**UCSM Esc. Prof. de Ingeniería de Sistemas**

**Diciembre - 2024**

INFORME DE PRÁCTICAS

**LENGUAJES DE**

**PROGRAMACIÓN III**



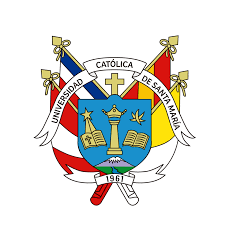
**Práctica N° 10**

Elaborado por:

FLOR VILCA GUILLERMO AARON KENAN

SUCASAIRE CUEVA, ARNOLD SAMUEL

**© IEEE 2013 The Institute of Electrical and Electronics Engineers, Inc.**

**GRUPO N° 05**

**PRÁCTICAS DE LENGUAJES DE PROGRAMACIÓN III**

Presentado por:

|  |  |  |
| --- | --- | --- |
| 2023242961  2023706451 | FLOR VILCA GUILLERMO AARON KENAN  SUCASAIRE CUEVA, ARNOLD SAMUEL | 100%  100% |
|  |  |  |

**Introduccion**

Este informe tiene como propósito presentar un análisis detallado de las técnicas avanzadas en Java Swing, centrándose en tres aspectos clave: el binding de datos, el manejo de gráficos y la integración de multimedia. A través de esta práctica, se pretende profundizar en la sincronización de datos entre el modelo de la aplicación y su interfaz gráfica, utilizando herramientas como las clases y métodos del framework Java Swing. Además, se abordará la implementación de gráficos interactivos que permiten la visualización dinámica de datos y la creación de aplicaciones gráficas más complejas y funcionales.

Asimismo, se explorará el manejo de multimedia dentro de las aplicaciones Java, donde se integran elementos como audio y video, ampliando las posibilidades de desarrollo más allá de interfaces gráficas tradicionales. A través de ejemplos prácticos, se ilustrará cómo utilizar estas herramientas para desarrollar aplicaciones interactivas y multimedia, que no solo gestionen datos de manera eficiente, sino que también ofrezcan una experiencia visual enriquecedora. Este enfoque busca consolidar los conocimientos adquiridos y su aplicación en escenarios reales de desarrollo de software.

**ACTIVIDADES**

1. Primero prueben ejecutar cada uno de los ejemplos vistos anteriormente, analicen las clases involucradas y la secuencia de pasos para ejecutar cada aplicación.

Luego creen 2 nuevas ventanas por integrante donde se realicen las siguientes acciones:

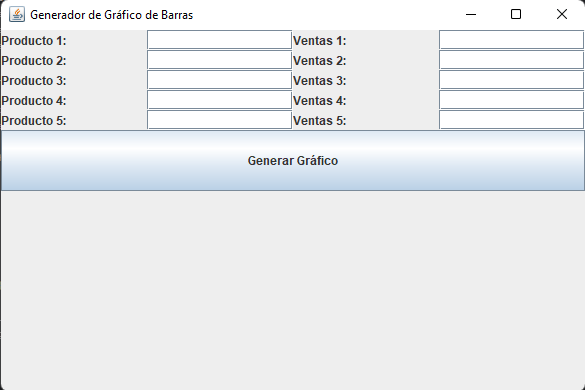
* Binding de Datos manual con Swing
* Creación de gráficos simples con Swing
* Creación de gráficos avanzados con Swing
* Creación de aplicaciones que utilicen elementos multimedia con Swing

Las ventanas pueden hacer uso combinado de una o más de las acciones vistas anteriormente.

Gráficos Avanzados con Swing:

|  |
| --- |
| import javax.swing.\*; import java.awt.\*; import java.awt.event.ActionEvent; import java.awt.event.ActionListener;  public class GAact extends JFrame {   private JTextField[] productFields;  private JTextField[] salesFields;  private BarChartPanel chartPanel;   public GAact() {  setTitle("Generador de Gráfico de Barras");  setLayout(new BorderLayout());  setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  setSize(600, 400);   JPanel inputPanel = new JPanel(new GridLayout(5, 2));   productFields = new JTextField[5];  salesFields = new JTextField[5];   for (int i = 0; i < 5; i++) {  inputPanel.add(new JLabel("Producto " + (i + 1) + ":"));  productFields[i] = new JTextField(10);  inputPanel.add(productFields[i]);   inputPanel.add(new JLabel("Ventas " + (i + 1) + ":"));  salesFields[i] = new JTextField(10);  inputPanel.add(salesFields[i]);  }   JButton generateButton = new JButton("Generar Gráfico");  generateButton.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  generateGraph();  }  });   chartPanel = new BarChartPanel();  chartPanel.setPreferredSize(new Dimension(600, 200));   add(inputPanel, BorderLayout.*NORTH*);  add(generateButton, BorderLayout.*CENTER*);  add(chartPanel, BorderLayout.*SOUTH*);   setVisible(true);  }   private void generateGraph() {  String[] products = new String[5];  int[] sales = new int[5];   for (int i = 0; i < 5; i++) {  products[i] = (productFields[i].getText() != null && !productFields[i].getText().trim().isEmpty()) ?  productFields[i].getText() : "Producto " + (i + 1);   try {  sales[i] = Integer.*parseInt*(salesFields[i].getText());  } catch (NumberFormatException e) {  sales[i] = 0;  }  }   chartPanel.setProducts(products);  chartPanel.setSales(sales);  chartPanel.repaint();  }   public static void main(String[] args) {  new GAact();  }   class BarChartPanel extends JPanel {  private String[] products;  private int[] sales;   public BarChartPanel() {  products = new String[5];  sales = new int[5];  }   public void setProducts(String[] products) {  this.products = products;  }   public void setSales(int[] sales) {  this.sales = sales;  }   @Override  protected void paintComponent(Graphics g) {  super.paintComponent(g);   Graphics2D g2d = (Graphics2D) g;  g2d.setRenderingHint(RenderingHints.*KEY\_ANTIALIASING*, RenderingHints.*VALUE\_ANTIALIAS\_ON*);   g2d.setColor(Color.*BLACK*);   int barWidth = 50;  int space = 20;  int startX = 50;   for (int i = 0; i < sales.length; i++) {  int barHeight = sales[i];  g2d.fillRect(startX + (i \* (barWidth + space)), getHeight() - barHeight - 50, barWidth, barHeight);  }   g2d.setColor(Color.*BLACK*);  for (int i = 0; i < products.length; i++) {  String productName = products[i];  if (productName != null && !productName.trim().isEmpty()) {  g2d.drawString(productName, startX + (i \* (barWidth + space)) + (barWidth / 4), getHeight() - 30);  }  }  }  } } |

Al ejecutar este código tenemos:



Podemos ingresar los datos como nombre de producto y sus respectivas ventas:

Tabla

Descripción generada automáticamente

Creación de aplicaciones que utilicen elementos multimedia con Swing:

|  |
| --- |
| import javax.swing.\*; import java.awt.\*; import javax.sound.sampled.\*; import java.io.File; import java.io.IOException;  public class EMact extends JFrame {   private Clip clip;   public EMact() {  setTitle("Reproductor de Audio");  setLayout(new FlowLayout());  setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  setSize(300, 250);   JButton playButton1 = new JButton("Reproducir Audio 1");  playButton1.addActionListener(e -> playAudio("C:\\Users\\Usuario\\IdeaProjects\\Sesion10\\audio1.wav"));   JButton playButton2 = new JButton("Reproducir Audio 2");  playButton2.addActionListener(e -> playAudio("C:\\Users\\Usuario\\IdeaProjects\\Sesion10\\audio2.wav"));   JButton playButton3 = new JButton("Reproducir Audio 3");  playButton3.addActionListener(e -> playAudio("C:\\Users\\Usuario\\IdeaProjects\\Sesion10\\audio3.wav"));   JButton stopButton = new JButton("Detener Audio");  stopButton.addActionListener(e -> stopAudio());   add(playButton1);  add(playButton2);  add(playButton3);  add(stopButton);   setVisible(true);  }   private void playAudio(String filePath) {  try {  File audioFile = new File(filePath);  AudioInputStream audioStream = AudioSystem.*getAudioInputStream*(audioFile);  clip = AudioSystem.*getClip*();  clip.open(audioStream);  clip.start();  } catch (UnsupportedAudioFileException | IOException | LineUnavailableException ex) {  ex.printStackTrace();  }  }   private void stopAudio() {  if (clip != null && clip.isRunning()) {  clip.stop();  }  }   public static void main(String[] args) {  new EMact();  } } |

