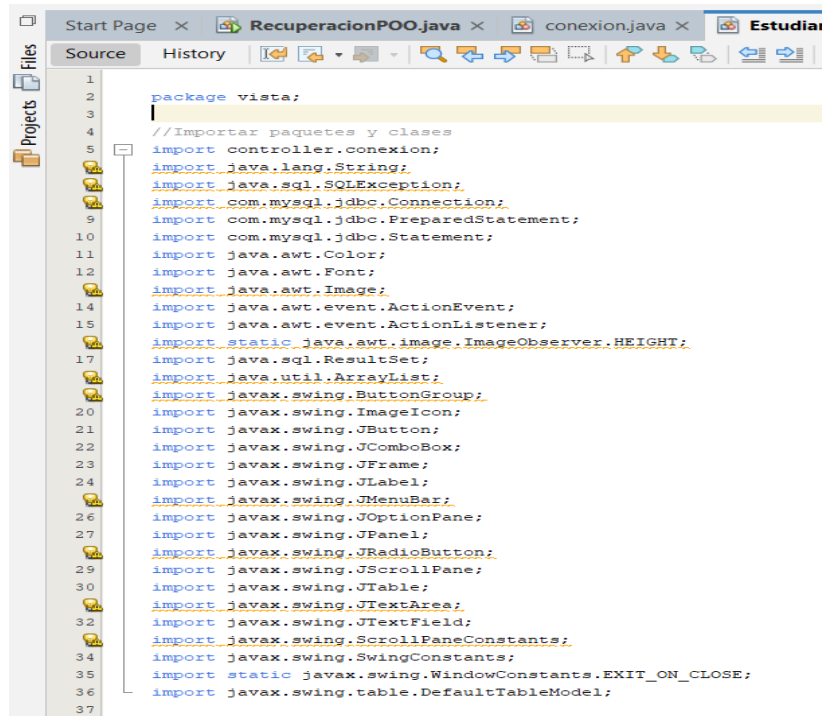


Guía del uso de tablas en interfaces por medio de programación

Paso 1: Importar paquetes y clases

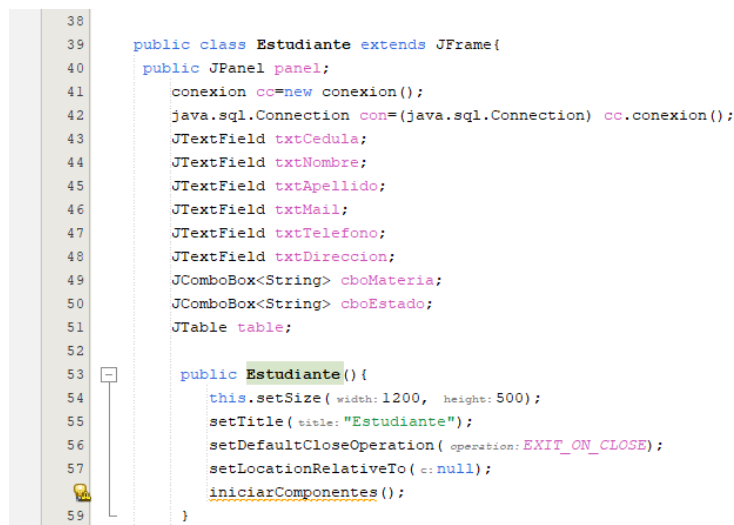
El código comienza importando los paquetes y clases necesarios, incluyendo las clases relacionadas con la GUI y la conexión a la base de datos MySQL.



```
1 package vista;
2
3 //Importar paquetes y clases
4 import controller.conexion;
5 import java.lang.String;
6 import java.sql.SQLException;
7 import com.mysql.jdbc.Connection;
8 import com.mysql.jdbc.PreparedStatement;
9 import com.mysql.jdbc.Statement;
10 import java.awt.Color;
11 import java.awt.Font;
12 import java.awt.Image;
13 import java.awt.event.ActionEvent;
14 import java.awt.event.ActionListener;
15 import static java.awt.image.ImageObserver.HEIGHT;
16 import java.sql.ResultSet;
17 import java.util.ArrayList;
18 import javax.swing.ButtonGroup;
19 import javax.swing.ImageIcon;
20 import javax.swing.JButton;
21 import javax.swing.JComboBox;
22 import javax.swing.JFrame;
23 import javax.swing.JLabel;
24 import javax.swing.JMenuBar;
25 import javax.swing.JOptionPane;
26 import javax.swing.JPanel;
27 import javax.swing.JRadioButton;
28 import javax.swing.JScrollPane;
29 import javax.swing.JTable;
30 import javax.swing.JTextArea;
31 import javax.swing.JTextField;
32 import javax.swing.ScrollPaneConstants;
33 import javax.swing.SwingConstants;
34 import static javax.swing.WindowConstants.EXIT_ON_CLOSE;
35 import javax.swing.table.DefaultTableModel;
36
37
```

