txtTelefono.setText("");
        comboCursos.setSelectedIndex(0); // Reinicia la selección del
        curso
    }
}
```

5.3.2.6. Resultado Visual:

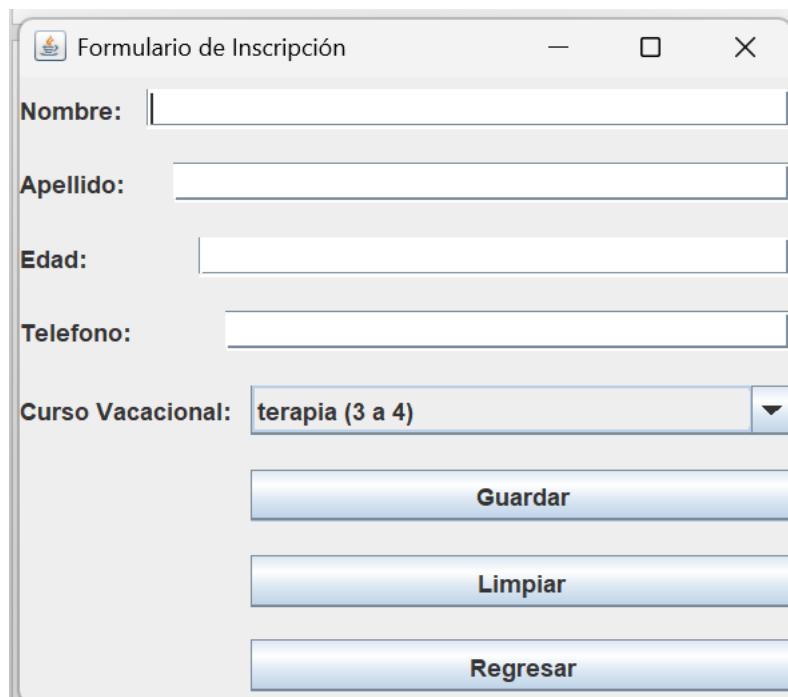


Imagen 4: Formulario Usuario

5.3.2.7. LoginAdmin.form:

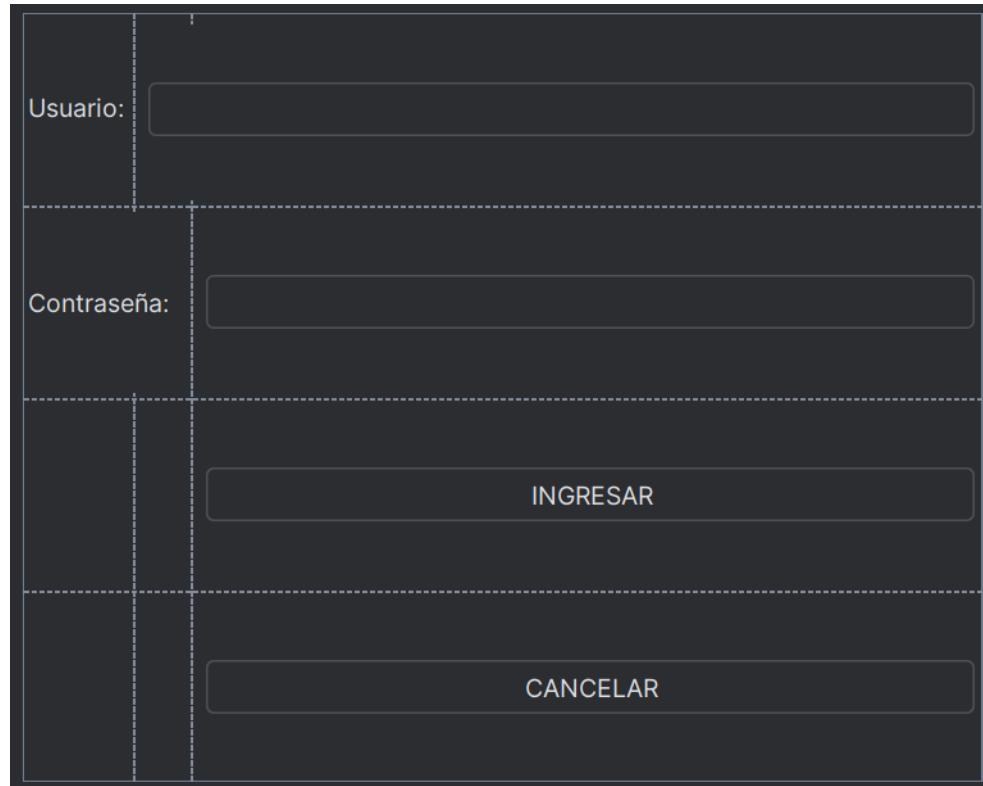


Imagen 6: Método Visual – Login Admin

5.3.2.8. LoginAdmin.java:

```
// Paquete donde se encuentra la clase de la vista
package vista;

// Importaciones para conexión a base de datos y componentes visuales
import persistencia.ConexionDB;
import javax.swing.*;
import java.sql.*;

// Ventana de acceso para administradores
public class LoginAdmin extends JFrame {

    // Campos para ingresar usuario y contraseña
    private JTextField txtUsuario;
    private JPasswordField txtClave;

    // Botones de acción
    private JButton btnIngresar;
    private JButton btnCancel;
```

```
// Panel principal del formulario
private JPanel rootPanel;

// Constructor de la ventana
public LoginAdmin() {
    setTitle("Acceso Administrador"); // Título de la
ventana
    setContentPane(rootPanel); // Establece el
panel principal
    setSize(350, 180); // Tamaño de la
ventana
    setLocationRelativeTo(null); // Centra la
ventana
    setDefaultCloseOperation(EXIT_ON_CLOSE); // Cierra la
app si se cierra esta ventana

    // Acción al presionar 'Ingresar': intenta validar
    credenciales
    btnIngresar.addActionListener(e -> {
        String usuario = txtUsuario.getText().trim();
// Obtiene usuario ingresado
        String clave = new String(txtClave.getPassword()).trim();
// Obtiene contraseña ingresada

        // Consulta SQL para validar administrador
        String sql = "SELECT * FROM administrador WHERE usuario =
? AND contraseña = ?";

        try {
            // Establece conexión con base de datos
            Connection conn = ConexionDB.conectar();

            // Prepara la consulta con parámetros seguros
            // (evita SQL Injection)
            PreparedStatement stmt =
conn.prepareStatement(sql)
        } {
            stmt.setString(1, usuario); // Asigna usuario al
            primer parámetro
            stmt.setString(2, clave); // Asigna contraseña al
            segundo parámetro

            ResultSet rs = stmt.executeQuery(); // Ejecuta la
            consulta

            if (rs.next()) {
                // Si se encuentra el usuario, abre el panel de
                administrador
                new PanelAdmin().setVisible(true);
                dispose(); // Cierra la ventana actual
            }
        }
    }
}
```

```
        } else {
            // Si no hay coincidencia, muestra mensaje de
            error
            JOptionPane.showMessageDialog(this, "Credenciales
            inválidas.");
        }
    } catch (SQLException ex) {
        // Si ocurre un error de conexión, muestra el mensaje
        JOptionPane.showMessageDialog(this, "Error de
        conexión: " + ex.getMessage());
    }
}

// Acción al presionar 'Cancelar': regresa a la ventana de
inicio
btnCancelar.addActionListener(e -> {
    new Inicio().setVisible(true); // Vuelve a la pantalla
principal
    dispose(); // Cierra la ventana
actual
})
}
```

5.3.2.9. Resultado Visual:

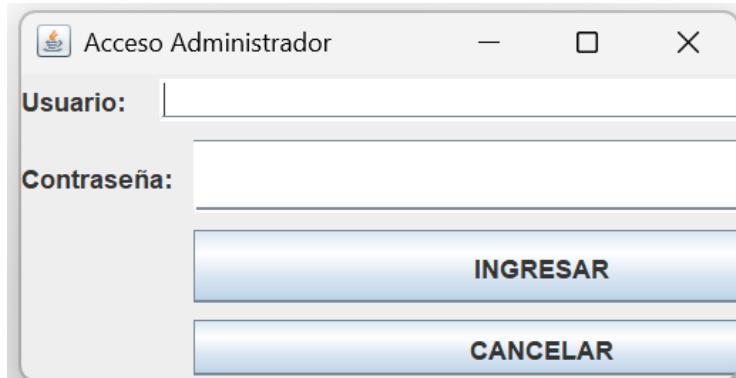


Imagen 5: Resultado Login Admin

5.3.2.10. PanelAdmin.form:



Imagen 6: Método Visual PanelAdmin

5.3.2.11. PanelAdmin.java:

```
// Paquete de interfaz gráfica
package vista;

// Importación de componentes Swing para construir la interfaz
import javax.swing.*;

// Ventana principal para el panel de administración
public class PanelAdmin extends JFrame {

    // Botones para realizar distintas acciones en el panel
    private JButton btnVerInscripciones;          // Ver lista de
inscripciones
    private JButton btnEliminarInscripcion;        // Eliminar una
inscripción existente
    private JButton btnAgregarCurso;               // Ir a formulario para
agregar curso
    private JButton btnEliminarCurso;              // Eliminar un curso
por nombre
```

```
    private JButton btnVerCursos;                      // Ver lista de cursos
disponibles
    private JButton btnRegresar;                     // Volver a la pantalla
de inicio
    private JButton btnGestionAdmin;                // Ir a la pantalla de
gestión de administrador

    // Panel principal de la interfaz, diseñado en un .form
private JPanel rootPanel;

    // Constructor que define la lógica del panel de administración
public PanelAdmin() {
    setTitle("Panel de Administración");           // Título de la
ventana
    setContentPane(rootPanel);                     // Establece el
panel principal
    setSize(420, 380);                            // Tamaño de la
ventana
    setLocationRelativeTo(null);                   // Centra la
ventana en pantalla
    setDefaultCloseOperation(EXIT_ON_CLOSE);          // Cierra el
programa al cerrar ventana

    // Acción: Ver inscripciones existentes
    btnVerInscripciones.addActionListener(e -> {
        var lista =
controlador.ControladorInscripcion.obtenerInscripciones(); // Lista
desde el controlador
        if (lista.isEmpty()) {
            JOptionPane.showMessageDialog(this, "No hay
inscripciones registradas.");
            return;
        }

        // Muestra cada inscripción en un cuadro de diálogo
        StringBuilder sb = new StringBuilder("Lista de
Inscripciones:\n\n");
        for (var ins : lista) {
            sb.append(ins.getDatos()).append("\n"); // Usa método
getDatos para resumir cada inscripción
        }

        JOptionPane.showMessageDialog(this, sb.toString());
    });

    // Acción: Eliminar una inscripción por nombre completo
    btnEliminarInscripcion.addActionListener(e -> {
        String nombre = JOptionPane.showInputDialog(this, "Nombre
completo del inscrito a eliminar:");
        if (nombre != null && !nombre.trim().isEmpty()) {
```