Ejecutando el código tenemos:



Con sus respectivos archivos de audio en formato WAV en la ruta especificada en el código:

Captura de pantalla de un celular

Descripción generada automáticamente

**EJERCICIOS PROPUESTOS**

1. Aplicaciones propuestas:
   1. Crea una aplicación para registrar y visualizar la temperatura diaria de una semana mediante un gráfico de líneas. La aplicación debe permitir al usuario ingresar la temperatura para cada día de la semana y mostrar una línea que conecte los puntos correspondientes a cada día en el gráfico. Diseñar una interfaz con: Siete campos de texto para ingresar la temperatura de cada día (de lunes a domingo). Un botón "Mostrar Gráfico" que dibuje el gráfico de líneas. Crear un panel donde se dibuje el gráfico de líneas, con puntos para cada día y una línea que conecte los puntos. Al presionar "Mostrar Gráfico", el gráfico debe actualizarse y mostrar las temperaturas para cada día.

|  |
| --- |
| import javax.swing.\*; import java.awt.\*; import java.util.ArrayList;  public class tempEJ {  public static void main(String[] args) {  JFrame frame = new JFrame("Temperature Graph");  frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  frame.setSize(600, 400);   // Panel para ingresar datos  JPanel inputPanel = new JPanel(new GridLayout(8, 2));  JTextField[] tempFields = new JTextField[7];  String[] days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};   for (int i = 0; i < 7; i++) {  inputPanel.add(new JLabel(days[i] + ":"));  tempFields[i] = new JTextField();  inputPanel.add(tempFields[i]);  }   JButton btnShowGraph = new JButton("Show Graph");  inputPanel.add(btnShowGraph);   // Panel para el gráfico  GraphPanel graphPanel = new GraphPanel();   frame.setLayout(new BorderLayout());  frame.add(inputPanel, BorderLayout.*WEST*);  frame.add(graphPanel, BorderLayout.*CENTER*);   btnShowGraph.addActionListener(e -> {  ArrayList<Integer> temperatures = new ArrayList<>();  for (JTextField field : tempFields) {  try {  temperatures.add(Integer.*parseInt*(field.getText()));  } catch (NumberFormatException ex) {  temperatures.add(0);  }  }  graphPanel.setTemperatures(temperatures);  });   frame.setVisible(true);  } }  class GraphPanel extends JPanel {  private ArrayList<Integer> temperatures = new ArrayList<>();   public void setTemperatures(ArrayList<Integer> temperatures) {  this.temperatures = temperatures;  repaint(); // Redibuja el gráfico  }   @Override  protected void paintComponent(Graphics g) {  super.paintComponent(g);  if (temperatures.isEmpty()) return;   Graphics2D g2 = (Graphics2D) g;  g2.setStroke(new BasicStroke(2));  g2.setColor(Color.*BLACK*);   int width = getWidth();  int height = getHeight();  int padding = 50;  int pointWidth = 8;   int xScale = (width - 2 \* padding) / (temperatures.size() - 1);  int maxTemp = temperatures.stream().max(Integer::*compare*).orElse(1);  int minTemp = temperatures.stream().min(Integer::*compare*).orElse(0);  int yScale = (height - 2 \* padding) / (maxTemp - minTemp);   // Dibuja ejes  g2.drawLine(padding, height - padding, width - padding, height - padding); // Eje X  g2.drawLine(padding, height - padding, padding, padding); // Eje Y   // Dibuja líneas y puntos  for (int i = 0; i < temperatures.size() - 1; i++) {  int x1 = padding + i \* xScale;  int y1 = height - padding - (temperatures.get(i) - minTemp) \* yScale;  int x2 = padding + (i + 1) \* xScale;  int y2 = height - padding - (temperatures.get(i + 1) - minTemp) \* yScale;  g2.drawLine(x1, y1, x2, y2);  g2.fillOval(x1 - pointWidth / 2, y1 - pointWidth / 2, pointWidth, pointWidth);  }   // Dibuja último punto  int lastX = padding + (temperatures.size() - 1) \* xScale;  int lastY = height - padding - (temperatures.get(temperatures.size() - 1) - minTemp) \* yScale;  g2.fillOval(lastX - pointWidth / 2, lastY - pointWidth / 2, pointWidth, pointWidth);  } } |

Ejecutando el código tenemos un panel donde ingresamos los datos y un botón para generar el gráfico:

Tabla

Descripción generada automáticamente

Gráfico de líneas

Descripción generada automáticamente

Imagen que contiene Gráfico

Descripción generada automáticamente

* 1. Crea una aplicación de Java Swing que permita al usuario reproducir, pausar y reanudar una pista de música. La aplicación debe tener botones de "Reproducir", "Pausar" y "Reanudar" que controlen la reproducción del audio. Diseña una interfaz con tres botones: "Reproducir", "Pausar" y "Reanudar". Al presionar "Reproducir", el audio comienza a reproducirse desde el inicio. Al presionar "Pausar", el audio se detiene temporalmente, y al presionar "Reanudar" continúa desde donde se detuvo. Utiliza Clip de javax.sound.sampled para cargar y controlar la reproducción de audio..

|  |
| --- |
| import javax.swing.\*; import javax.sound.sampled.\*; import java.io.File; import java.io.IOException; import java.awt.\*;  public class audioEJ extends JFrame {   private Clip clip;  private AudioInputStream audioStream;  private long clipTimePosition;  private JTextField statusField;   public audioEJ() {  setTitle("Reproductor de Audio");  setLayout(new BorderLayout());  setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  setSize(400, 300);   JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.*CENTER*, 10, 10));   JButton playButton = new JButton("Reproducir");  playButton.addActionListener(e -> {  playAudio("C:\\Users\\Usuario\\IdeaProjects\\Sesion10\\pista.wav");  updateStatus("Reproduciendo");  });   JButton pauseButton = new JButton("Pausar");  pauseButton.addActionListener(e -> {  pauseAudio();  updateStatus("Pausado");  });   JButton resumeButton = new JButton("Reanudar");  resumeButton.addActionListener(e -> {  resumeAudio();  updateStatus("Reanudado");  });   buttonPanel.add(playButton);  buttonPanel.add(pauseButton);  buttonPanel.add(resumeButton);   statusField = new JTextField("Estado: Detenido");  statusField.setEditable(false);  statusField.setHorizontalAlignment(SwingConstants.*CENTER*);   add(statusField, BorderLayout.*NORTH*);  add(buttonPanel, BorderLayout.*CENTER*);   setVisible(true);  }   private void playAudio(String filePath) {  try {  File audioFile = new File(filePath);  audioStream = AudioSystem.*getAudioInputStream*(audioFile);  clip = AudioSystem.*getClip*();  clip.open(audioStream);  clip.start();  clipTimePosition = 0;  } catch (UnsupportedAudioFileException | IOException | LineUnavailableException ex) {  ex.printStackTrace();  }  }   private void pauseAudio() {  if (clip != null && clip.isRunning()) {  clipTimePosition = clip.getMicrosecondPosition();  clip.stop();  }  }   private void resumeAudio() {  if (clip != null && !clip.isRunning()) {  clip.setMicrosecondPosition(clipTimePosition);  clip.start();  }  }   private void updateStatus(String status) {  statusField.setText("Estado: " + status);  }   public static void main(String[] args) {  new audioEJ();  } } |

Ejecutando el código tenemos 3 botones y un TextField que nos indica el estado de la pista:

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

1. Deben crear una aplicación de temática libre que utilice el patrón modelo-vista-controlador (mvc) y la librería swing para la interfaz gráfica. La aplicación debe interactuar con una base de datos para gestionar y almacenar información de acuerdo con las necesidades de la aplicación elegida, tomando en cuenta las siguientes consideraciones:
   1. Antes de comenzar a programar, deberán realizar un diseño en papel (mockup) de la interfaz gráfica de su aplicación. Este diseño debe incluir las pantallas principales de la aplicación, como los formularios, tablas o menús, y debe mostrar cómo será la disposición de los elementos en la ventana.
   2. El diseño en papel será evaluado para asegurar que tienen una visión clara de la estructura visual y de la interacción con el usuario.
   3. La interfaz debe ser desarrollada utilizando la librería swing de java. La interfaz debe ser clara, fácil de usar y con una navegación lógica y amigable.
   4. Deben integrar una base de datos para almacenar y gestionar la información de tu aplicación. Puede ser una base de datos local como sqlite o una base de datos mysql.
   5. La aplicación debe seguir el patrón modelo-vista-controlador (mvc) para asegurar una correcta separación de responsabilidades.