Paso 2: Definir la clase Estudiante

Aquí se define la clase Estudiante, que hereda de JFrame, lo que significa que la interfaz gráfica se basa en una ventana. En el constructor de la clase, se establece el tamaño de la ventana, el título y la operación de cierre.



```
38
39 public class Estudiante extends JFrame{
40     public JPanel panel;
41     conexion cc=new conexion();
42     java.sql.Connection con=(java.sql.Connection) cc.conexion();
43     JTextField txtCedula;
44     JTextField txtNombre;
45     JTextField txtApellido;
46     JTextField txtMail;
47     JTextField txtTelefono;
48     JTextField txtDireccion;
49     JComboBox<String> cboMateria;
50     JComboBox<String> cboEstado;
51     JTable table;
52
53     public Estudiante(){
54         this.setSize( width: 1200, height: 500);
55         setTitle( title: "Estudiante");
56         setDefaultCloseOperation( operation: EXIT_ON_CLOSE);
57         setLocationRelativeTo( c: null);
58         iniciarComponentes();
59     }
60
61 }
```

Paso 3: Iniciar componentes

En el método `iniciarComponentes()`, se llaman a varios métodos para configurar los diferentes componentes en la GUI. Esto incluye colocar etiquetas, cajas de texto, comboboxes, una tabla y botones.

```
61 public void iniciarComponentes() {  
62     //llamamos a los todos los metodos creados |  
63     colocarPanel();  
64     colocarEtiqueta();  
65     ColocarCajatexto();  
66     colpcarComobox();  
67     tabla();  
68     colocarBotones();  
69 }  
70
```

Paso 4: Colocar paneles y etiquetas

Los métodos `colocarPanel()` y `colocarEtiqueta()` se encargan de configurar el panel de fondo y colocar etiquetas en la interfaz. Las etiquetas son los títulos o nombres de campo, como "Cédula", "Nombre", etc.

```
public void colocarPanel() {  
    //creamos un panel  
    panel = new JPanel();  
    //le asignamos un color  
    panel.setBackground( bg:Color.PINK);  
    this.getContentPane().add( comp:panel);  
}  
  
public void colocarEtiqueta(){  
    //instanciamos un objeto JLabel y le asignamos el nombre  
    JLabel label = new JLabel( text: "Estudiante", horizontalAlignment: SwingConstants.CENTER);  
    panel.add( comp:label);  
    panel.setLayout( mgr:null);  
    label.setBounds( x:400, y:10, width:200, height:50);  
    label.setForeground( fg:Color.BLACK);  
    label.setBackground( bg:Color.PINK);  
    label.setOpaque( isOpaque:true);  
    // creamos otro objeto  
    JLabel label2 = new JLabel( text:"Cedula:");  
    //agregamos la etiqueta al panel  
    panel.add( comp:label2);  
    label2.setHorizontalAlignment( alignment:SwingConstants.LEFT);  
    //ubicacion y cambiar la fuente  
    label2.setFont(new Font( name:"Arial", style:Font.PLAIN, size:15));  
    label2.setBounds( x:10, y:90, width:100, height:25);  
  
    JLabel label3 = new JLabel( text:"Nombre:");  
    panel.add( comp:label3);  
    label3.setHorizontalAlignment( alignment:SwingConstants.LEFT);  
    label3.setFont(new Font( name:"Arial", style:Font.PLAIN, size:15));  
    label3.setBounds( x:10, y:130, width:100, height:25);  
  
    JLabel label4 = new JLabel( text:"Apellido:");  
    panel.add( comp:label4);  
    label4.setHorizontalAlignment( alignment:SwingConstants.LEFT);  
    label4.setFont(new Font( name:"Arial", style:Font.PLAIN, size:15));  
    label4.setBounds( x:10, y:170, width:100, height:25);  
}
```