```
controlador.ControladorInscripcion.eliminarInscripcion(nombre.trim())
;
        JOptionPane.showMessageDialog(this, "Inscripción
eliminada.");
    }
} );
}

// Acción: Abrir formulario para agregar curso
btnAgregarCurso.addActionListener(e -> {
    new CursosForm().setVisible(true); // Abre formulario de
curso
    dispose(); // Cierra esta ventana
});

// Acción: Eliminar curso por nombre
btnEliminarCurso.addActionListener(e -> {
    String nombre = JOptionPane.showInputDialog(this, "Nombre
del curso a eliminar:");
    if (nombre != null && !nombre.trim().isEmpty()) {

controlador.ControladorCurso.eliminarCurso(nombre.trim());
        JOptionPane.showMessageDialog(this, "Curso
eliminado.");
    }
});

// Acción: Ver lista de cursos registrados
btnVerCursos.addActionListener(e -> {
    var lista = controlador.ControladorCurso.obtenerCursos();
    if (lista.isEmpty()) {
        JOptionPane.showMessageDialog(this, "No hay cursos
registrados.");
        return;
    }

    // Muestra cada curso con su horario
    StringBuilder sb = new StringBuilder("Lista de
Cursos:\n\n");
    for (var c : lista) {
        sb.append("- ").append(c.getNombre()).append("(
")
.append(c.getHorario()).append(")\n");
    }

    JOptionPane.showMessageDialog(this, sb.toString());
});

// Acción: Abrir ventana de gestión de administrador
btnGestionAdmin.addActionListener(e -> {
    new GestionAdmin().setVisible(true);
});
```

```
        dispose(); // Cierra la ventana actual
    });

// Acción: Regresa al inicio del sistema
btnRegresar.addActionListener(e -> {
    new Inicio().setVisible(true);
    dispose();
});
}
```

5.3.2.12. Resultado Visual:

Para poder acceder al Panel de administración, debemos ingresar nuestro usuario y contraseña

- **Usuario:** admin
- **Contraseña:** 1234

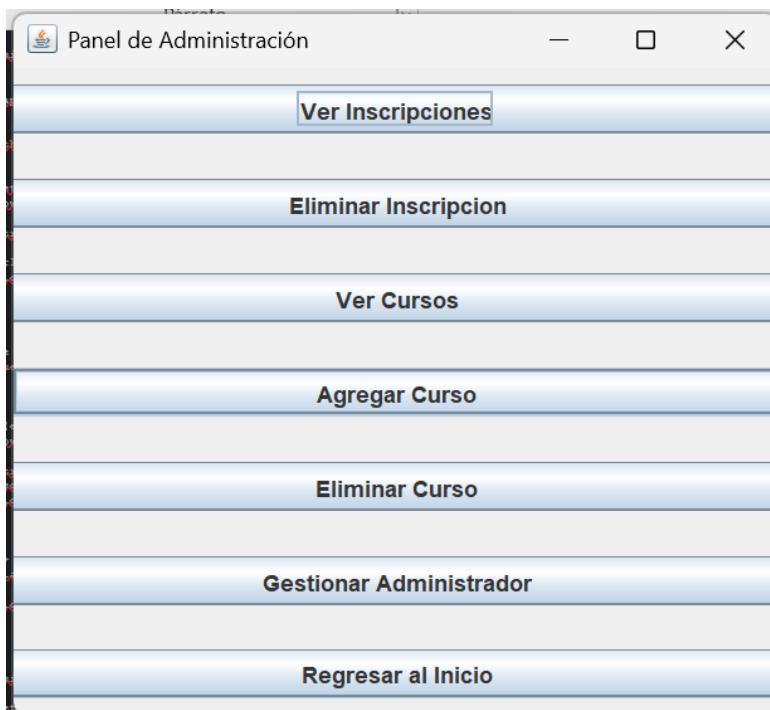


Imagen 7: Resultado Panel Admin

5.3.2.13. CursosForm.form:

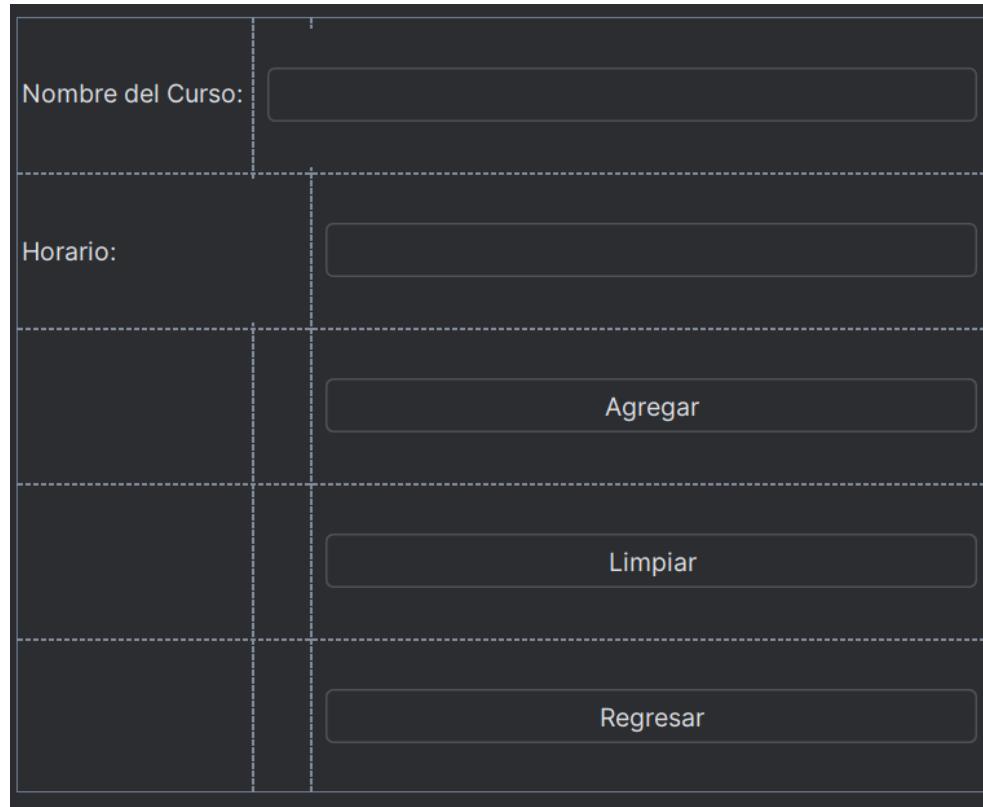


Imagen 8: Método Visual Formulario Cursos

5.3.2.14. CursosForm.java:

```
package vista; // Paquete donde se encuentra esta clase (vista del sistema)

import controlador.ControladorCurso; // Importa el controlador para manejar los cursos
import modelo.Curso; // Importa el modelo Curso para crear objetos de tipo curso

import javax.swing.*; // Librería de Swing para interfaces gráficas

// Clase que representa el formulario de gestión de cursos
public class CursosForm extends JFrame {
    // Componentes gráficos (campos de texto, botones y panel principal)
    private JTextField txtNombre;
```

```
private JTextField txtHorario;
private JButton btnAgregar;
private JButton btnLimpiar;
private JButton btnRegresar;
private JPanel rootPanel;

// Constructor de la ventana
public CursosForm() {
    setTitle("Gestión de Cursos"); // Título de la ventana
    setContentPane(rootPanel); // Define el panel principal
    setSize(350, 220); // Tamaño de la ventana
    setLocationRelativeTo(null); // Centra la ventana en la pantalla
    setDefaultCloseOperation(EXIT_ON_CLOSE); // Cierra la aplicación al salir

    // Acción del botón "Agregar"
    btnAgregar.addActionListener(e -> {
        String nombre = txtNombre.getText().trim(); // Obtiene el nombre del curso
        String horario = txtHorario.getText().trim(); // Obtiene el horario del curso

        // Valida que los campos no estén vacíos
        if (!nombre.isEmpty() && !horario.isEmpty()) {
            // Crea un nuevo objeto Curso con los datos ingresados
            Curso nuevo = new Curso(nombre, horario);
            // Llama al controlador para agregar el curso a la lista o base de datos
            ControladorCurso.agregarCurso(nuevo);
            // Muestra mensaje de confirmación
            JOptionPane.showMessageDialog(this, "Curso agregado.");
            // Limpia los campos del formulario
            limpiarCampos();
        } else {
            // Si algún campo está vacío, muestra advertencia
            JOptionPane.showMessageDialog(this, "Completa ambos campos.");
        }
    });

    // Acción del botón "Limpiar"
    btnLimpiar.addActionListener(e -> limpiarCampos()); // Llama al método que borra los campos

    // Acción del botón "Regresar"
    btnRegresar.addActionListener(e -> {
        new PanelAdmin().setVisible(true); // Abre la ventana
    });
}
```

```
del panel de administración
        dispose(); // Cierra la ventana actual
    );
}

// Método para limpiar los campos de texto
private void limpiarCampos() {
    txtNombre.setText(""); // Borra el contenido del campo
nombre
    txtHorario.setText(""); // Borra el contenido del campo
horario
}
}
```

5.3.2.15. Resultado Visual:

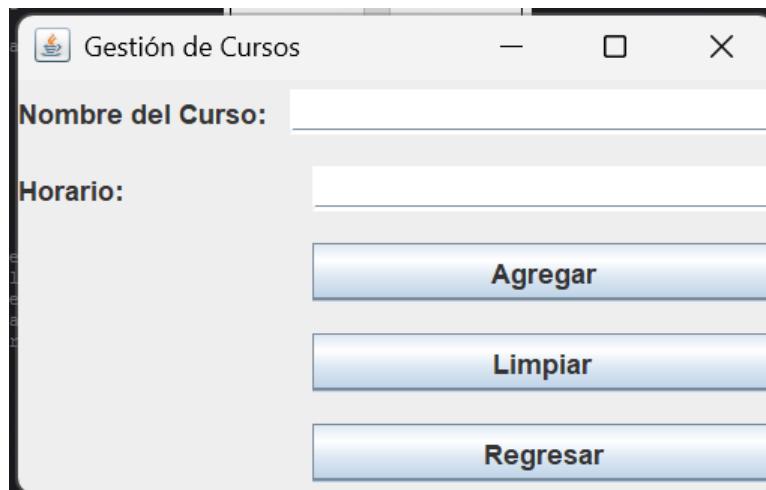


Imagen 9: Resultado Formulario Cursos

5.3.2.16. GestionAdmin.form:

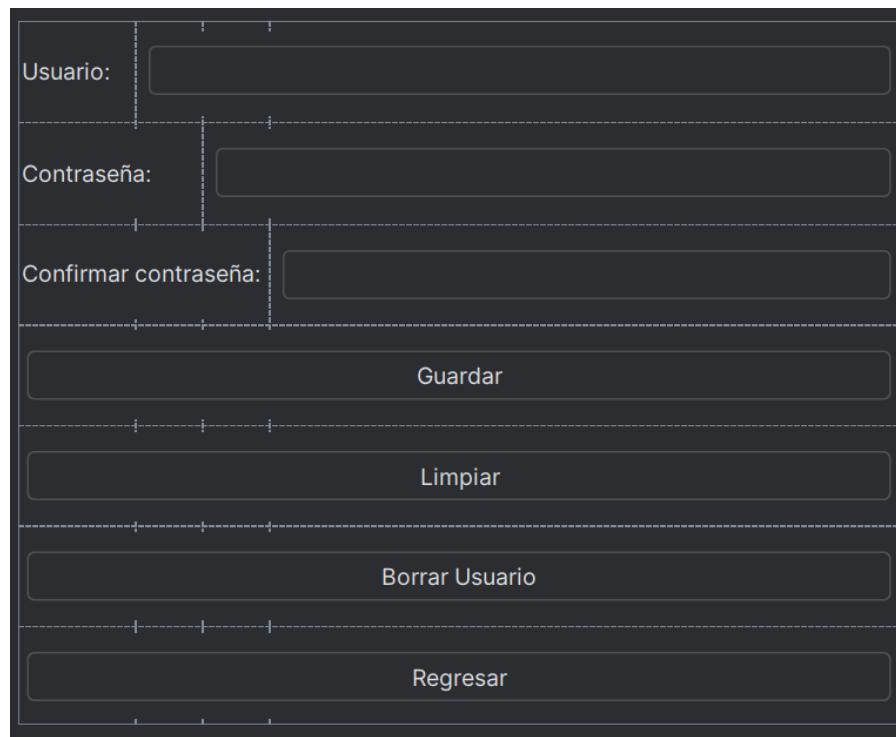


Imagen 10: Formulario Gestion de administradores

5.3.2.17. GestionAdmin.java:

```
// Paquete gráfico de la aplicación
package vista;

// Importación de clase para conectar con la base de datos
import persistencia.ConexionDB;

import javax.swing.*;
import java.sql.*;

// Ventana para administrar cuentas de tipo "administrador"
public class GestionAdmin extends JFrame {

    // Campos para ingresar datos del nuevo administrador
    private JTextField txtUsuario;
    private JPasswordField txtClave;
    private JPasswordField txtConfirmar;

    // Botones de acción en la interfaz
    private JButton btnGuardar;
    private JButton btnLimpiar;
    private JButton btnBorrar;
```

```
private JButton btnRegresar;

// Panel principal que contiene todos los elementos
private JPanel rootPanel;

// Constructor que define el comportamiento de la ventana
public GestionAdmin() {
    setTitle("Gestionar Administrador"); // Título de la
ventana
    setContentPane(rootPanel); // Panel principal
como contenido
    setSize(400, 280); // Tamaño de la
ventana
    setLocationRelativeTo(null); // Centra la ventana
en la pantalla
    setDefaultCloseOperation(EXIT_ON_CLOSE); // Cierra el programa
al cerrar esta ventana

    // Al presionar 'Guardar', ejecuta función para registrar un nuevo
administrador
    btnGuardar.addActionListener(e -> guardarAdministrador());

    // Limpia los campos del formulario
    btnLimpiar.addActionListener(e -> limpiarCampos());

    // Borra un administrador por nombre de usuario
    btnBorrar.addActionListener(e -> borrarAdministrador());

    // Regresa al Panel de Administración
    btnRegresar.addActionListener(e -> {
        new PanelAdmin().setVisible(true);
        dispose(); // Cierra esta ventana
    });
}

// Lógica para guardar un nuevo administrador en la base de datos
private void guardarAdministrador() {
    String usuario = txtUsuario.getText().trim();
    String clave = new String(txtClave.getPassword()).trim();
    String confirmar = new String(txtConfirmar.getPassword()).trim();

    // Validación: campos vacíos
    if (usuario.isEmpty() || clave.isEmpty() || confirmar.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Completa todos los
campos.");
        return;
    }

    // Validación: contraseñas deben coincidir
    if (!clave.equals(confirmar)) {
        JOptionPane.showMessageDialog(this, "Las contraseñas no
coinciden.");
        return;
    }
}
```

```
String sql = "INSERT INTO administrador (usuario, contraseña) VALUES  
 (?, ?);  
  
// Conexión con la base de datos usando conexión segura  
try (Connection conn = ConexionDB.conectar();  
     PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
    // Asignación de valores al SQL  
    stmt.setString(1, usuario);  
    stmt.setString(2, clave);  
    stmt.executeUpdate(); // Ejecuta el guardado  
  
    JOptionPane.showMessageDialog(this, "Administrador guardado.");  
    limpiarCampos(); // Limpia campos luego de guardar  
  
} catch (SQLException ex) {  
    JOptionPane.showMessageDialog(this, "Error al guardar: " +  
        ex.getMessage());  
}  
}  
  
// Lógica para eliminar un administrador por nombre de usuario  
private void borrarAdministrador() {  
    String usuario = txtUsuario.getText().trim();  
  
    // Validación: debe ingresar usuario  
    if (usuario.isEmpty()) {  
        JOptionPane.showMessageDialog(this, "Ingresa el nombre del  
        usuario a borrar.");  
        return;  
    }  
  
    String sql = "DELETE FROM administrador WHERE usuario = ?";  
  
    try (Connection conn = ConexionDB.conectar();  
         PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
        stmt.setString(1, usuario); // Asigna parámetro al SQL  
        int filas = stmt.executeUpdate(); // Ejecuta borrado  
  
        // Mensaje según resultado  
        if (filas > 0) {  
            JOptionPane.showMessageDialog(this, "Usuario eliminado.");  
        } else {  
            JOptionPane.showMessageDialog(this, "Usuario no  
            encontrado.");  
        }  
        limpiarCampos();  
  
    } catch (SQLException ex) {  
        JOptionPane.showMessageDialog(this, "Error al eliminar: " +  
            ex.getMessage());  
    }
}
```

```
// Función para dejar los campos vacíos
private void limpiarCampos() {
    txtUsuario.setText("");
    txtClave.setText("");
    txtConfirmar.setText("");
}
```

5.3.2.18. Resultado Visual:

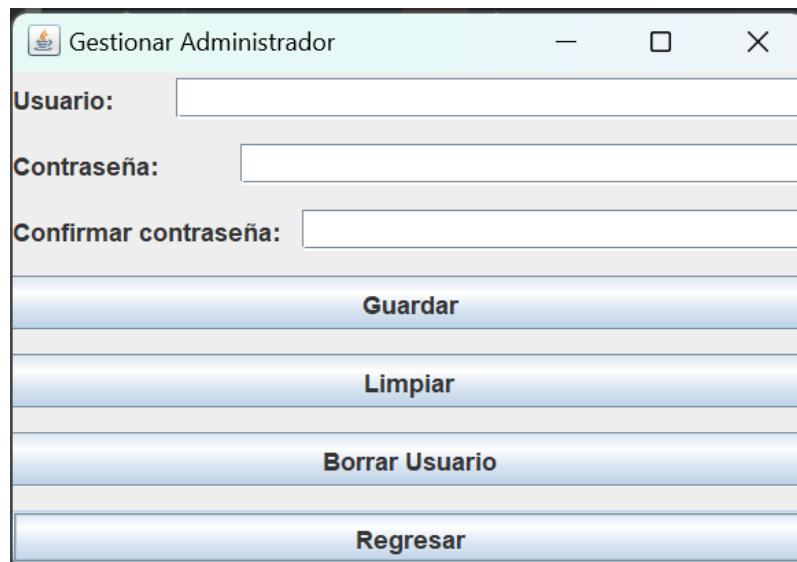


Imagen 11: Resultado visual

5.3.3. CARPETA PERSISTENCIA

5.3.3.1. ConexionDB.java:

Vamos a crear una nueva conexión Clever Cloud, para ello vamos a ingresar el siguiente código.

```
package persistencia; // Paquete donde estará la clase para manejar
la conexión a la base de datos

import java.sql.Connection; // Clase para manejar la conexión a
la base de datos
import java.sql.DriverManager; // Clase que administra los drivers
para conectarse a la BD
import java.sql.SQLException; // Excepción que se lanza si ocurre
```

```
un error de SQL

// Clase para manejar la conexión a la base de datos MySQL
public class ConexionDB {

    // Datos de conexión: URL, usuario y contraseña
    private static final String URL =
"jdbc:mysql://bxroumhh7twmq4mhsult-mysql.services.clever-
cloud.com:3306/bxroumhh7twmq4mhsult?useSSL=false&serverTimezone=UTC";
    private static final String USUARIO = "ujcmibexiima4psl";
    private static final String CLAVE = "2rBrF45e1xqt0RztcoAF";

    // Método estático que establece la conexión a la base de datos
    public static Connection conectar() {
        Connection conn = null; // Objeto que representará la
        conexión

        try {
            // Cargar el driver de MySQL (necesario para conectar con
            JDBC)
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Establecer conexión usando los datos definidos (URL,
            usuario, clave)
            conn = DriverManager.getConnection(URL, USUARIO, CLAVE);

            // Si todo salió bien, mensaje de confirmación
            System.out.println("Conexión exitosa a MySQL (Clever
Cloud)");

        } catch (ClassNotFoundException | SQLException e) {
            // Si falla el driver o la conexión, muestra el error
            System.out.println("Error de conexión: " +
e.getMessage());
        }

        // Devuelve el objeto Connection (puede ser null si falló)
        return conn;
    }
}
```

5.3.3.2. Conexión en MYSQL

Ingresamos a nuestra base de datos y creamos una nueva conexión.

Esto lo creamos en la pestaña que dice Connections.

Welcome to MySQL Workbench

MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

[Browse Documentation >](#)

[Read the Blog >](#)

[Discuss on the Forums >](#)

MySQL Connections

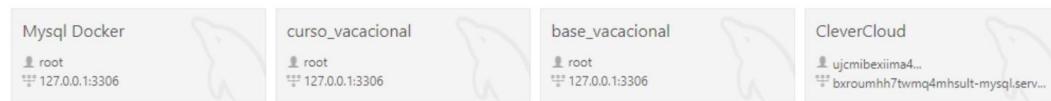


Imagen 12: Interfaz de MySQL

Creamos una nueva ventana de para insertar datos y colocamos los siguientes datos.