* AsignaturaDAO.java

|  |
| --- |
| package CRUD;  import java.sql.\*;  import javax.swing.\*;  import java.util.ArrayList;  import java.util.List;  public class AsignaturaDAO {        // METODO -----      // agregar una nueva asignatura a la base de datos (curso)      public boolean agregarAsignatura(String nombre) {          String sql = "INSERT INTO Asignaturas(nombre) VALUES(?)";          try (Connection conn = ConexionBD.conectar(); // Conexion a la base de datos               PreparedStatement pstmt = conn.prepareStatement(sql)) { // Preparar consulta              pstmt.setString(1, nombre); // Asignar el nombre de la asignatura 1:indice parametro              pstmt.executeUpdate(); // Ejecuta              return true;          } catch (SQLException e) { //error              JOptionPane.showMessageDialog(null, "Error al agregar asignatura: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);              return false;          }      }      public boolean asignaturaExiste(String nombre) {          String sql = "SELECT COUNT(\*) FROM Asignaturas WHERE nombre = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, nombre);              ResultSet rs = pstmt.executeQuery();              if (rs.next()) {                  return rs.getInt(1) > 0; // Si el conteo es mayor a 0, la asignatura existe              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al verificar asignatura: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return false; // Retorna falso si ocurre algún error o no existe la asignatura      }        // listar las asignaturas registradas      public List<String> listarAsignaturas() {          List<String> asignaturas = new ArrayList<>(); // Lista para almacenar resultados          String sql = "SELECT \* FROM Asignaturas";          try (Connection conn = ConexionBD.conectar(); // Conexion a la base de datos               Statement stmt = conn.createStatement(); // Crear el Statement -> consultar               ResultSet rs = stmt.executeQuery(sql)) { // Ejecutar la consulta              while (rs.next()) {                  // Agregar asignatura con ID y nombre a la lista //nombre                  asignaturas.add(rs.getInt("id") + " - " + rs.getString("nombre"));              }          } catch (SQLException e) {              // Mostrar error en un cuadro de dialogo              JOptionPane.showMessageDialog(null, "Error al listar asignaturas: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return asignaturas; // Retornar la lista de asignaturas      }  } |

* CalificacionDAO.java

|  |
| --- |
| package CRUD;  import java.sql.\*;  import javax.swing.\*;  import java.util.ArrayList;  import java.util.List;  public class CalificacionDAO {      private final List<Observer> observers = new ArrayList<>(); // Lista de observadores      // METODOS --------      // registrar un observador      public void addObserver(Observer observer) {          observers.add(observer);      }      // notificar a los observadores      private void notifyObservers(String mensaje) {          for (Observer observer : observers) {              observer.update(mensaje); // Notificar con el mensaje          }      }      public boolean agregarCalificacion(int estudianteId, int asignaturaId, double f1, double f2, double f3, double promedio) {          String sql = "INSERT INTO Calificaciones(estudiante\_id, asignatura\_id, F1, F2, F3, promedio) VALUES(?, ?, ?, ?, ?, ?)";          try (Connection conn = ConexionBD.conectar(); // Conectar a la base de datos               PreparedStatement pstmt = conn.prepareStatement(sql)) { // Preparar la consulta              // Asignar parámetros a la consulta              pstmt.setInt(1, estudianteId);              pstmt.setInt(2, asignaturaId);              pstmt.setDouble(3, f1);              pstmt.setDouble(4, f2);              pstmt.setDouble(5, f3);              pstmt.setDouble(6, promedio);                // Ejecutar la consulta              pstmt.executeUpdate();                // Notificar a los observadores del cambio              String mensaje = "Nueva calificación registrada: \n" +                               "Estudiante ID: " + estudianteId + "\n" +                               "Asignatura ID: " + asignaturaId + "\n" +                               "F1: " + f1 + ", F2: " + f2 + ", F3: " + f3 + ", Promedio: " + promedio;              notifyObservers(mensaje);                return true;          } catch (SQLException e) { // Manejar errores de SQL              JOptionPane.showMessageDialog(null,                  "Error al registrar las calificaciones: " + e.getMessage(),                  "Error",                  JOptionPane.ERROR\_MESSAGE);              return false;          }      }      // Actualizar una calificación existente      public boolean actualizarCalificacion(int estudianteId, int asignaturaId, double f1, double f2, double f3, double promedio) {          String sql = "UPDATE Calificaciones SET F1 = ?, F2 = ?, F3 = ?, promedio = ? WHERE estudiante\_id = ? AND asignatura\_id = ?";          try (Connection conn = ConexionBD.conectar(); // Conectar a la base de datos               PreparedStatement pstmt = conn.prepareStatement(sql)) { // Preparar la consulta              // Asignar parámetros a la consulta              pstmt.setDouble(1, f1);              pstmt.setDouble(2, f2);              pstmt.setDouble(3, f3);              pstmt.setDouble(4, promedio);              pstmt.setInt(5, estudianteId);              pstmt.setInt(6, asignaturaId);                // Ejecutar la consulta              int rowsAffected = pstmt.executeUpdate();              if (rowsAffected > 0) {                  // Notificar a los observadores del cambio                  String mensaje = "Calificación actualizada: \n" +                                   "Estudiante ID: " + estudianteId + "\n" +                                   "Asignatura ID: " + asignaturaId + "\n" +                                   "F1: " + f1 + ", F2: " + f2 + ", F3: " + f3 + ", Promedio: " + promedio;                  notifyObservers(mensaje);                  return true;              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null,                  "Error al actualizar las calificaciones: " + e.getMessage(),                  "Error",                  JOptionPane.ERROR\_MESSAGE);          }          return false;      }      // Verificar si un estudiante ya tiene calificación en una asignatura      public boolean tieneCalificacion(int estudianteId, int asignaturaId) {          String sql = "SELECT COUNT(\*) FROM Calificaciones WHERE estudiante\_id = ? AND asignatura\_id = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setInt(1, estudianteId);              pstmt.setInt(2, asignaturaId);              ResultSet rs = pstmt.executeQuery();              return rs.next() && rs.getInt(1) > 0;          } catch (SQLException e) {              JOptionPane.showMessageDialog(null,                  "Error al verificar calificación: " + e.getMessage(),                  "Error",                  JOptionPane.ERROR\_MESSAGE);          }          return false;      }      // Calcular el promedio de un estudiante por DNI      public double calcularPromedio(String dni) {          String sql = "SELECT AVG(c.calificacion) AS promedio FROM Calificaciones c INNER JOIN Student s ON c.estudiante\_id = s.id WHERE s.dni = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, dni);              ResultSet rs = pstmt.executeQuery();              if (rs.next()) {                  return rs.getDouble("promedio");              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al calcular promedio: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return 0.0;      }      // Obtener historial de calificaciones de un estudiante por DNI      public List<String[]> historialCalificaciones(String dni) {          List<String[]> historial = new ArrayList<>();          String sql = "SELECT a.nombre AS asignatura, c.F1, c.F2, c.F3, c.promedio " +                      "FROM Calificaciones c " +                      "JOIN Asignaturas a ON c.asignatura\_id = a.id " +                      "JOIN Student s ON c.estudiante\_id = s.id " +                      "WHERE s.dni = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, dni);              ResultSet rs = pstmt.executeQuery();              while (rs.next()) {                  String[] registro = {                      rs.getString("asignatura"),                      String.valueOf(rs.getDouble("F1")),                      String.valueOf(rs.getDouble("F2")),                      String.valueOf(rs.getDouble("F3")),                      String.valueOf(rs.getDouble("promedio"))                  };                  historial.add(registro);              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al obtener historial: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return historial;      }      // Obtener historial de calificaciones por asignatura      public List<String[]> historialPorAsignatura(int asignaturaId) {          List<String[]> historial = new ArrayList<>();          String sql = "SELECT s.nombre AS estudiante, c.F1, c.F2, c.F3, c.promedio " +                       "FROM Calificaciones c " +                       "JOIN Student s ON c.estudiante\_id = s.id " +                       "WHERE c.asignatura\_id = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setInt(1, asignaturaId);              ResultSet rs = pstmt.executeQuery();              while (rs.next()) {                  String[] registro = {                      rs.getString("estudiante"),                      String.valueOf(rs.getDouble("F1")),                      String.valueOf(rs.getDouble("F2")),                      String.valueOf(rs.getDouble("F3")),                      String.valueOf(rs.getDouble("promedio"))                  };                  historial.add(registro);              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al obtener historial: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return historial;      }  } |