Paso 5: Colocar cajas de texto

El método `ColocarCajatexto()` coloca cajas de texto en la interfaz donde los usuarios pueden ingresar datos. Cada caja de texto se crea y se agrega al panel.

```

144 public void ColocarCajatexto() {
145     //coloca cajas de texto en la interfaz donde los usuarios pueden ingresar datos.
146     //Cada caja de texto se crea y se agrega al panel
147     txtCedula= new JTextField();
148     txtCedula.setBounds( x:100, y:90, width: 250, height:25);
149     panel.add( comp:txtCedula);
150     txtCedula.setText( s: "");
151     System.out.println("El texto de la caja es:"+txtCedula.getText());
152
153     txtNombre= new JTextField();
154     txtNombre.setBounds( x:100, y:133, width: 250, height:25);
155     panel.add( comp:txtNombre);
156     txtNombre.setText( s: "");
157     System.out.println("El texto de la caja es:"+txtNombre.getText());
158
159     txtApellido= new JTextField();
160     txtApellido.setBounds( x:100, y:170, width: 250, height:25);
161     panel.add( comp:txtApellido);
162     txtApellido.setText( s: "");
163     System.out.println("El texto de la caja es:"+txtApellido.getText());
164
165     txtMail= new JTextField();
166     txtMail.setBounds( x:100, y:210, width: 250, height: 25);
167     panel.add( comp:txtMail);
168     txtMail.setText( s: "");
169     System.out.println("El texto de la caja es:"+txtMail.getText());
170
171     txtTelefono= new JTextField();
172     txtTelefono.setBounds( x:100, y:250, width: 250, height:25);
173     panel.add( comp:txtTelefono);
174     txtTelefono.setText( s: "");
175     System.out.println("El texto de la caja es:"+txtTelefono.getText());
176
177     txtDireccion= new JTextField();
178     txtDireccion.setBounds( x:100, y:290, width: 250, height: 25);
179     panel.add( comp:txtDireccion);
180     txtDireccion.setText( s: "");
181     System.out.println("El texto de la caja es:"+txtDireccion.getText());
182 }

```

Paso 6: Colocar comboboxes

El método colocarCombobox() coloca dos comboboxes en la interfaz, uno para seleccionar la materia y otro para seleccionar el estado del estudiante.

```

183
184 public void colocarCombobox() {
185     //creamos un combo box
186     cboMateria = new JComboBox<>();
187     //ubicacion
188     cboMateria.setBounds( x:100, y:330, width: 150, height:25);
189     panel.add( comp:cboMateria);
190     //agregamos los items
191     cboMateria.addItem( item:"Computo");
192     cboMateria.addItem( item:"Edo");
193     cboMateria.addItem( item:"Calculo");
194     cboMateria.addItem( item:"Poo");
195     cboMateria.addItem( item:"Fisica");
196     //usamos el ActionListener para asignarle un evento
197     cboMateria.addActionListener( (ActionEvent e) -> {
198         JComboBox<String> comboBox = (JComboBox<String>) e.getSource();
199         String selectedOption = (String) comboBox.getSelectedItem();
200         System.out.println("Opción seleccionada: " + selectedOption);
201     });
202     panel.add( comp:cboMateria);
203
204     cboEstado = new JComboBox<>();
205     cboEstado.setBounds( x:100, y:360, width: 150, height: 25);
206     panel.add( comp:cboEstado);
207     cboEstado.addItem( item:"Activo");
208     cboEstado.addItem( item:"Inactivo");
209     cboEstado.addActionListener( (ActionEvent e) -> {
210         JComboBox<String> comboBox = (JComboBox<String>) e.getSource();
211         String selectedOption = (String) comboBox.getSelectedItem();
212         System.out.println("Opción seleccionada: " + selectedOption);
213     });
214     panel.add( comp:cboEstado);
215 }
216