```
CREATE TABLE curso (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100),
    horario VARCHAR(100)
);
```

```
CREATE TABLE inscripcion (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100),
    apellido VARCHAR(100),
    edad INT,
    telefono VARCHAR(20),
```

```
curso_id INT,  
    FOREIGN KEY (curso_id) REFERENCES curso(id)  
);
```

```
CREATE TABLE administrador(  
    Id INT PRIMARY KEY AUTO_INCREMENT,  
    nombre VARCHAR(100),  
    password VARCHAR(100)  
);
```

El resultado se verá así:

The screenshot shows a MySQL Workbench interface with three SQL statements listed in the query editor:

```
1 CREATE TABLE curso (  
2     id INT PRIMARY KEY AUTO_INCREMENT,  
3     nombre VARCHAR(100),  
4     horario VARCHAR(100)  
5 );  
6  
7 • CREATE TABLE inscripcion (  
8     id INT PRIMARY KEY AUTO_INCREMENT,  
9     nombre VARCHAR(100),  
10    apellido VARCHAR(100),  
11    edad INT,  
12    telefono VARCHAR(20),  
13    curso_id INT,  
14    FOREIGN KEY (curso_id) REFERENCES curso(id)  
15 );  
16  
17 • CREATE TABLE administrador(  
18     Id INT PRIMARY KEY AUTO_INCREMENT,  
19     nombre VARCHAR(100),  
20     password VARCHAR(100)  
21 );
```

Imagen 13: Sentencia para Base de Datos

Y nuestras tablas se verán así:

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
1 • SELECT * FROM bxroumhh7twmq4mhsult.administrador;
```

The result grid displays the following data:

	id	usuario	contraseña
▶	1	admin	1234
▶	3	cenestur	1234
*	NULL	NULL	NULL

Imagen 14: Tabla administrador

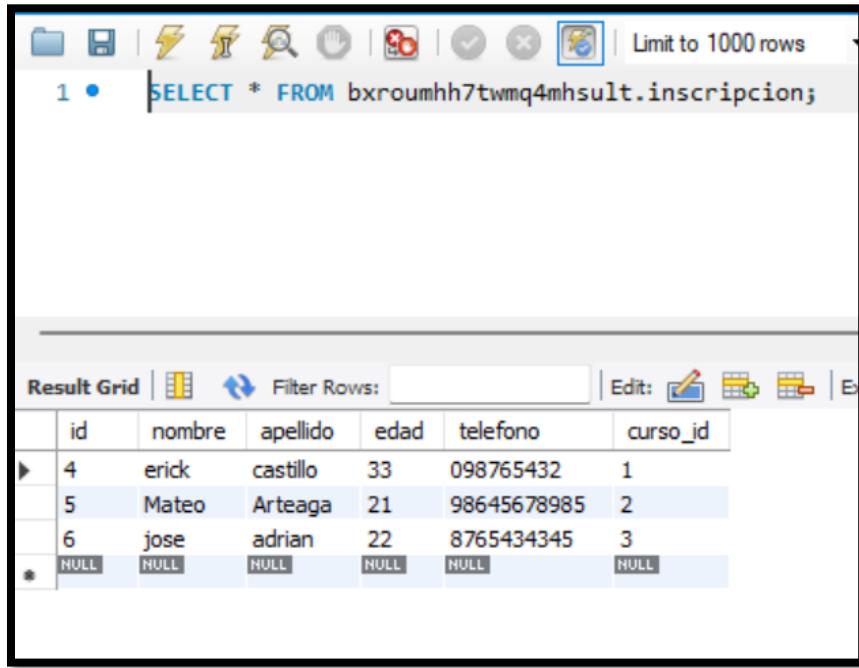
The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
1 • SELECT * FROM bxroumhh7twmq4mhsult.curso;
```

The result grid displays the following data:

	id	nombre	horario
▶	1	terapia	3 a 4
▶	2	jardineria	4 a 6
▶	3	veterinaria	6 a 7
*	NULL	NULL	NULL

Imagen 15: Tabla curso



The screenshot shows a MySQL Workbench interface. At the top, there is a toolbar with various icons. Below the toolbar, a query editor window displays the SQL command: `SELECT * FROM bxroumhh7twmq4mhslt.inscripcion;`. The result grid below the query editor shows the data from the 'inscripcion' table:

	id	nombre	apellido	edad	telefono	curso_id
▶	4	erick	castillo	33	098765432	1
	5	Mateo	Arteaga	21	98645678985	2
*	6	jose	adrian	22	8765434345	3
	NULL	HULL	NULL	NULL	NULL	NULL

Imagen 16: Tabla Inscripción

Con esto hemos terminado de realizar con éxito nuestra conexión.

5.3.4. CARPETA CONTROLADOR

Nuestros archivos de controlador nos permitirán gestionar la información desde nuestra base de datos hacia nuestra aplicación.

5.3.4.1. ControladorCurso.java:

```
// Paquete que contiene la lógica de negocio
package controlador;

// Clase modelo que representa los datos del curso
import modelo.Curso;

// Clase para establecer conexión con la base de datos
import persistencia.ConexionDB;

import java.sql.*;
import java.util.ArrayList;

// Clase que gestiona operaciones CRUD para los cursos
public class ControladorCurso {
```

```

// Agrega un nuevo curso a la base de datos
public static void agregarCurso(Curso curso) {
    String sql = "INSERT INTO curso (nombre, horario) VALUES (?, ?);"

    // Se usan parámetros para prevenir inyecciones SQL
    try (Connection conn = ConexionDB.conectar();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, curso.getNombre());      // Nombre del curso
        stmt.setString(2, curso.getHorario());      // Horario del curso
        stmt.executeUpdate();                      // Ejecuta la inserción

    } catch (SQLException e) {
        System.out.println("Error al agregar curso: " + e.getMessage());
    }
}

// Elimina un curso por su nombre
public static void eliminarCurso(String nombreCurso) {
    String sql = "DELETE FROM curso WHERE nombre = ?";

    try (Connection conn = ConexionDB.conectar();
         PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, nombreCurso);           // Nombre del curso a
eliminar
        stmt.executeUpdate();                  // Ejecuta la
eliminación

    } catch (SQLException e) {
        System.out.println("Error al eliminar curso: " +
e.getMessage());
    }
}

// Obtiene todos los cursos registrados en la base de datos
public static ArrayList<Curso> obtenerCursos() {
    ArrayList<Curso> listaCursos = new ArrayList<>();
    String sql = "SELECT id, nombre, horario FROM curso";

    // Se utiliza Statement para una consulta sin parámetros
    try (Connection conn = ConexionDB.conectar();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {

        // Recorre los resultados y los convierte a objetos Curso
        while (rs.next()) {
            int id = rs.getInt("id");
            String nombre = rs.getString("nombre");
            String horario = rs.getString("horario");
            listaCursos.add(new Curso(id, nombre, horario)); // Constructor con ID
        }

    } catch (SQLException e) {
}
}

```

```
        System.out.println("Error al obtener cursos: " +
e.getMessage());
    }

    return listaCursos; // Devuelve la lista completa
}
}
```

5.3.4.2. ControladorInscripcion.java:

```
// Paquete del controlador que gestiona inscripciones
package controlador;

import modelo.Curso;
import modelo.Inscripcion;
import persistencia.ConexionDB;

import java.sql.*;
import java.util.ArrayList;

// Clase encargada de realizar operaciones CRUD sobre inscripciones
public class ControladorInscripcion {

    // Agrega una nueva inscripción a la base de datos
    public static void agregarInscripcion(Inscripcion inscripcion) {
        String sql = "INSERT INTO inscripcion (nombre, apellido, edad,
telefono, curso_id) VALUES (?, ?, ?, ?, ?)";

        try (Connection conn = ConexionDB.conectar());
// Se conecta a la base de datos
        PreparedStatement stmt = conn.prepareStatement(sql)) {
// Prepara la consulta con parámetros

        // Asignación de valores a los parámetros SQL
        stmt.setString(1, inscripcion.getNombre());
        stmt.setString(2, inscripcion.getApellido());
        stmt.setInt(3, inscripcion.getEdad());
        stmt.setString(4, inscripcion.getTelefono());
        stmt.setInt(5, inscripcion.getCurso().getId());

        stmt.executeUpdate(); // Ejecuta la inserción

    } catch (SQLException e) {
        // Si ocurre error en la inserción
        System.out.println("Error al guardar inscripción: " +
e.getMessage());
    }
}

// Elimina una inscripción según el nombre completo
public static void eliminarInscripcion(String nombreCompleto) {
```

```

        String sql = "DELETE FROM inscripcion WHERE CONCAT(nombre, ' ',  
apellido) = ?";  
  

        try (Connection conn = ConexionDB.conectar();  
PreparedStatement stmt = conn.prepareStatement(sql)) {  
  

            stmt.setString(1, nombreCompleto); // 🚧 Nombre y apellido  
concatenados como en la base de datos  
stmt.executeUpdate(); // Ejecuta la eliminación  
  

        } catch (SQLException e) {  
            System.out.println("Error al eliminar inscripción: " +  
e.getMessage());  
        }
    }  
  

    // Obtiene todas las inscripciones con sus datos de curso
    public static ArrayList<Inscripcion> obtenerInscripciones() {
        ArrayList<Inscripcion> lista = new ArrayList<>();  
  

        // Consulta que une inscripciones con el curso correspondiente
        String sql = """  

            SELECT i.nombre, i.apellido, i.edad, i.telefono,  

                c.id AS curso_id, c.nombre AS curso_nombre, c.horario  

            FROM inscripcion i  

            JOIN curso c ON i.curso_id = c.id
        """;  
  

        try (Connection conn = ConexionDB.conectar();  
Statement stmt = conn.createStatement();  
ResultSet rs = stmt.executeQuery(sql)) {  
  

            // Recorre cada fila del resultado
            while (rs.next()) {
                // Datos del inscrito
                String nombre = rs.getString("nombre");
                String apellido = rs.getString("apellido");
                int edad = rs.getInt("edad");
                String telefono = rs.getString("telefono");  
  

                // Datos del curso asociado
                int cursoId = rs.getInt("curso_id");
                String cursoNombre = rs.getString("curso_nombre");
                String horario = rs.getString("horario");  
  

                // Construye objetos y los añade a la lista
                Curso = new Curso(cursoId, cursoNombre, horario);
                Inscripcion ins = new Inscripcion(nombre, apellido, edad,
telefono, curso);
                lista.add(ins);
            }
        } catch (SQLException e) {
            System.out.println("Error al obtener inscripciones: " +

```

```
    e.getMessage() );
}

return lista; // Devuelve la lista de inscripciones completas
}
```

5.3.5. MAIN

Nuestro archivo main prueba la conexión y lanza la interfaz gráfica.