* ConexionBD.java

|  |
| --- |
| package CRUD;  /\*  --------------------------------- NOMBRE BASE DE DATOS -------------------------------------  CalificacionesAcademicas.bd  --------------------------------------------------------------------------------------------  ------------------------------ ELIMINAR BASE DE DATOS ---------------------------------------  E:  CD UCSM\4. IV Semestre\Lenguaje de Programación - 05\Practica 10  del CalificacionesAcademicas.bd  ---------------------------------------------------------------------------------------------  -------------------------- ELIMINAR UNA TABLA DE BASE DE DATOS ------------------------------  E:  CD UCSM\4. IV Semestre\Lenguaje de Programación - 05\Practica 10  sqlite3 CalificacionesAcademicas.bd  DROP TABLE IF EXISTS Name\_table;  ----------------------------------------------------------------------------------------------  -----------------------------CREAR UNA TABLA DE BASE DE DATOS --------------------------------  E:  CD UCSM\4. IV Semestre\Lenguaje de Programación - 05\Practica 10  sqlite3 CalificacionesAcademicas.bd  CREATE TABLE Calificaciones (      id INTEGER PRIMARY KEY AUTOINCREMENT,      estudiante\_id INTEGER NOT NULL,      asignatura\_id INTEGER NOT NULL,      F1 REAL NOT NULL CHECK (F1 >= 0 AND F1 <= 20),      F2 REAL NOT NULL CHECK (F2 >= 0 AND F2 <= 20),      F3 REAL NOT NULL CHECK (F3 >= 0 AND F3 <= 20),      promedio REAL NOT NULL,      FOREIGN KEY (estudiante\_id) REFERENCES Student(id),      FOREIGN KEY (asignatura\_id) REFERENCES Asignaturas(id)  );  ------------------------------------------------------------------------------------------------  ----------------------------------- CREAR BASE DE DATOS --------------------------------------  E:  cd UCSM\4. IV Semestre\Lenguaje de Programación - 05\Practica 10  sqlite3 CalificacionesAcademicas.bd  CREATE TABLE Usuarios (      id INTEGER PRIMARY KEY AUTOINCREMENT,      dni TEXT NOT NULL UNIQUE,      contraseña TEXT NOT NULL,      rol TEXT NOT NULL CHECK (rol IN ('Docente', 'Alumno'))  );  CREATE TABLE Student (      id INTEGER PRIMARY KEY AUTOINCREMENT,      nombre TEXT NOT NULL,      dni TEXT NOT NULL UNIQUE  );  CREATE TABLE Asignaturas (      id INTEGER PRIMARY KEY AUTOINCREMENT,      nombre TEXT NOT NULL UNIQUE  );  CREATE TABLE Calificaciones (      id INTEGER PRIMARY KEY AUTOINCREMENT,      estudiante\_id INTEGER NOT NULL,      asignatura\_id INTEGER NOT NULL,      calificacion REAL NOT NULL CHECK (calificacion >= 0 AND calificacion <= 20),      FOREIGN KEY (estudiante\_id) REFERENCES Student(id),      FOREIGN KEY (asignatura\_id) REFERENCES Asignaturas(id)  );  --------------------------------------------------------------------------------------  ----------------------------------- MOSTRAR TABLAS ------------------------------------  E:  cd UCSM\4. IV Semestre\Lenguaje de Programación - 05\Practica 10  sqlite3 CalificacionesAcademicas.bd  .tables  ---------------------------------------------------------------------------------------  ------------------------------ MOSTRAR TABLAS DETALLADAS ------------------------------  E:  cd UCSM\4. IV Semestre\Lenguaje de Programación - 05\Practica 10  sqlite3 CalificacionesAcademicas.bd  .schema  ----------------------------------------------------------------------------------------  ---------------------------BORRAR TODOS LOS DATOS DE UNA TABLA --------------------------------  E:  cd UCSM\4. IV Semestre\Lenguaje de Programación - 05\Practica 10  sqlite3 CalificacionesAcademicas.bd  DELETE FROM Nom\_table1;  DELETE FROM Nom\_table2;  ...  ----------------------------------------------------------------------------------------  ------------------------------ REINICIAR EL ID (CONTADOR) ------------------------------  E:  cd UCSM\4. IV Semestre\Lenguaje de Programación - 05\Practica 10  sqlite3 CalificacionesAcademicas.bd  DELETE FROM sqlite\_sequence WHERE name='Nom\_table1';  DELETE FROM sqlite\_sequence WHERE name='Nom\_table2';  -------------------------------------------------------------------------------------------  \*/    import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.SQLException;  public class ConexionBD {      // URL de la BD      private static final String URL = "jdbc:sqlite:E:\\UCSM\\4. IV Semestre\\Lenguaje de Programación - 05\\Practica 10\\CalificacionesAcademicas.bd";      private static Connection connection;   // Variable para almacenar la conexión      // METODOS -----------      // conectar a la base de datos      public static Connection conectar() {          try {              // Establecer la conexión a la base de datos              connection = DriverManager.getConnection(URL);              System.out.println("Conectado a la base de datos.");          } catch (SQLException e) {    // Manejar errores de conexión              System.out.println("Error al conectar a la base de datos: " + e.getMessage());          }          return connection;      }      //  cerrar la conexión a la base de datos      public static void close() {          try {              if (connection != null && !connection.isClosed()) {                  connection.close(); //Cierra la conexión                  System.out.println("Conexión cerrada.");              }          } catch (SQLException e) {              // Manejar errores al cerrar la conexión              System.out.println("Error al cerrar la conexión: " + e.getMessage());          }      }  } |

* LoggerObserver.java

|  |
| --- |
| // Archivo: CRUD/LoggerObserver.java  package CRUD;  // Clase que implementa el patrón Observer  public class LoggerObserver implements Observer {      @Override      public void update(String mensaje) {          // Muestra el mensaje recibido en consola          System.out.println("[LOG]: " + mensaje);      }  } |

* Observer.java

|  |
| --- |
| package CRUD;  // Interfaz Observer para actualizar el mensaje  public interface Observer {      void update(String mensaje);  } |

* StudentDAO.java

|  |
| --- |
| package CRUD;  import java.sql.\*;  import javax.swing.\*;  import java.util.ArrayList;  import java.util.List;  public class StudentDAO {      // METODO      // Agrega un estudiante a la base de datos      public boolean agregarEstudiante(String nombre, String dni) {          String sql = "INSERT INTO Student(nombre, dni) VALUES(?, ?)";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, nombre);              pstmt.setString(2, dni);              pstmt.executeUpdate();              return true;          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al agregar estudiante: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);              return false;          }      }      // Verifica si un DNI esta registrado en la base de datos      public boolean validarDNI(String dni) {          String sql = "SELECT COUNT(\*) AS total FROM Student WHERE dni = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, dni);              ResultSet rs = pstmt.executeQuery();              if (rs.next()) {                  return rs.getInt("total") > 0;              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al validar DNI: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return false;      }      // Devuelve una lista con todos los estudiantes registrados      public List<String> listarEstudiantes() {          List<String> estudiantes = new ArrayList<>();          String sql = "SELECT \* FROM Student";          try (Connection conn = ConexionBD.conectar();               Statement stmt = conn.createStatement();               ResultSet rs = stmt.executeQuery(sql)) {              while (rs.next()) {                  estudiantes.add(rs.getInt("id") + " - " + rs.getString("nombre") + " (DNI: " + rs.getString("dni") + ")");              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al listar estudiantes: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return estudiantes;      }        // Obtiene el nombre de un estudiante dado su DNI      public String obtenerNombrePorDNI(String dni) {          String sql = "SELECT nombre FROM Student WHERE dni = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, dni);              ResultSet rs = pstmt.executeQuery();              if (rs.next()) {                  return rs.getString("nombre");              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al obtener el nombre del estudiante: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return "";      }      public boolean dniExiste(String dni) {          String sql = "SELECT COUNT(\*) FROM Student WHERE dni = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, dni);              ResultSet rs = pstmt.executeQuery();              if (rs.next()) {                  return rs.getInt(1) > 0; // Si el conteo es mayor a 0, el DNI existe              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al verificar DNI: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return false; // Retorna falso si ocurre algún error o no existe el DNI      }      } |