```

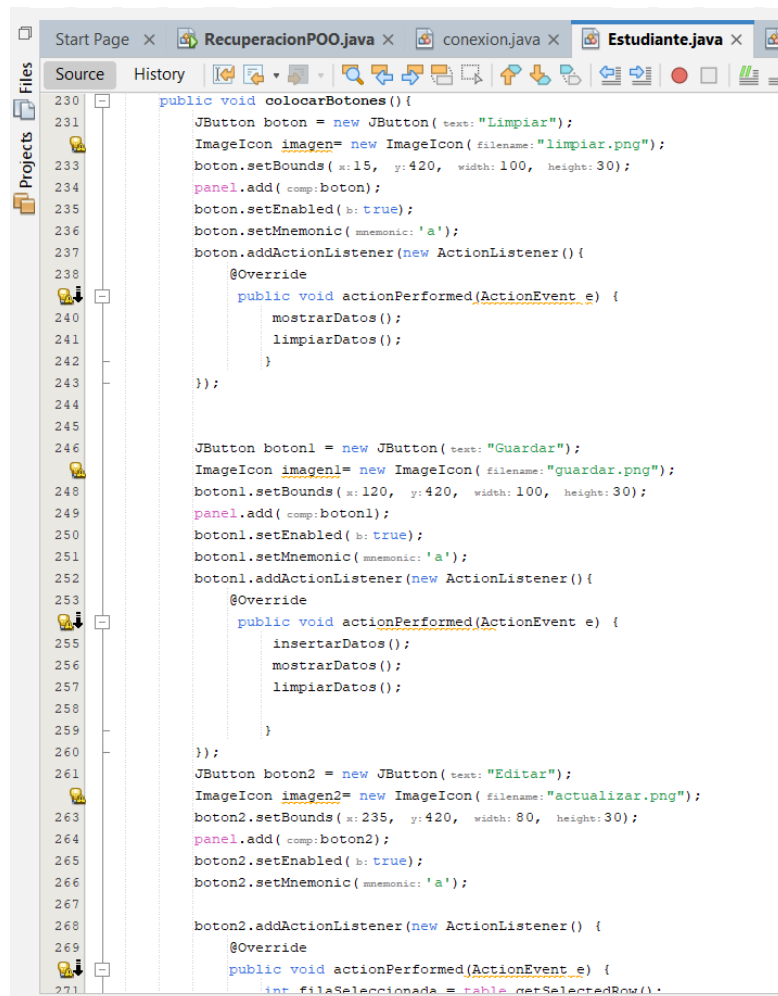
Paso 7: Configurar la tabla

En el método `tabla()`, se configura una tabla para mostrar los datos de los estudiantes. Se crea un modelo de tabla personalizado y se asocia con la tabla. También se llama a la función `mostrarDatos()` para cargar los datos en la tabla.

```
217 public void tabla(){
218     //ponemos los nombres a nuestra columnas
219     String[] columnNames= {"Codigo","Cedula","Nombre","Apellido","Mail","Telefono","Direccion","Materia","Estado"};
220     DefaultTableModel tableModel= new DefaultTableModel( columnNames, rowCount: 0);
221     //creamos la tabla
222     JTable table= new JTable( dm: tableModel);
223     //ubicacion
224     JScrollPane scrollPane= new JScrollPane( view: table);
225     scrollPane.setBounds( x: 400, y: 80, width: 600, height: 250);
226     panel.add( comp: scrollPane);
227     mostrarDatos();
228 }
229 }
```

Paso 8: Colocar botones

El método `colocarBotones()` agrega varios botones a la interfaz, como "Limpiar", "Guardar", "Editar", "Eliminar" y "Salir". Cada botón tiene un `ActionListener` que define su comportamiento cuando se hace clic.



```
230 public void colocarBotones(){
231     JButton boton = new JButton( text: "Limpiar");
232     ImageIcon imagen= new ImageIcon( filename: "limpiar.png");
233     boton.setBounds( x: 15, y: 420, width: 100, height: 30);
234     panel.add( comp: boton);
235     boton.setEnabled( b: true);
236     boton.setMnemonic( mnemonic: 'a');
237     boton.addActionListener( new ActionListener() {
238         @Override
239         public void actionPerformed(ActionEvent e) {
240             mostrarDatos();
241             limpiarDatos();
242         }
243     });
244
245     JButton boton1 = new JButton( text: "Guardar");
246     ImageIcon imagen1= new ImageIcon( filename: "guardar.png");
247     boton1.setBounds( x: 120, y: 420, width: 100, height: 30);
248     panel.add( comp: boton1);
249     boton1.setEnabled( b: true);
250     boton1.setMnemonic( mnemonic: 'a');
251     boton1.addActionListener( new ActionListener() {
252         @Override
253         public void actionPerformed(ActionEvent e) {
254             insertarDatos();
255             mostrarDatos();
256             limpiarDatos();
257         }
258     });
259
260     JButton boton2 = new JButton( text: "Editar");
261     ImageIcon imagen2= new ImageIcon( filename: "actualizar.png");
262     boton2.setBounds( x: 235, y: 420, width: 80, height: 30);
263     panel.add( comp: boton2);
264     boton2.setEnabled( b: true);
265     boton2.setMnemonic( mnemonic: 'a');
266
267     boton2.addActionListener( new ActionListener() {
268         @Override
269         public void actionPerformed(ActionEvent e) {
270             int fileSeleccionada = table.getSelectedRow();
```