```
// Paquete donde está la clase principal
package vista;

// Importa la clase que maneja la conexión a la base de datos
import persistencia.ConexionDB;
import java.sql.Connection;

// Punto de entrada de la aplicación
public class Main {
    public static void main(String[] args) {

        // Intenta conectar a la base de datos
        Connection conn = ConexionDB.conectar();

        // Imprime mensaje de confirmación
        if (conn != null) {
            System.out.println("¡Conectado correctamente!");
        } else {
            System.out.println("No se pudo conectar.");
        }

        // Abre la interfaz principal de la aplicación
        new Inicio().setVisible(true);
    }
}
```

VI. PRUEBAS

6.1. INGRESO Y GUARDAR USUARIO



Imagen 17: menú Inicio

Nos aparecerá un formulario en el cual debemos llenar los datos y procedemos a guardar.

A screenshot of a "Formulario de Inscripción" window. It contains fields for Nombre (LISETH), Apellido (NATO), Edad (25), Telefono (094566534), and Curso Vacacional (terapia (3 a 4)). Below these fields are three buttons: "Guardar" (highlighted with a red border), "Limpiar", and "Regresar".

Imagen 18: Ingreso de datos -formulario de inscripción

Nos aparecerá un mensaje en el cual nos indica que se guardó con éxito los datos.

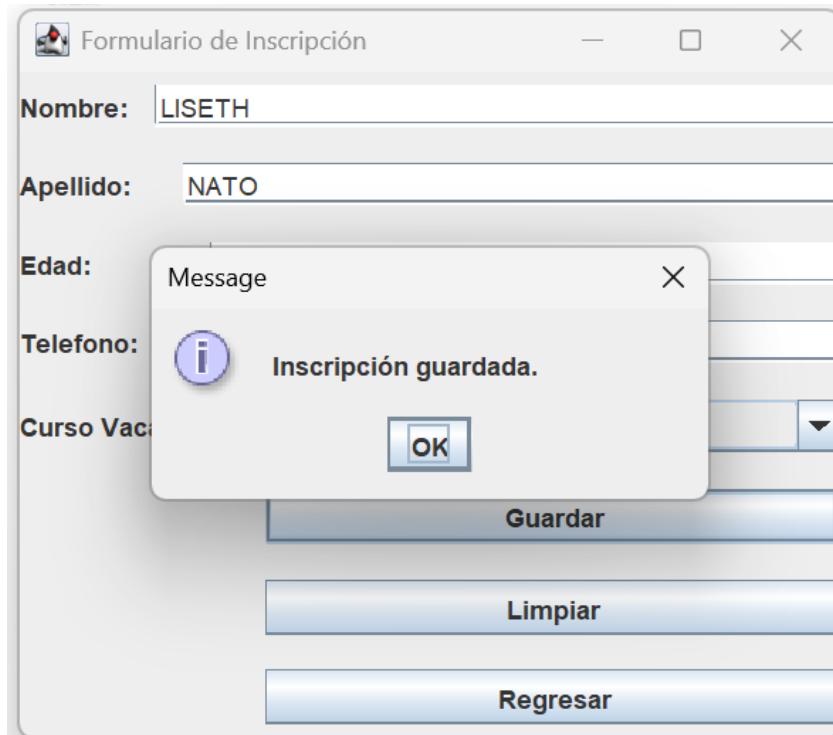


Imagen 19: Inscripción guardada

6.2. FUNCIONES DE ADMINISTRADOR

Regresamos al menú principal y seleccionamos ingreso administrador.

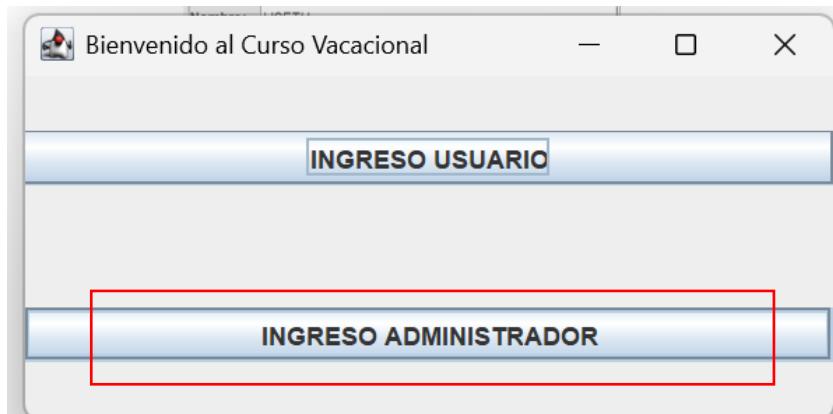


Imagen 20: Ingreso Administrador

Ingresamos el usuario y contraseña y click en ingresar

- USUARIO: Admin
- Contraseña:1234

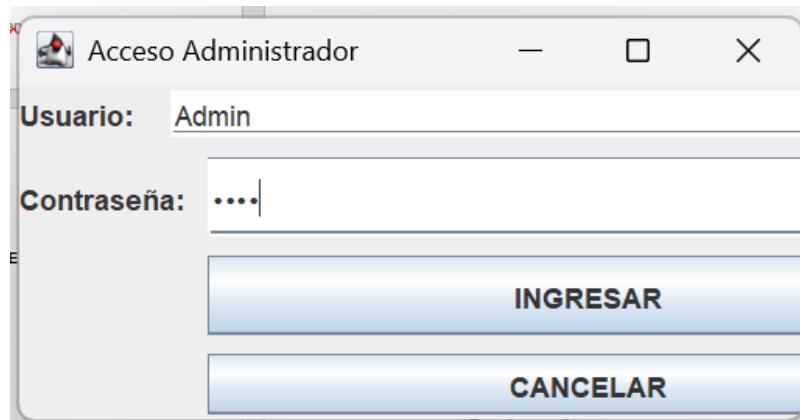


Imagen 21: Usuario y clave correctos – ingreso permitido

Si ingresamos un usuario y contraseña diferentes no nos permitirá el acceso

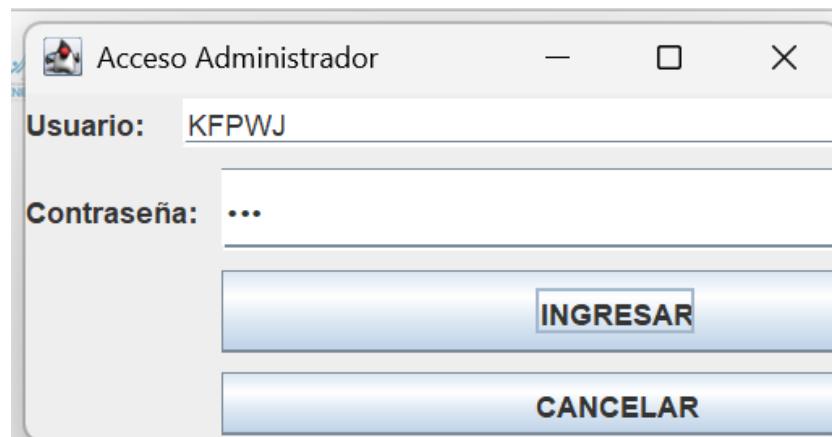


Imagen 22: Usuario y contraseña incorrectos

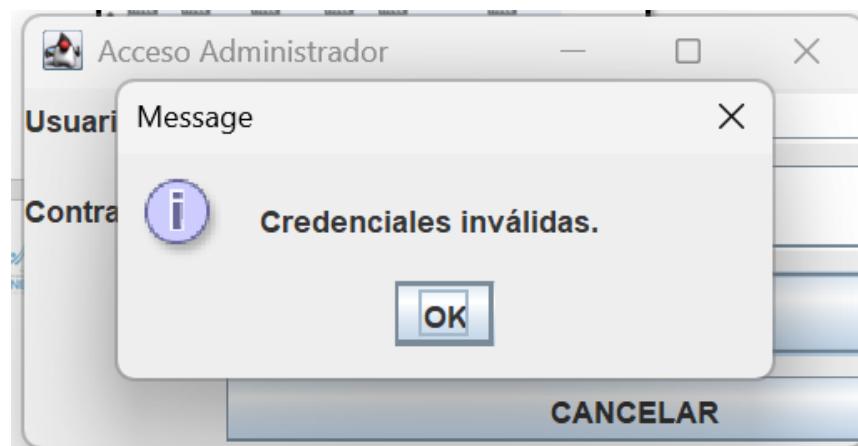


Imagen 23: Credenciales invalidas

6.3. MENU – VER INSCRIPCIONES

Se nos despliega un menú con varias opciones en este caso seleccionamos ver inscripciones

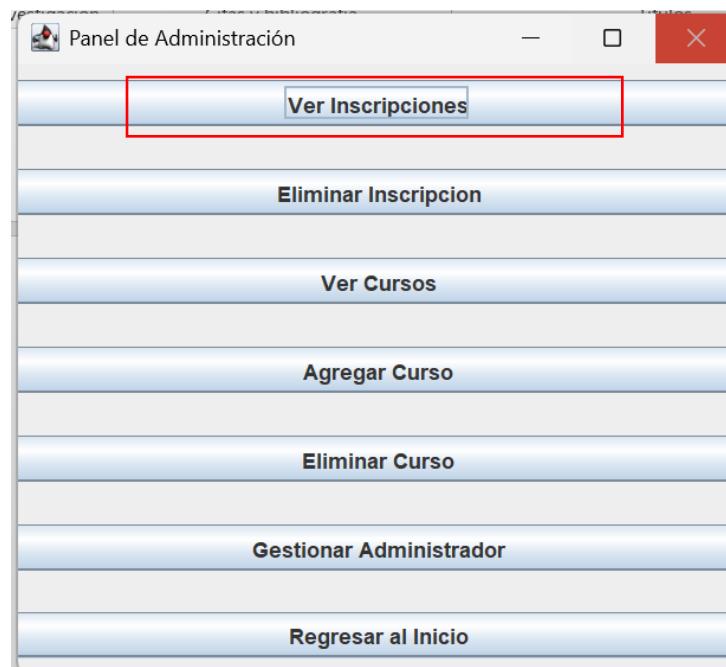


Imagen 24: Ver Inscripciones

Nos despliega un mensaje mostrándonos todos los datos que hemos ingresado



Imagen 25: Lista de inscripciones