* UsuarioDAO.java

|  |
| --- |
| package CRUD;  import java.sql.\*;  import javax.swing.\*;  public class UsuarioDAO {      // METODO      // Registra un nuevo usuario en la base de datos      public boolean registrarUsuario(String dni, String contraseña, String rol) {          String sql = "INSERT INTO Usuarios(dni, contraseña, rol) VALUES(?, ?, ?)";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, dni);              pstmt.setString(2, contraseña);              pstmt.setString(3, rol);              pstmt.executeUpdate();              return true;          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al registrar usuario: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);              return false;          }      }      //////////// Valida que el DNI y contraseña coincidan con un usuario registrado      public String validarUsuario(String dni, String contraseña) {          String sql = "SELECT rol FROM Usuarios WHERE dni = ? AND contraseña = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, dni);              pstmt.setString(2, contraseña);              ResultSet rs = pstmt.executeQuery();              if (rs.next()) {                  return rs.getString("rol");              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al validar usuario: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return null;      }      /////////////////////      // Obtiene el DNI de un usuario      public String obtenerUsuario(String dni) {          String sql = "SELECT dni FROM Usuarios WHERE dni = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, dni);              ResultSet rs = pstmt.executeQuery();              if (rs.next()) {                  return rs.getString("dni");              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al obtener usuario: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return "";      }        ///// Obtiene la contraseña de un usuario por su DNI      public String obtenerContrasena(String dni) {          String sql = "SELECT contraseña FROM Usuarios WHERE dni = ?";          try (Connection conn = ConexionBD.conectar();               PreparedStatement pstmt = conn.prepareStatement(sql)) {              pstmt.setString(1, dni);              ResultSet rs = pstmt.executeQuery();              if (rs.next()) {                  return rs.getString("contraseña");              }          } catch (SQLException e) {              JOptionPane.showMessageDialog(null, "Error al obtener contraseña: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);          }          return "";      }      ////////////  } |

* Main.java

|  |
| --- |
| package CRUD;  import javax.swing.\*;  import java.awt.\*;  import java.io.File;  import CRUD.main\_function.ListarEstudiantes;  import CRUD.main\_function.MostrarHistorialDeCalificaciones;  import CRUD.main\_function.MostrarPromedio;  import CRUD.main\_function.RegistrarAsignatura;  import CRUD.main\_function.RegistrarCalificacion;  import CRUD.main\_function.RegistrarEstudiante;  public class Main {      public static void main(String[] args) {          // Patrón DAO: Data Access Object: lógica del programa          StudentDAO estudianteDAO = new StudentDAO();          AsignaturaDAO asignaturaDAO = new AsignaturaDAO();          CalificacionDAO calificacionDAO = new CalificacionDAO();          // Registrar un observador para notificaciones de cambios          calificacionDAO.addObserver(new LoggerObserver());          // Cargar la imagen desde la ruta especificada          String rutaImagen = "images/ucsm.png";  // Usando una ruta relativa          File imagenArchivo = new File(rutaImagen);          // Crear la imagen          Image img = Toolkit.getDefaultToolkit().getImage(imagenArchivo.getAbsolutePath());          // Redimensionar la imagen si es necesario          Image nuevaImg = img.getScaledInstance(50, 50, Image.SCALE\_SMOOTH); // Redimensionar si es necesario          // MENU DOCENTE O ALUMNO          String[] roles = {"Docente", "Alumno"};          String rolSeleccionado = (String) JOptionPane.showInputDialog(              null,                             // null = ventana independiente              "Seleccione el tipo de usuario:", // Mensaje que muestra              "Inicio",                         // Título              JOptionPane.PLAIN\_MESSAGE,        // Cambiar a PLAIN\_MESSAGE para no mostrar íconos predeterminados              new ImageIcon(nuevaImg),         // Usar la imagen redimensionada como ícono              roles,                            // Opciones arreglo de Strings              roles[0]                          // Opción seleccionada por defecto/ docente (1)          );          // selecciono un rol?          if (rolSeleccionado == null) {              JOptionPane.showMessageDialog(null, "Debe seleccionar un rol.");              System.exit(0); // Finalizar          }          String rol = rolSeleccionado; // Guardar el rol seleccionado          String dni = null;          String nombreAlumno = null;          // Si es Alumno, solicitar y validar DNI          if (rol.equals("Alumno")) {              dni = JOptionPane.showInputDialog("Ingrese su DNI:");              if (dni == null || dni.trim().isEmpty()) {                  JOptionPane.showMessageDialog(null, "Debe proporcionar su DNI.");                  System.exit(0);              }              // DNI existe?              boolean existeDni = estudianteDAO.validarDNI(dni);              if (!existeDni) {                  JOptionPane.showMessageDialog(null, "El DNI ingresado no está registrado.");                  System.exit(0);              }              // Sacar nombre del Alumno              nombreAlumno = estudianteDAO.obtenerNombrePorDNI(dni);          }          // MENU PRINCIPAL ----------------------------          while (true) {              // Crear un arreglo adicional para las opciones dependiendo del rol              String[] opciones = new String[0];  // Inicialmente vacío              switch (rol) {                  case "Docente":                      opciones = new String[]{                          "Registrar Estudiante",                          "Registrar Asignatura",                          "Registrar Calificación",                          "Listar Estudiantes",                          "Mostrar Historial de Calificaciones",                          "Salir"                      };                      break;                  case "Alumno":                      opciones = new String[]{                          "Mostrar Promedio",                          "Mostrar Historial de Calificaciones",                          "Salir"                      };                      break;              }              // Mostrar menú de opciones              String opcion = (String) JOptionPane.showInputDialog(                      null,                      "Seleccione una opción",                      "Menú Principal (" + rol + ")",                      JOptionPane.PLAIN\_MESSAGE,    // Cambiado a PLAIN\_MESSAGE para no mostrar íconos predeterminados                      new ImageIcon(nuevaImg),      // Usar la imagen redimensionada para el ícono                      opciones,                     // opciones                      opciones[0]                   // por defecto              );              // Verificar si selecciono salir              if (opcion == null || opcion.equals("Salir")) {                  JOptionPane.showMessageDialog(null, "¡Hasta luego!");                  System.exit(0);              }              // OPCIONES DEL MENU              switch (opcion) {                  case "Registrar Estudiante":                      if (rol.equals("Docente")) {                          RegistrarEstudiante.ejecutar(estudianteDAO);                      }                      break;                  case "Registrar Asignatura":                      if (rol.equals("Docente")) {                          RegistrarAsignatura.ejecutar(asignaturaDAO);                      }                      break;                  case "Registrar Calificación":                      if (rol.equals("Docente")) {                          RegistrarCalificacion.ejecutar(estudianteDAO, asignaturaDAO, calificacionDAO);                      }                      break;                  case "Mostrar Historial de Calificaciones":                      if (rol.equals("Docente")) {                          MostrarHistorialDeCalificaciones.ejecutarParaDocente(calificacionDAO, asignaturaDAO);                      } else if (rol.equals("Alumno")) {                          MostrarHistorialDeCalificaciones.ejecutarParaAlumno(calificacionDAO, dni);                      }                      break;                  case "Listar Estudiantes":                      ListarEstudiantes.ejecutar(estudianteDAO);                      break;                  case "Mostrar Promedio":                      if (rol.equals("Alumno")) {                          MostrarPromedio.ejecutar(calificacionDAO, dni, nombreAlumno);                      }                      break;              }          }      }  } |