Paso 9: Insertar datos en la base de datos

El método insertarDatos() se utiliza para insertar los datos ingresados por el usuario en la base de datos MySQL.

```
316 public void insertarDatos() {
317     try {
318         String SQL = "INSERT INTO estudiante(estu_cedula,estu_nombre, estu_apellido,"
319             + " estu_mail,estu_telefono,estu_direccion,estu_materia,estu_estado) VALUES (?, ?, ?, ?, ?, ?, ?,?)";
320         PreparedStatement pst = (PreparedStatement) con.prepareStatement( string:SQL);
321
322         pst.setString( parameterIndex: 1,  x: txtCedula.getText());
323         pst.setString( parameterIndex: 2,  x: txtNombre.getText());
324         pst.setString( parameterIndex: 3,  x: txtApellido.getText());
325         pst.setString( parameterIndex: 4,  x: txtMail.getText());
326         pst.setString( parameterIndex: 5,  x: txtTelefono.getText());
327         pst.setString( parameterIndex: 6,  x: txtDireccion.getText());
328         int seleccion1 = cboMateria.getSelectedIndex();
329         pst.setString( parameterIndex: 7,  x: cboMateria.getItemAt( index: seleccion1));
330         int seleccion2 = cboEstado.getSelectedIndex();
331         pst.setString( parameterIndex: 8,  x: cboEstado.getItemAt( index: seleccion2));
332         pst.execute();
333         JOptionPane.showMessageDialog( parentComponent: null,  message: "Registro guardado exitosamente");
334     } catch (Exception e) {
335         JOptionPane.showMessageDialog( parentComponent: null,  "Error al guardar el registro: " + e);
336     }
337 }
```

Paso 10: Limpiar los campos

El método limpiarDatos() se encarga de restablecer todos los campos de entrada a sus valores iniciales después de realizar una acción.

```
338 public void limpiarDatos() {
339     txtCedula.setText( t: "");
340     txtNombre.setText( t: "");
341     txtApellido.setText( t: "");
342     txtMail.setText( t: "");
343     txtTelefono.setText( t: "");
344     txtDireccion.setText( t: "");
345     cboMateria.setSelectedItem( anObject: null);
346     cboEstado.setSelectedItem( anObject: null);
347 }
```

Paso 11: Mostrar datos en la tabla

El método mostrarDatos() recupera los registros de la base de datos y los muestra en la tabla.

```
349 public void mostrarDatos() {
350     String titulos[]={"Codigo","Cedula","Nombres","Apellidos","Mail","Telefono","Direccion","Materia","Estado"};
351     String registro[] = new String [9];
352     DefaultTableModel modelo=new DefaultTableModel( data: null,  columnNames: titulos);
353     String SQL = "SELECT * FROM 'estudiante'";
354     try{
355         Statement st=(Statement) con.createStatement();
356         ResultSet rs=st.executeQuery( string:SQL);
357         while(rs.next()) {
358             registro [0]=rs.getString( string: "estu_codigo");
359             registro [1]=rs.getString( string: "estu_cedula");
360             registro [2]=rs.getString( string: "estu_nombre");
361             registro [3]=rs.getString( string: "estu_apellido");
362             registro [4]=rs.getString( string: "estu_mail");
363             registro [5]=rs.getString( string: "estu_telefono");
364             registro [6]=rs.getString( string: "estu_direccion");
365             registro [7]=rs.getString( string: "estu_materia");
366             registro [8]=rs.getString( string: "estu_estado");
367
368             modelo.addRow( conData: registro);
369         }
370         table.setModel( defaultModel: modelo);
371     } catch (Exception e){
372         JOptionPane.showMessageDialog( parentComponent: null,  "Error al mostrar los datos: "+e);
373     }
374 }
```