6.3.1. ELIMINAR INSCRIPCION

Volvemos al menú y eliminaremos una inscripción en este caso será liseth nato, ingresamos el nombre y damos click en ok

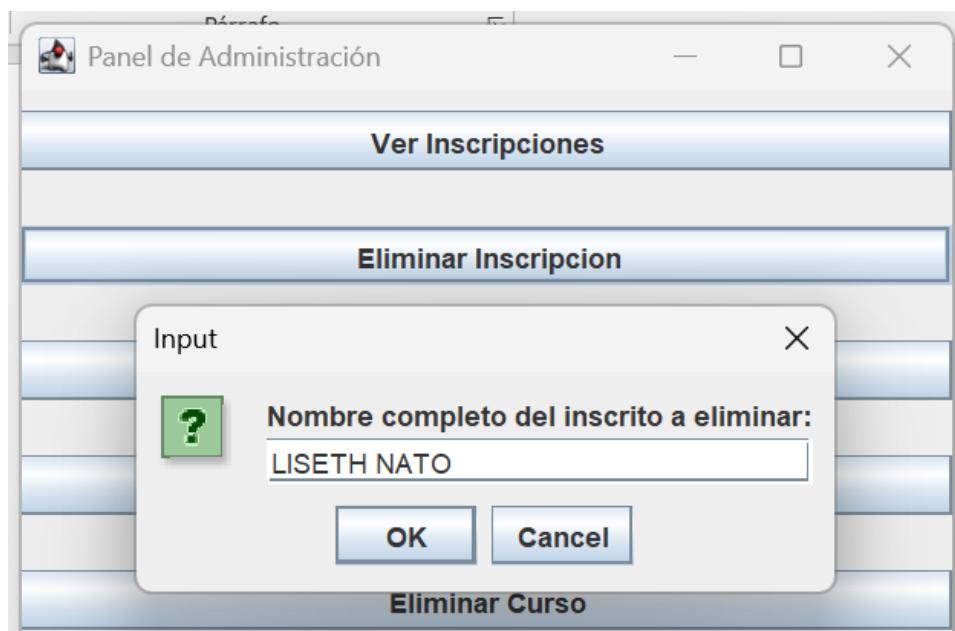


Imagen 26: Eliminar Inscripción

La inscripción ha sido eliminada

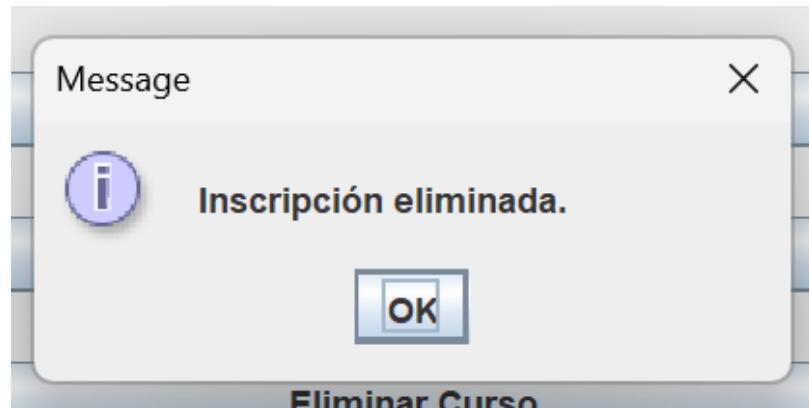


Imagen 27: Inscripción Eliminada

Verificamos, liseth nato ya no aparece en los datos



Imagen 28: verificación de la Inscripción Eliminada

6.4. CURSOS DISPONIBLES

En el menú damos click en ver cursos

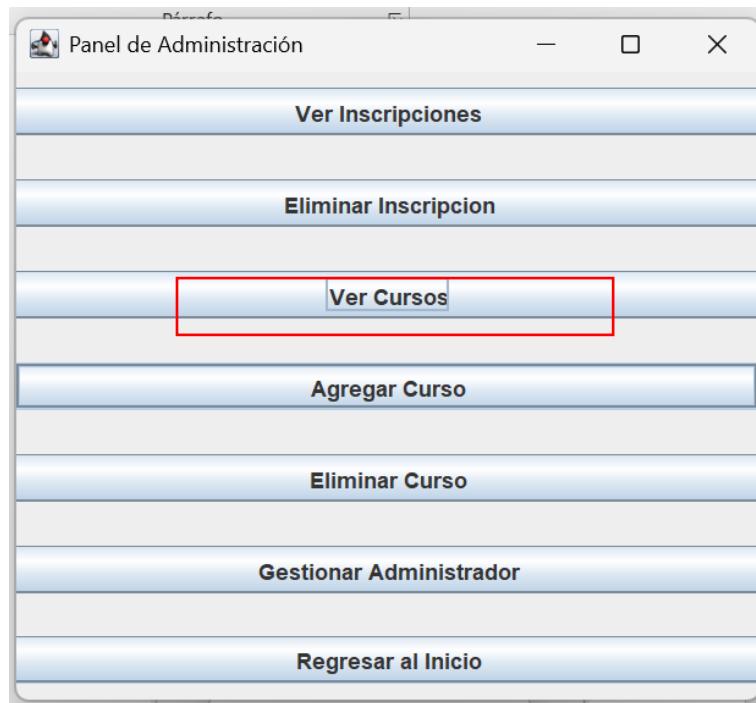


Imagen 29 Ver Cursos

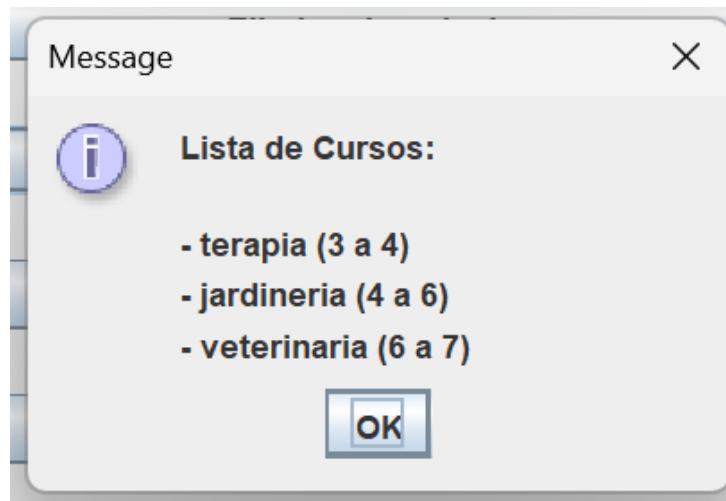


Imagen 30: Ver cursos

6.4.1. AGREGAR UN NUEVO CURSO.

Seleccionamos agregar curso.

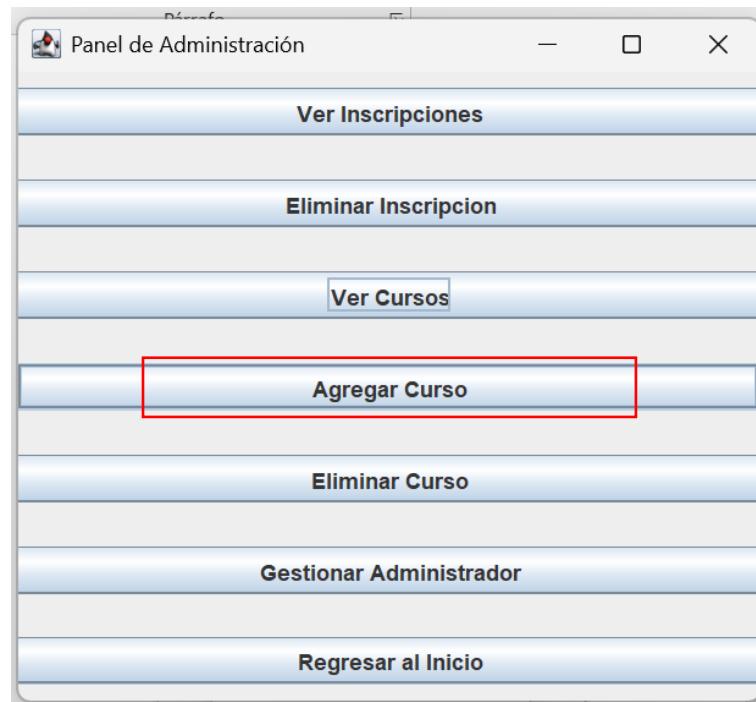


Imagen 31: Agregar curso

Llenamos los campos y click en agregar, el curso ha sido agregado

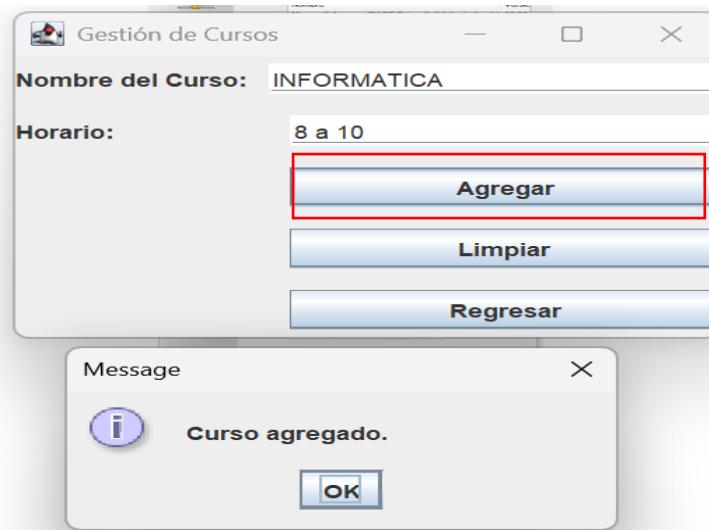


Imagen 32: Agregar y curso agregado

Verificamos el nuevo curso

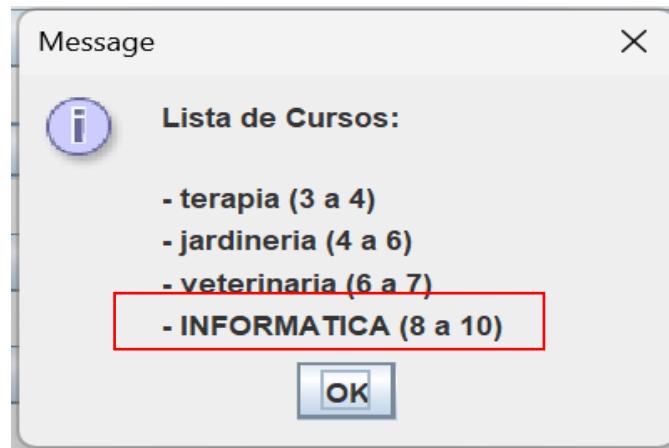


Imagen 33: Verificación – Agregar curso

6.5. ELIMINAR CURSO

Seleccionamos eliminar

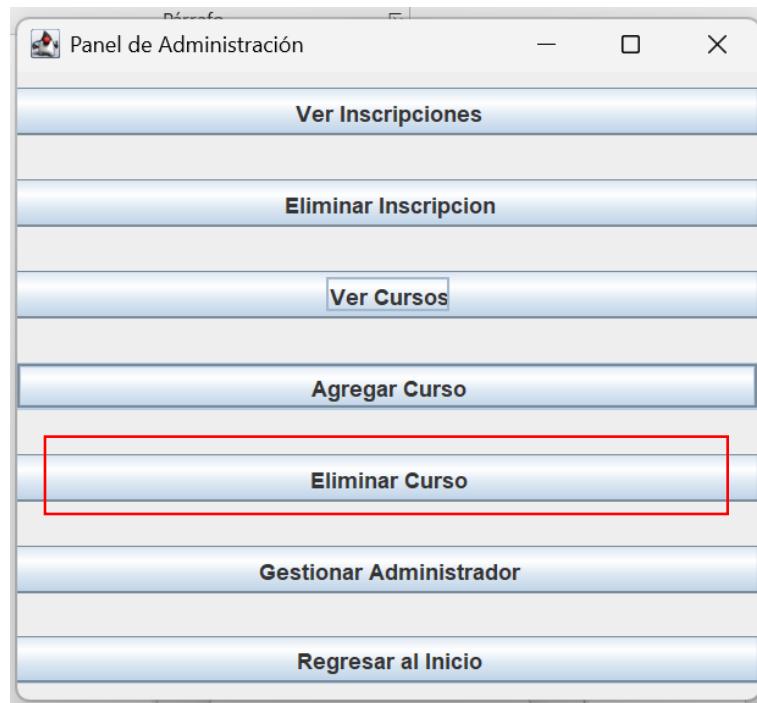


Imagen 34: Eliminar Curso

Colocamos el nombre y click en ok

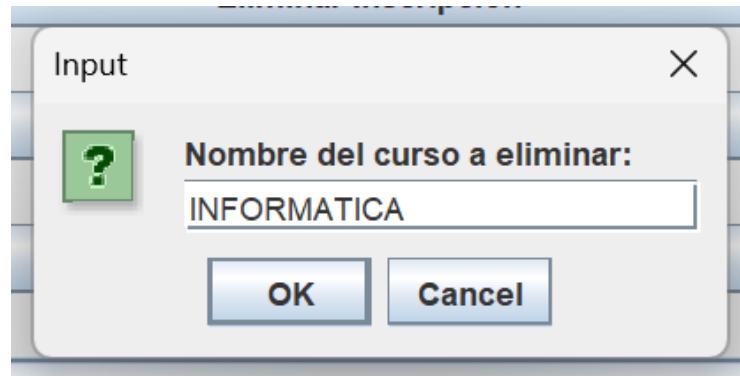


Imagen 35: Eliminar Curso INFORMATICA

El curso se ha eliminado

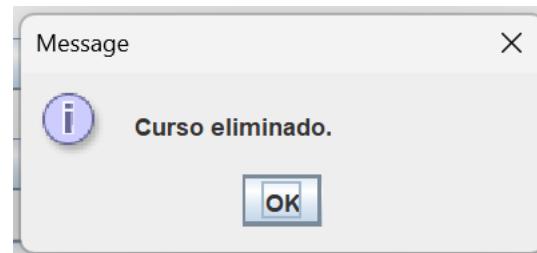


Imagen 36: Curo Eliminado

Verificamos la eliminación del curso

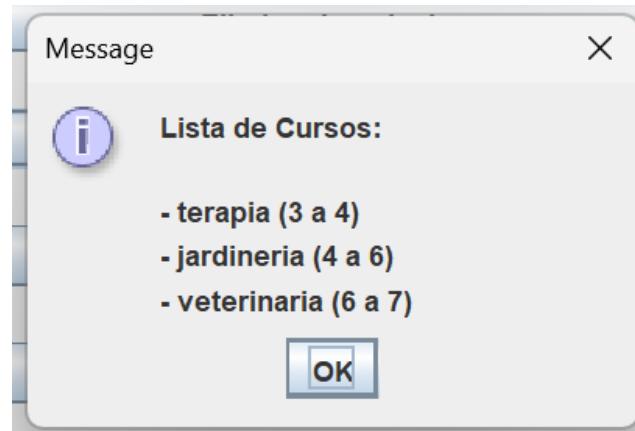


Imagen 37: Verificación – Curso Eliminado

6.6. AGREGAR UN NUEVO ADMINISTRADOR - GESTIONAR ADMINISTRADOR

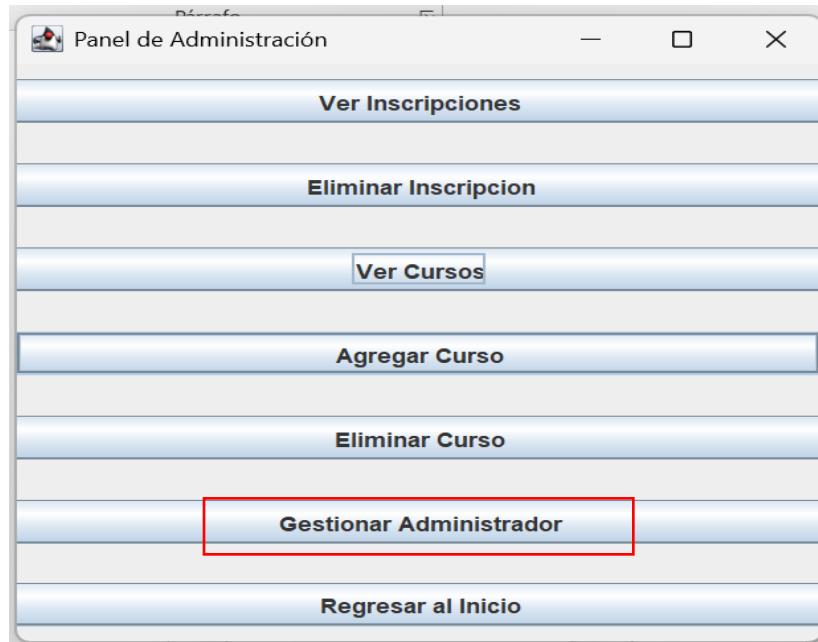


Imagen 38: Gestionar Administrador

Llenamos los datos solicitados y click en guardar

El nuevo administrador ha sido guardado

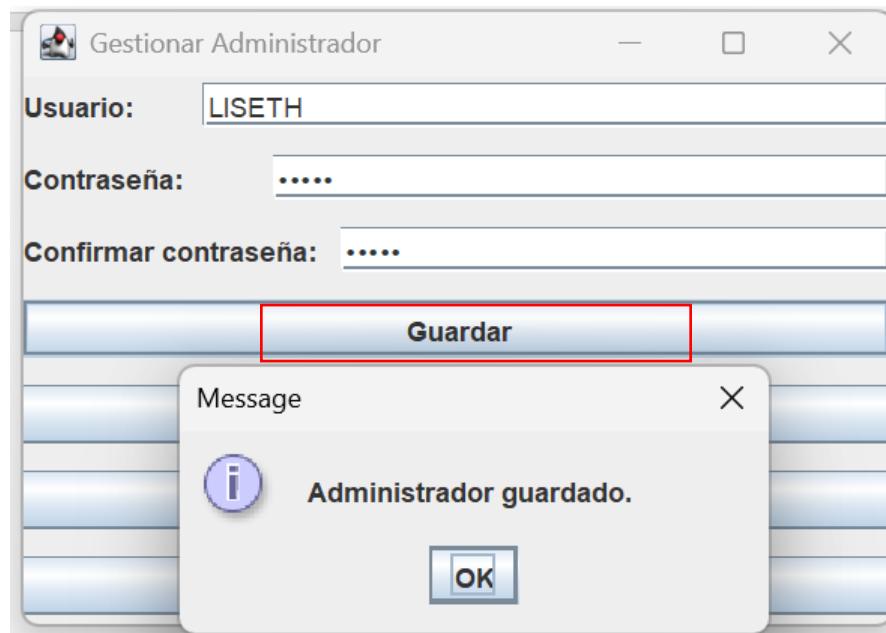


Imagen 39: Administrador guardado

Ingresamos con el nuevo usuario

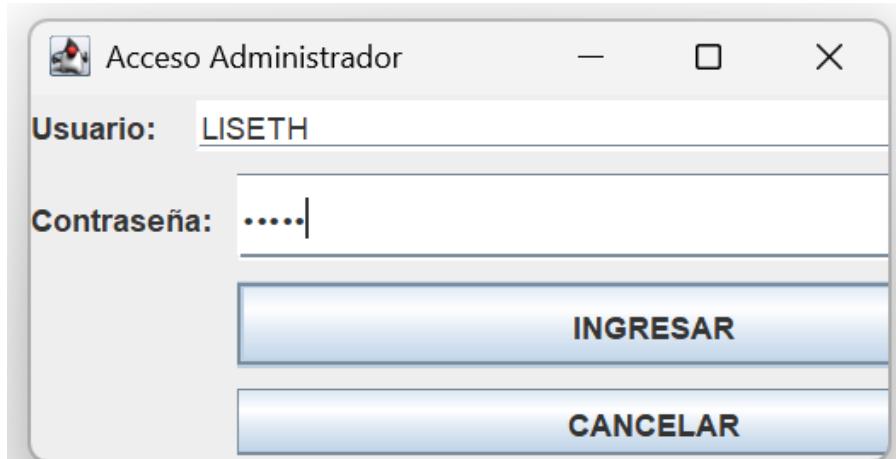


Imagen 40: Verificación – nuevo administrador

6.7. ELIMINAR EL NUEVO ADMINISTRADOR

Nos volvemos a dirigir a gestionar administrador, ingresamos el usuario y damos click en borrar.

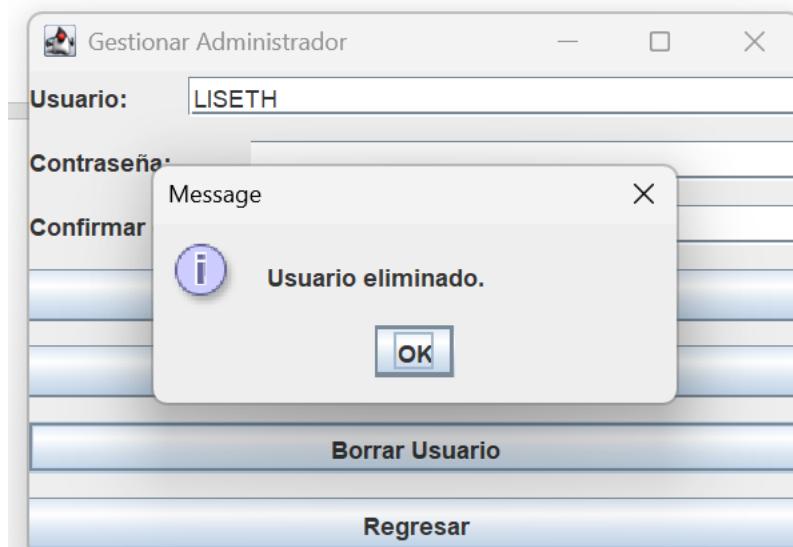


Imagen 4e1: Eliminar el nuevo administrador

VII. CREACION DE ARCHIVO .jar

- Ve a File > Project Structure o usa el atajo Ctrl + Alt + Shift + S.