* main ListarEstudiantes.java

|  |
| --- |
| package CRUD.main\_function;  import CRUD.StudentDAO;  import javax.swing.\*;  import java.util.List;  public class ListarEstudiantes {      public static void ejecutar(StudentDAO estudianteDAO) {          List<String> estudiantes = estudianteDAO.listarEstudiantes();          JOptionPane.showMessageDialog(null, estudiantes.isEmpty() ? "No hay estudiantes registrados." : String.join("\n", estudiantes));      }  } |

* main MostrarHistorialDeCalificaciones.java

|  |
| --- |
| package CRUD.main\_function;  import CRUD.CalificacionDAO;  import CRUD.AsignaturaDAO;  import javax.swing.\*;  import javax.swing.table.DefaultTableModel;  import java.awt.\*;  import java.io.File;  import java.util.List;  public class MostrarHistorialDeCalificaciones {      public static void ejecutarParaDocente(CalificacionDAO calificacionDAO, AsignaturaDAO asignaturaDAO) {          // Cargar la imagen desde la ruta especificada          String rutaImagen = "images/ucsm.png";  // Usando una ruta relativa          File imagenArchivo = new File(rutaImagen);          // Crear la imagen          Image img = Toolkit.getDefaultToolkit().getImage(imagenArchivo.getAbsolutePath());          // Redimensionar la imagen si es necesario          Image nuevaImg = img.getScaledInstance(50, 50, Image.SCALE\_SMOOTH); // Redimensionar si es necesario          List<String> asignaturas = asignaturaDAO.listarAsignaturas();          if (asignaturas.isEmpty()) {              JOptionPane.showMessageDialog(null, "No hay asignaturas registradas.");              return;          }          String asignaturaSeleccionada = (String) JOptionPane.showInputDialog(                  null,                  "Seleccione una asignatura:",                  "Asignaturas",                  JOptionPane.QUESTION\_MESSAGE,                  new ImageIcon(nuevaImg),  // Usar la imagen redimensionada como ícono                  asignaturas.toArray(),                  asignaturas.get(0)          );          if (asignaturaSeleccionada == null) return;          int asignaturaId = Integer.parseInt(asignaturaSeleccionada.split(" - ")[0]);          List<String[]> historial = calificacionDAO.historialPorAsignatura(asignaturaId);          if (historial.isEmpty()) {              JOptionPane.showMessageDialog(null, "No hay calificaciones registradas para esta asignatura.");          } else {              String[] columnas = {"Estudiante", "F1", "F2", "F3", "Promedio", "Estado"};              DefaultTableModel modeloTabla = new DefaultTableModel(columnas, 0) {                  // Sobrescribir este método para que las celdas no sean editables                  @Override                  public boolean isCellEditable(int row, int column) {                      return false; // No permitir edición                  }              };              int cantidadAprobados = 0;              int cantidadDesaprobados = 0;              for (String[] fila : historial) {                  // Redondear el promedio a 2 decimales                  double promedio = Double.parseDouble(fila[4]);                  fila[4] = String.format("%.2f", promedio); // Redondear a 2 decimales                  // Determinar si aprobó o desaprobó                  String estado = promedio >= 11.5 ? "APROBÓ" : "DESAPROBÓ";                  // Actualizar contadores de aprobados y desaprobados                  if ("APROBÓ".equals(estado)) {                      cantidadAprobados++;                  } else {                      cantidadDesaprobados++;                  }                  // Crear nueva fila con el estado agregado                  String[] filaConEstado = new String[fila.length + 1];                  System.arraycopy(fila, 0, filaConEstado, 0, fila.length);                  filaConEstado[fila.length] = estado;                  modeloTabla.addRow(filaConEstado);              }              JTable tabla = new JTable(modeloTabla);              JScrollPane scrollPane = new JScrollPane(tabla);              // Crear el panel para mostrar los resultados adicionales              JPanel panel = new JPanel();              panel.setLayout(new BorderLayout());              panel.add(scrollPane, BorderLayout.CENTER);              // Crear un texto con la cantidad de alumnos, aprobados y desaprobados              String textoResumen = String.format(                      "Cantidad de alumnos: %d\nCantidad de aprobados: %d\nCantidad de desaprobados: %d",                      historial.size(), cantidadAprobados, cantidadDesaprobados              );              JTextArea textArea = new JTextArea(textoResumen);              textArea.setEditable(false);              panel.add(textArea, BorderLayout.SOUTH);              JOptionPane.showMessageDialog(null, panel, "Historial de Calificaciones", JOptionPane.INFORMATION\_MESSAGE);          }      }      public static void ejecutarParaAlumno(CalificacionDAO calificacionDAO, String dni) {          // Cargar la imagen desde la ruta especificada          String rutaImagen = "images/ucsm.png";  // Usando una ruta relativa          File imagenArchivo = new File(rutaImagen);          // Crear la imagen          Image img = Toolkit.getDefaultToolkit().getImage(imagenArchivo.getAbsolutePath());          // Redimensionar la imagen si es necesario          Image nuevaImg = img.getScaledInstance(50, 50, Image.SCALE\_SMOOTH); // Redimensionar si es necesario          List<String[]> historial = calificacionDAO.historialCalificaciones(dni);          if (historial.isEmpty()) {              JOptionPane.showMessageDialog(null, "No hay calificaciones registradas para este estudiante.");          } else {              String[] columnas = {"Asignatura", "F1", "F2", "F3", "Promedio", "Estado"};              DefaultTableModel modeloTabla = new DefaultTableModel(columnas, 0) {                  // Sobrescribir este método para que las celdas no sean editables                  @Override                  public boolean isCellEditable(int row, int column) {                      return false; // No permitir edición                  }              };              int cantidadAprobados = 0;              int cantidadDesaprobados = 0;              for (String[] fila : historial) {                  // Redondear el promedio a 2 decimales                  double promedio = Double.parseDouble(fila[4]);                  fila[4] = String.format("%.2f", promedio); // Redondear a 2 decimales                  // Determinar si aprobó o desaprobó                  String estado = promedio >= 11.5 ? "APROBÓ" : "DESAPROBÓ";                  // Actualizar contadores de aprobados y desaprobados                  if ("APROBÓ".equals(estado)) {                      cantidadAprobados++;                  } else {                      cantidadDesaprobados++;                  }                  // Crear nueva fila con el estado agregado                  String[] filaConEstado = new String[fila.length + 1];                  System.arraycopy(fila, 0, filaConEstado, 0, fila.length);                  filaConEstado[fila.length] = estado;                  modeloTabla.addRow(filaConEstado);              }              JTable tabla = new JTable(modeloTabla);              JScrollPane scrollPane = new JScrollPane(tabla);              // Crear el panel para mostrar los resultados adicionales              JPanel panel = new JPanel();              panel.setLayout(new BorderLayout());              panel.add(scrollPane, BorderLayout.CENTER);              // Crear un texto con la cantidad de aprobados, desaprobados y total de asignaturas              String textoResumen = String.format(                      "Cantidad de asignaturas: %d\nCantidad de aprobados: %d\nCantidad de desaprobados: %d",                      historial.size(), cantidadAprobados, cantidadDesaprobados              );              JTextArea textArea = new JTextArea(textoResumen);              textArea.setEditable(false);              panel.add(textArea, BorderLayout.SOUTH);              JOptionPane.showMessageDialog(null, panel, "Historial de Calificaciones", JOptionPane.INFORMATION\_MESSAGE);          }      }  } |

* main MostrarPromedio.java

|  |
| --- |
| package CRUD.main\_function;  import CRUD.CalificacionDAO;  import javax.swing.\*;  public class MostrarPromedio {      public static void ejecutar(CalificacionDAO calificacionDAO, String dni, String nombreAlumno) {          double promedio = calificacionDAO.calcularPromedio(dni);          JOptionPane.showMessageDialog(null,                  "Alumno: " + nombreAlumno + "\n\nSu promedio general es: " + promedio,                  "Promedio General",                  JOptionPane.INFORMATION\_MESSAGE);      }  } |