Paso 12: Eliminar datos

El método `eliminarDatos()` elimina un registro seleccionado de la base de datos y actualiza la tabla.

```
376 private void eliminarDatos(int filaSeleccionada) {
377     String codigoEstudiante = table.getValueAt(row: filaSeleccionada, column: 0).toString();
378     String SQL = "DELETE FROM estudiante WHERE estu_codigo = ?";
379     try {
380         PreparedStatement pst = (PreparedStatement) con.prepareStatement(string: SQL);
381         pst.setString(parameterIndex: 1, x: codigoEstudiante);
382         pst.executeUpdate();
383         JOptionPane.showMessageDialog(parentComponent: null, message: "Registro eliminado correctamente.");
384     } catch (Exception e) {
385         JOptionPane.showMessageDialog(parentComponent: null, "Error al eliminar el registro: " + e);
386     }
387     mostrarDatos();
388 }
389 }
```

Paso 13: Editar datos en la tabla

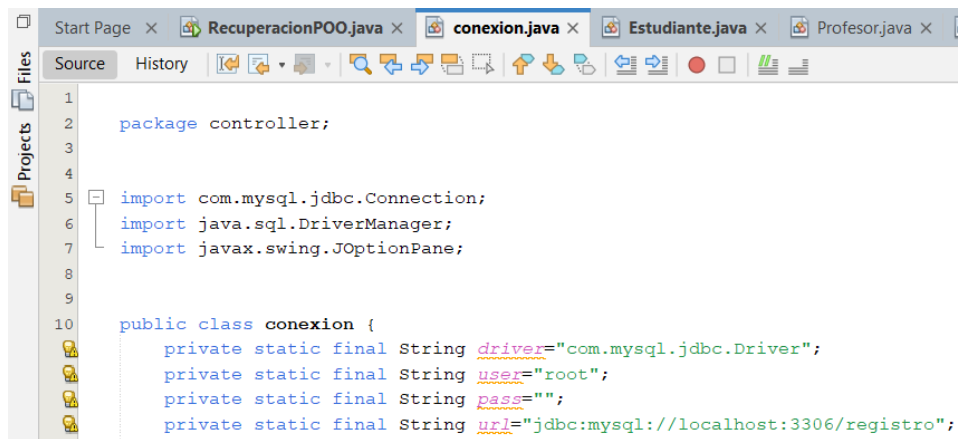
El método `editarRegistroEnTabla()` permite editar los valores directamente en la tabla.

```
389 private void editarRegistroEnTabla(int fila) {
390     DefaultTableModel model = (DefaultTableModel) table.getModel();
391     model.setValueAt(aValue: txtCedula.getText(), row: fila, column: 1);
392     model.setValueAt(aValue: txtNombre.getText(), row: fila, column: 2);
393     model.setValueAt(aValue: txtApellido.getText(), row: fila, column: 3);
394     model.setValueAt(aValue: txtMail.getText(), row: fila, column: 4);
395     model.setValueAt(aValue: txtTelefono.getText(), row: fila, column: 5);
396     model.setValueAt(aValue: txtDireccion.getText(), row: fila, column: 6);
397     model.setValueAt(aValue: cboMateria.getSelectedItemAt(), row: fila, column: 7);
398     model.setValueAt(aValue: cboEstado.getSelectedItemAt(), row: fila, column: 8);
399 }
400 }
401 }
402 }
```

Clase conexión

Paso 14: Declaración de variables de conexión

El código comienza declarando las variables necesarias para establecer la conexión, incluyendo el nombre del controlador (driver), el nombre de usuario (user), la contraseña (pass), y la URL de la base de datos (url).



```
1 package controller;
2
3
4
5 import com.mysql.jdbc.Connection;
6 import java.sql.DriverManager;
7 import javax.swing.JOptionPane;
8
9
10 public class conexion {
11     private static final String driver="com.mysql.jdbc.Driver";
12     private static final String user="root";
13     private static final String pass="";
14     private static final String url="jdbc:mysql://localhost:3306/registro";
```

Paso 16: Creación de la instancia de conexión, Capturar posibles excepciones y Retorno de la conexión