- En la ventana que aparece, selecciona Artifacts en el menú izquierdo.
- Haz clic en el botón  y elige:
- JAR > From modules with 'main' class...
- Selecciona tu clase principal (Main en el paquete vista) como punto de entrada.
- Asegúrate de marcar Extract to the target JAR en la sección de dependencias.
- Pulsa OK o Apply.

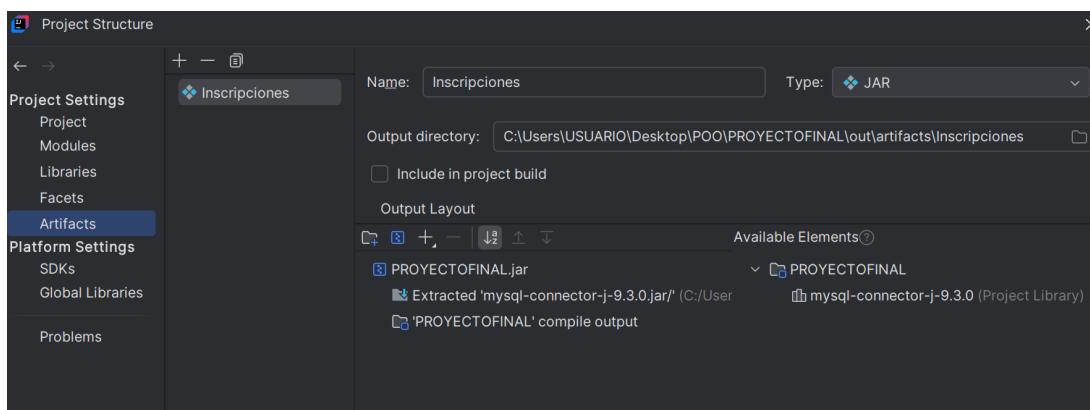


Imagen 42: Visualización de archivo .jar

- Ve a Build > Build Artifacts.
- Selecciona el artifact que creaste y haz clic en Build.
- El .jar se guardará en la carpeta out/artifacts/... dentro de tu proyecto.

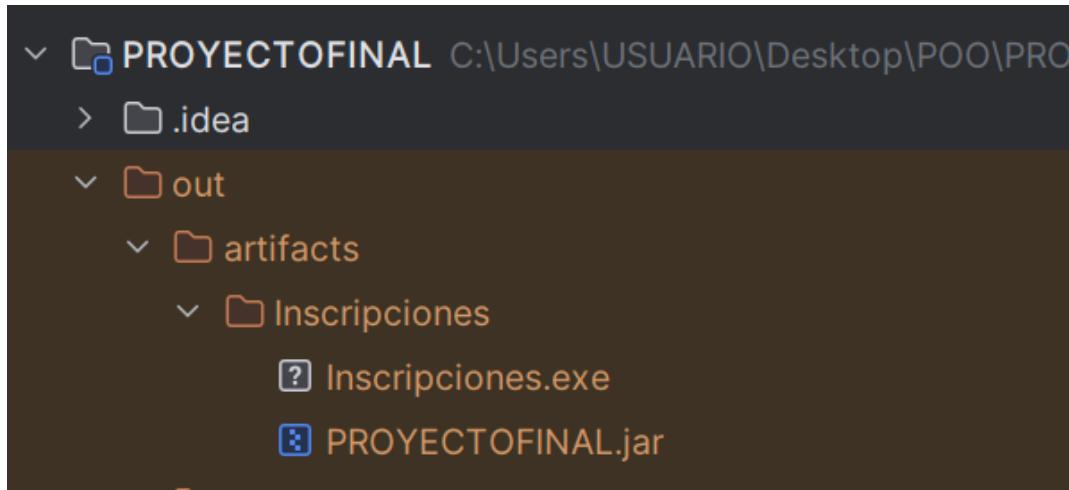


Imagen 43: Carpeta de Out

VIII. CREACION DE ARCHIVO .exe

Abre Launch4j y completa los siguientes campos:

- Output file: Ruta donde se guardará tu .exe.
- Jar: Ruta de tu archivo .jar.
- Icon (opcional): Selecciona un archivo .ico si quieres personalizar el ícono del .exe.
- Don't wrap the jar, launch only: Déjalo desmarcado.

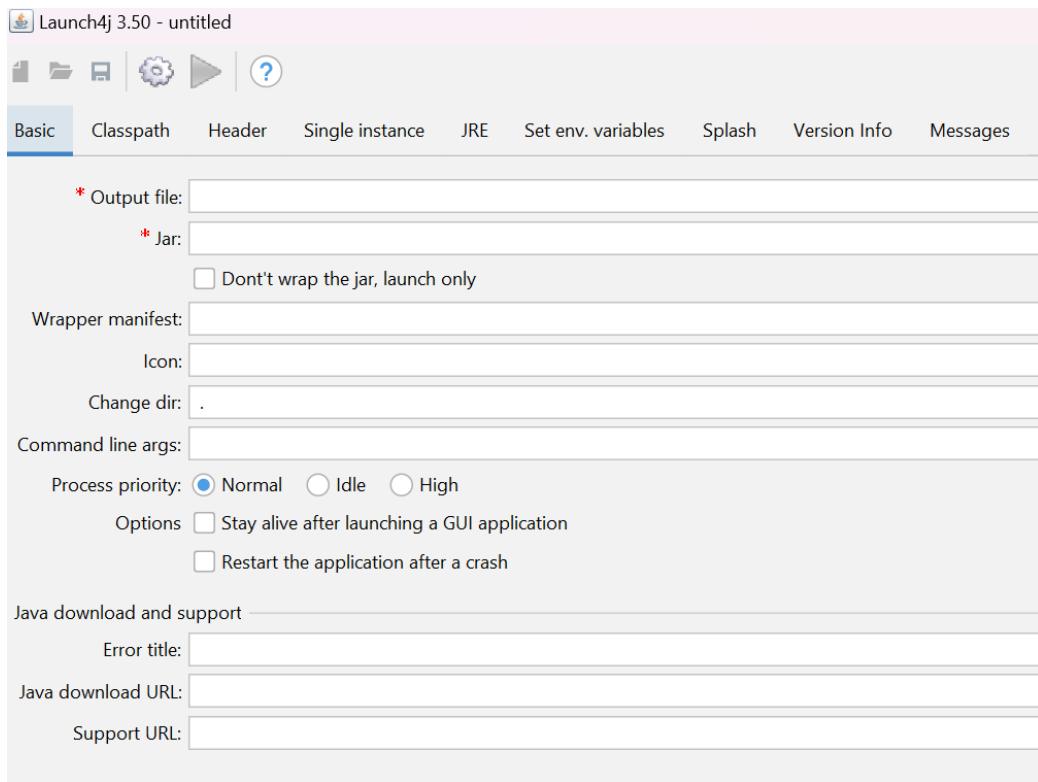


Imagen 44: Visualización de Launch4j

- Marca Custom classpath.
- En Main class, escribe el nombre completo de tu clase principal

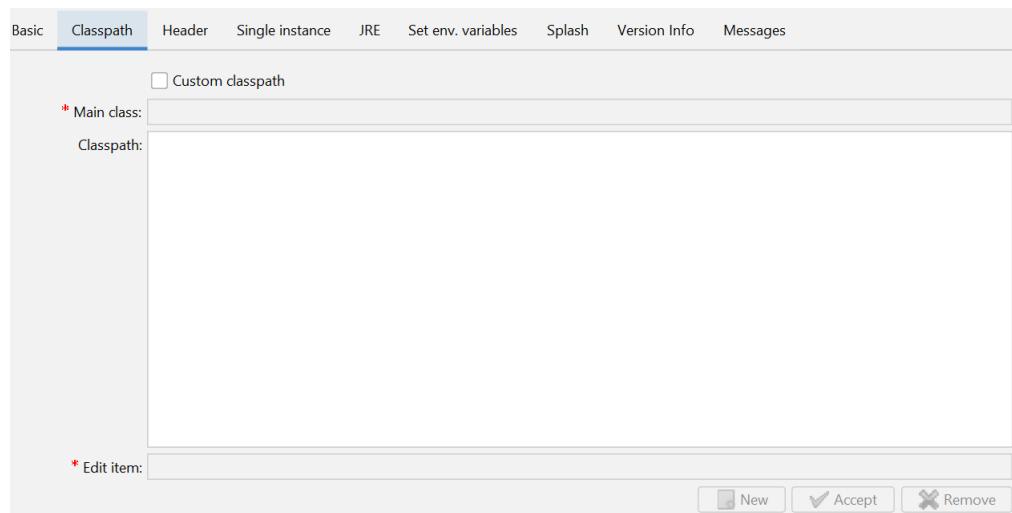


Imagen 45: Visualización de Classpath

- Min JRE version: Especifica la versión mínima requerida (por ejemplo, 1.8.0).
- Si quieras que funcione sin tener Java instalado, puedes incluir una carpeta runtime con el JRE y poner ./runtime en Bundled JRE path
- Haz clic en el ícono de engranaje () en la parte superior.
- Si todo está bien configurado, se generará el .exe en la ruta que especificaste.



Imagen 46: Icono de Inscripciones.exe

Entonces ya quedó nuestro archivo listo para ejecutarse en cualquier computadora.

IX. CONCLUSIONES

Hemos implementado con éxito el método CRUD:

- Se logró manipular datos de cursos y usuarios a través de operaciones básicas, facilitando una interacción dinámica entre la interfaz gráfica y la base de datos.
- Al aplicar la operación **Create**, los cursos y las inscripciones pudieron agregarse a la base de datos mediante formularios intuitivos, lo que mejora la experiencia del usuario.
- La operación **Read** permitió mostrar inscripciones y cursos registrados en tiempo real, fortaleciendo la visibilidad y trazabilidad del sistema.