* main RegistrarAsignatura.java

|  |
| --- |
| package CRUD.main\_function;  import CRUD.AsignaturaDAO;  import javax.swing.\*;  public class RegistrarAsignatura {      public static void ejecutar(AsignaturaDAO asignaturaDAO) {          String nombre;          while (true) {              nombre = JOptionPane.showInputDialog("Ingrese el nombre de la asignatura:");              if (nombre == null) {                  // Si se presiona "Cancelar", regresar al menú principal                  return;              }              if (nombre.trim().isEmpty()) {                  JOptionPane.showMessageDialog(null, "Debe ingresar un nombre para la asignatura.");                  continue; // Volver a pedir el nombre si está vacío              }              // Validar el nombre (solo letras y espacios)              boolean nombreValido = true;              for (char c : nombre.toCharArray()) {                  if (!((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') || c == ' ' ||                  c == 'á' || c == 'é' || c == 'í' || c == 'ó' || c == 'ú' ||                  c == 'Á' || c == 'É' || c == 'Í' || c == 'Ó' || c == 'Ú' ||                  c == 'ñ' || c == 'Ñ')) {                      nombreValido = false;                      break;                  }              }              if (!nombreValido) {                  JOptionPane.showMessageDialog(null, "El nombre solo puede contener letras y espacios. Inténtelo de nuevo.");              } else {                  break; // Salir del ciclo si el nombre es válido              }          }          // Verificar si la asignatura ya existe          if (asignaturaDAO.asignaturaExiste(nombre.trim())) {              JOptionPane.showMessageDialog(null, "Esta asignatura ya ha sido registrada anteriormente.");          } else {              // Registrar la asignatura              boolean exito = asignaturaDAO.agregarAsignatura(nombre.trim());              JOptionPane.showMessageDialog(null, exito ? "Asignatura registrada con éxito." : "Error al registrar asignatura.");          }      }  } |

* main RegistrarCalificacion.java

|  |
| --- |
| package CRUD.main\_function;  import CRUD.CalificacionDAO;  import CRUD.AsignaturaDAO;  import CRUD.StudentDAO;  import javax.swing.\*;  import java.awt.\*;  import java.io.File;  import java.util.List;  public class RegistrarCalificacion {      public static void ejecutar(StudentDAO estudianteDAO, AsignaturaDAO asignaturaDAO, CalificacionDAO calificacionDAO) {          // Cargar la imagen desde la ruta especificada          String rutaImagen = "images/ucsm.png";  // Usando una ruta relativa          File imagenArchivo = new File(rutaImagen);          // Crear la imagen          Image img = Toolkit.getDefaultToolkit().getImage(imagenArchivo.getAbsolutePath());          // Redimensionar la imagen si es necesario          Image nuevaImg = img.getScaledInstance(50, 50, Image.SCALE\_SMOOTH); // Redimensionar si es necesario          // Listar estudiantes y asignaturas registrados          List<String> estudiantes = estudianteDAO.listarEstudiantes();          List<String> asignaturas = asignaturaDAO.listarAsignaturas();          if (estudiantes.isEmpty() || asignaturas.isEmpty()) {              JOptionPane.showMessageDialog(null, "Debe registrar al menos un estudiante y una asignatura primero.");              return;          }          // Seleccionar un estudiante          String estudianteSeleccionado = (String) JOptionPane.showInputDialog(                  null,                  "Seleccione un estudiante:",                  "Estudiantes",                  JOptionPane.QUESTION\_MESSAGE,                  new ImageIcon(nuevaImg),  // Usar la imagen redimensionada como ícono                  estudiantes.toArray(),                  estudiantes.get(0)          );          if (estudianteSeleccionado == null) return;          int estudianteId = Integer.parseInt(estudianteSeleccionado.split(" - ")[0]);          String nombreEstudiante = estudianteSeleccionado.split(" - ")[1];          // Seleccionar una asignatura          String asignaturaSeleccionada = (String) JOptionPane.showInputDialog(                  null,                  "Seleccione una asignatura:",                  "Asignaturas",                  JOptionPane.QUESTION\_MESSAGE,                  new ImageIcon(nuevaImg),  // Usar la imagen redimensionada como ícono                  asignaturas.toArray(),                  asignaturas.get(0)          );          if (asignaturaSeleccionada == null) return;          int asignaturaId = Integer.parseInt(asignaturaSeleccionada.split(" - ")[0]);          String nombreAsignatura = asignaturaSeleccionada.split(" - ")[1];          // Verificar si el alumno ya tiene calificación en esa asignatura          boolean tieneCalificacion = calificacionDAO.tieneCalificacion(estudianteId, asignaturaId);          if (tieneCalificacion) {              int opcion = JOptionPane.showConfirmDialog(                      null,                      "El alumno " + nombreEstudiante + " ya tiene nota en el curso de " + nombreAsignatura + ". ¿Desea reemplazar esa nota?",                      "Confirmar reemplazo",                      JOptionPane.YES\_NO\_OPTION              );              if (opcion == JOptionPane.NO\_OPTION) {                  JOptionPane.showMessageDialog(null, "Operación cancelada. Regresando al menú principal.");                  return;              }          }          // Solicitar las calificaciones en un solo cuadro de entrada          JPanel panel = new JPanel();          JTextField f1Field = new JTextField(5);          JTextField f2Field = new JTextField(5);          JTextField f3Field = new JTextField(5);          panel.add(new JLabel("F1:"));          panel.add(f1Field);          panel.add(new JLabel("F2:"));          panel.add(f2Field);          panel.add(new JLabel("F3:"));          panel.add(f3Field);          int option = JOptionPane.showConfirmDialog(null, panel, "Ingrese las calificaciones", JOptionPane.OK\_CANCEL\_OPTION);          if (option == JOptionPane.OK\_OPTION) {              try {                  double f1 = Double.parseDouble(f1Field.getText());                  double f2 = Double.parseDouble(f2Field.getText());                  double f3 = Double.parseDouble(f3Field.getText());                  // Calcular promedio                  double promedio = (f1 + f2 + f3) / 3.0;                  // Registrar o actualizar calificación                  boolean exito;                  if (tieneCalificacion) {                      exito = calificacionDAO.actualizarCalificacion(estudianteId, asignaturaId, f1, f2, f3, promedio);                  } else {                      exito = calificacionDAO.agregarCalificacion(estudianteId, asignaturaId, f1, f2, f3, promedio);                  }                  if (exito) {                      JOptionPane.showMessageDialog(null, "Calificación registrada con éxito.");                  } else {                      JOptionPane.showMessageDialog(null, "Hubo un problema al registrar la calificación.");                  }              } catch (NumberFormatException e) {                  JOptionPane.showMessageDialog(null, "Por favor, ingrese valores numéricos válidos para las calificaciones.");              }          }      }  } |