Dentro del método `conexion()`, se hace lo siguiente:

- Se carga la clase del controlador del driver MySQL utilizando `Class.forName(driver)`.
- Se establece la conexión utilizando `DriverManager.getConnection(url, user, pass)`. La URL proporciona la dirección de la base de datos y el puerto, y se utiliza el nombre de usuario y la contraseña para autenticarse.
- Dentro del bloque try-catch, cualquier excepción que ocurra durante la conexión se captura y se muestra un mensaje de error en caso de que la conexión falle.
- Finalmente, el método `conexion()` retorna el objeto de conexión creado, que es de tipo `Connection`.



```
16 public Connection conexion() {
17     try{
18         Class.forName( className:driver );
19         conectar=(Connection) DriverManager.getConnection(url,user, password:pass);
20     }catch(Exception e){
21         JOptionPane.showMessageDialog( parentComponent: null,"error de conexion "+e.getMessage());
22     }
23     return conectar;
24 }
25
26
27 }
```

Clase principal

```
15
16 public class RecuperacionPOO extends JFrame {
17
18     public RecuperacionPOO() {
19         setTitle( title: "Menú Principal");
20         setSize( width: 500, height: 200);
21         setDefaultCloseOperation( operation: EXIT_ON_CLOSE);
22
23         JMenuBar menuBar = new JMenuBar();
24         JMenu estudianteMenu = new JMenu( s: "Estudiantes");
25         JMenu profesorMenu = new JMenu( s: "Profesores");
26         JMenu horarioMenu = new JMenu( s: "Horarios");
27
28         JMenuItem estudianteItem = new JMenuItem( text: "Estudiantes");
29         estudianteItem.addActionListener(new ActionListener() {
30             @Override
31             public void actionPerformed(ActionEvent e) {
32                 Estudiante estudianteFrame = new Estudiante();
33                 estudianteFrame.setVisible( b: true);
34             }
35         });
36
37         JMenuItem profesorItem = new JMenuItem( text: "Profesores");
38         profesorItem.addActionListener(new ActionListener() {
39             @Override
40             public void actionPerformed(ActionEvent e) {
41                 Profesor profesorFrame = new Profesor();
42                 profesorFrame.setVisible( b: true);
43             }
44         });
45
46         JMenuItem horarioItem = new JMenuItem( text: "Horarios");
47         horarioItem.addActionListener(new ActionListener() {
48             @Override
49             public void actionPerformed(ActionEvent e) {
50                 horario horarioFrame = new horario();
51                 horarioFrame.setVisible( b: true);
52             }
53         });
54
55         estudianteMenu.add( menuItem: estudianteItem);
56         profesorMenu.add( menuItem: profesorItem);
57         horarioMenu.add( menuItem: horarioItem);
58
59         menuBar.add( e: estudianteMenu);
60         menuBar.add( e: profesorMenu);
61         menuBar.add( e: horarioMenu);
62
63         setJMenuBar( menubar: menuBar);
64     }
65
66     public static void main(String[] args) {
67         SwingUtilities.invokeLater(new Runnable() {
68             @Override
69             public void run() {
70
71                 RecuperacionPOO re = new RecuperacionPOO();
72                 re.setVisible( b: true);
73             }
74         });
75     }
76 }
```


Ejecución

The screenshot shows a Java Swing window titled "Estudiante" with a pink background. On the left, there is a form with labels and input fields for "Cedula:", "Nombre:", "Apellido:", "Mail:", "Telefono:", "Direccion:", "Materia:", and "Estado:". The "Materia:" and "Estado:" fields are dropdown menus. On the right, there is a table with the following data:

Codigo	Cedula	Nombres	Apellidos	Mail	Telefono	Direccion	Materia	Estado
6	17259645...	Eduardo	Ordoñez	Eordo_12...	994545878	Los Rosal...	Calculo	Activo

At the bottom of the window, there are five buttons: "Limpiar", "Guardar", "Editar", "Eliminar", and "Salir".

Conclusión

El uso de tablas en programación orientada a objetos nos brinda una mejor visión de los datos que ingresados que nos muestra en por columnas y en orden los mismo, para hacer uso de esta uno se puede guiar con el Api Java la nos proporciona todas las librerías y muestras de unos de las mismo en el caso de la tabla usamos el TableModel y JTable.