- La lógica detrás de **Update** está preparada para futuras ampliaciones, como modificar horarios o actualizar datos de contacto del usuario.
- A través de **Delete**, se logró eliminar registros obsoletos o incorrectos, manteniendo la integridad y relevancia del contenido.

Este proyecto nos permitió profundizar en el manejo de Java Swing y MySQL para crear un sistema funcional de gestión de inscripciones. La interacción con la base de datos nos ayudó a entender cómo conectar interfaces gráficas con sistemas de almacenamiento de datos, así como también importancia de realizar pruebas constantes para asegurar la precisión y seguridad del sistema.

El desarrollo de este sistema nos ayudó a programar correctamente, es fundamental crear una interfaz amigable para el usuario. La facilidad de navegación y la claridad en las opciones como "Ver Cursos" o "Aregar Curso" hacen la diferencia para que el sistema sea efectivo. También entendí que la seguridad y validación de datos son aspectos clave en cualquier aplicación real.

Este proyecto nos ayuda a entender cómo aplicar los principios de programación orientada a objetos en un sistema real. La organización del código, el manejo de eventos en Java Swing y la integración con SQL me permitieron adquirir un conocimiento más profundo sobre la modularidad y la gestión de datos

EL trabajar en diferentes funcionalidades, como gestionar cursos e inscripciones, nos permitió distribuir tareas y aprender unos de otros. La planificación, la división del trabajo y la revisión conjunta fueron fundamentales para el éxito del sistema. Esta experiencia reforzó mi interés en seguir trabajando en proyectos colaborativos en el futuro.

X. ENLACES DE DESCARGA DEL PROYECTO

10.1. Enlace GitHub

<https://github.com/ErickDavidCastillo/PROYECTOFINAL.git>

10.2. Enlace Video

<https://youtu.be/ackHvM8bUHY>

XI. BIBLIOGRAFIA

- Blasco, J. L. (2023, 27 octubre). Introducción a POO en Java: Objetos y clases. *OpenWebinars.net.*

<https://openwebinars.net/blog/introduccion-a-poo-en-java-objetos-y-clases/>

- W3Schools.com. (s. f.-b).

https://www.w3schools.com/java/java_classes.asp

- Blasco, J. L. (2023b, noviembre 17). Introducción a POO en Java: Herencia y polimorfismo. *OpenWebinars.net.*

<https://openwebinars.net/blog/introduccion-a-poo-en-java-herencia-y-polimorfismo/>