* main RegistrarEstudiante.java

|  |
| --- |
| package CRUD.main\_function;  import CRUD.StudentDAO;  import javax.swing.\*;  import java.awt.\*;  import java.io.File;  public class RegistrarEstudiante {      public static void ejecutar(StudentDAO estudianteDAO) {          // Ruta de la imagen para usar como ícono          String rutaImagen = "images/ucsm.png"; // Ajusta la ruta según la ubicación real de tu imagen          File imagenArchivo = new File(rutaImagen);          Image img = Toolkit.getDefaultToolkit().getImage(imagenArchivo.getAbsolutePath());          Image nuevaImg = img.getScaledInstance(50, 50, Image.SCALE\_SMOOTH); // Redimensionar si es necesario          ImageIcon icono = new ImageIcon(nuevaImg);          String nombre;          String dni;          // Solicitar nombre y validar en el momento usando ASCII          while (true) {              nombre = (String) JOptionPane.showInputDialog(                  null,                  "Ingrese el nombre del estudiante:",                  "Registrar Estudiante",                  JOptionPane.PLAIN\_MESSAGE,                  icono, // Icono personalizado                  null,                  null              );              if (nombre == null) {                  // Si se presiona "Cancelar", regresar al menú principal                  return;              }              if (nombre.trim().isEmpty()) {                  JOptionPane.showMessageDialog(                      null,                      "Debe ingresar un nombre.",                      "Error",                      JOptionPane.ERROR\_MESSAGE,                      icono                  );                  continue; // Volver a pedir el nombre si está vacío              }              // Validar el nombre (solo letras, espacios y caracteres válidos)              boolean nombreValido = true;              for (char c : nombre.toCharArray()) {                  if (!((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') || c == ' ' ||                        c == 'á' || c == 'é' || c == 'í' || c == 'ó' || c == 'ú' ||                        c == 'Á' || c == 'É' || c == 'Í' || c == 'Ó' || c == 'Ú' ||                        c == 'ñ' || c == 'Ñ')) {                      nombreValido = false;                      break;                  }              }              if (!nombreValido) {                  JOptionPane.showMessageDialog(                      null,                      "Por favor ingrese un nombre correcto.",                      "Error",                      JOptionPane.ERROR\_MESSAGE,                      icono                  );              } else {                  break; // Salir del ciclo si el nombre es válido              }          }          // Solicitar DNI y validar en el momento          while (true) {              dni = (String) JOptionPane.showInputDialog(                  null,                  "Ingrese el DNI del estudiante:",                  "Registrar Estudiante",                  JOptionPane.PLAIN\_MESSAGE,                  icono, // Icono personalizado                  null,                  null              );              if (dni == null) {                  // Si se presiona "Cancelar", regresar al menú principal                  return;              }              if (dni.trim().isEmpty()) {                  JOptionPane.showMessageDialog(                      null,                      "Debe ingresar un DNI.",                      "Error",                      JOptionPane.ERROR\_MESSAGE,                      icono                  );                  continue; // Volver a pedir el DNI si está vacío              }              // Validar el DNI (solo números y 8 dígitos)              if (!dni.matches("\\d{8}")) {                  JOptionPane.showMessageDialog(                      null,                      "El DNI debe ser solo números y tener exactamente 8 dígitos. Inténtelo de nuevo.",                      "Error",                      JOptionPane.ERROR\_MESSAGE,                      icono                  );              } else {                  break; // Salir del ciclo si el DNI es válido              }          }          // Verificar si el DNI ya existe          if (estudianteDAO.dniExiste(dni.trim())) {              JOptionPane.showMessageDialog(                  null,                  "Este DNI ya ha sido ingresado anteriormente.",                  "Error",                  JOptionPane.ERROR\_MESSAGE,                  icono              );          } else {              // Registrar el estudiante              boolean exito = estudianteDAO.agregarEstudiante(nombre.trim(), dni.trim());              JOptionPane.showMessageDialog(                  null,                  exito ? "Estudiante registrado con éxito." : "Error al registrar estudiante.",                  "Resultado",                  JOptionPane.INFORMATION\_MESSAGE,                  icono              );          }      }  } |

**PRUEBA DE PROGRAMA**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente con confianza media

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente con confianza media

Tabla

Descripción generada automáticamente

**Conclusiones**

* La guía proporciona una comprensión profunda de las herramientas avanzadas de Java Swing, como el binding de datos, la creación de gráficos y la integración de multimedia, esenciales para el desarrollo de aplicaciones interactivas y dinámicas.
* A través de las actividades y ejercicios propuestos, los estudiantes pueden aplicar lo aprendido en ejemplos prácticos, lo que fortalece su habilidad para diseñar interfaces gráficas y gestionar datos de manera eficiente.
* El uso de patrones de diseño como MVC y la integración de bases de datos en proyectos permite a los estudiantes desarrollar aplicaciones más robustas y escalables, preparándolos para enfrentar desafíos reales en el desarrollo de software.

**Cuestionario**

1. ¿Cómo se puede sincronizar un modelo de datos y una interfaz gráfica en Swing de forma manual?

Se puede sincronizar utilizando listeners de eventos y métodos de getters y setters. Por ejemplo, un ActionListener se puede usar para actualizar el modelo con los valores modificados en los componentes gráficos, como JTextField. [1]

1. ¿Qué métodos de los componentes de Swing se utilizan comúnmente para obtener y establecer valores al realizar binding de datos manual?

Los métodos comunes son getText() y setText() para JTextField, getSelectedItem() y setSelectedItem() para JComboBox, entre otros. Estos permiten obtener y establecer valores en los componentes gráficos. [2]

1. ¿Cómo se maneja la validación de datos en un formulario que utiliza binding de datos manual?

La validación se puede hacer utilizando validadores antes de actualizar el modelo, verificando que los valores ingresados sean correctos (por ejemplo, utilizando try-catch para convertir cadenas a números y asegurarse de que la entrada es válida). [3]

1. ¿Cómo podrías implementar un binding de datos manual para una lista de objetos y mostrar los datos en un componente JTable?

Se puede utilizar un TableModel personalizado que maneje la lista de objetos. El TableModel debe implementar los métodos getValueAt() y setValueAt() para mostrar y modificar los datos en la tabla.

1. ¿Cómo se pueden crear gráficos interactivos que respondan a eventos de ratón y teclado en un JPanel en Swing?

Se pueden utilizar los listeners de eventos, como MouseListener para eventos de ratón y KeyListener para eventos de teclado. Estos eventos pueden modificar el gráfico dibujado en el componente JPanel al capturar las interacciones del usuario.

1. ¿Cómo podrías implementar un gráfico de datos (barras o líneas) utilizando Graphics en Swing para visualizar valores numéricos y permitir que se actualicen dinámicamente?

Se dibujan las formas, que pueden ser barras o líneas, utilizando el objeto Graphics en el método paintComponent(). Para actualizaciones dinámicas, se puede llamar a repaint() en el componente para redibujar el gráfico cada vez que cambien los datos.

1. ¿Cómo funciona el método paintComponent(Graphics g) y cuándo se llama automáticamente?

El método paintComponent(Graphics g) se usa para redibujar el contenido del componente. Se llama automáticamente cuando el componente necesita ser redibujado, como cuando la ventana se redimensiona o cuando se invoca repaint().

1. ¿Qué es antialiasing y cómo se aplica en gráficos creados con Graphics2D en Java Swing?

El antialiasing es una técnica que suaviza los bordes de las formas para hacerlas más visualmente agradables. Se aplica en Graphics2D utilizando el método setRenderingHint() con la clave RenderingHints.KEY\_ANTIALIASING y el valor RenderingHints.VALUE\_ANTIALIAS\_ON. [4]

1. ¿Qué clases de Java estándar permiten trabajar con audio en una aplicación Swing y cuáles son sus principales métodos?

La clase Clip de javax.sound.sampled se utiliza para cargar y reproducir archivos de audio. Los métodos principales son open(), start(), stop() y close() para controlar la reproducción del audio.

1. ¿Cuál es la diferencia entre los formatos de audio soportados nativamente en Java (WAV, AU) y otros formatos populares como MP3? ¿Cómo se pueden reproducir estos formatos en Java Swing?

Java soporta nativamente los formatos WAV y AU, pero no MP3. Para reproducir MP3, se necesita una biblioteca externa como JavaZoom JLayer, que proporciona soporte para la decodificación y reproducción de archivos MP3.

1. ¿Cómo se usa la clase Clip en javax.sound.sampled para cargar y reproducir archivos de audio en Java Swing?

Se utiliza creando un objeto Clip, luego se carga el archivo de audio con AudioSystem.getAudioInputStream(), y se reproduce con el método clip.start().

1. ¿Cómo se pueden pausar, detener y reanudar archivos de audio en una aplicación de Java Swing utilizando la clase Clip?

La clase Clip no tiene soporte directo para pausar y reanudar, pero se puede lograr deteniendo la reproducción con clip.stop() y luego reanudando desde la misma posición utilizando clip.setFramePosition() para ajustar la posición del audio antes de llamar a clip.start() nuevamente.

**REFERENCIAS BIBLIOGRAFICAS**

|  |  |
| --- | --- |
| [1] | A. J. Harkins, «Oracle,» 2020. [En línea]. Available: https://docs.oracle.com/javase/tutorial/uiswing/painting/closer.html. |
| [2] | Universo Java, «Universo Java,» noviembre 2022. [En línea]. Available: https://www.universojava.com/2022/11/eventos-de-java-swing.html. |
| [3] | J. L. Martín, «JDAL.org,» 2022. [En línea]. Available: https://www.jdal.org/doc/data\_binding.php. |
| [4] | Oracle, «The Java Tutorials,» 2022. [En línea]. Available: https://docs.oracle.com/javase/tutorial/2d/advanced/quality.html. |