Universidad Politécnica de Sinaloa

MANUAL DE “BIBLIOTECH”

Alumnos del TCI 7-2

García Bustamante Gael Alejandro

Macias Manzano Manuel Alejandro

Quintero Herrera Irving Zuriel

Ramírez Zatarain José Manuel

Rosas García Bernardo

Introducción:

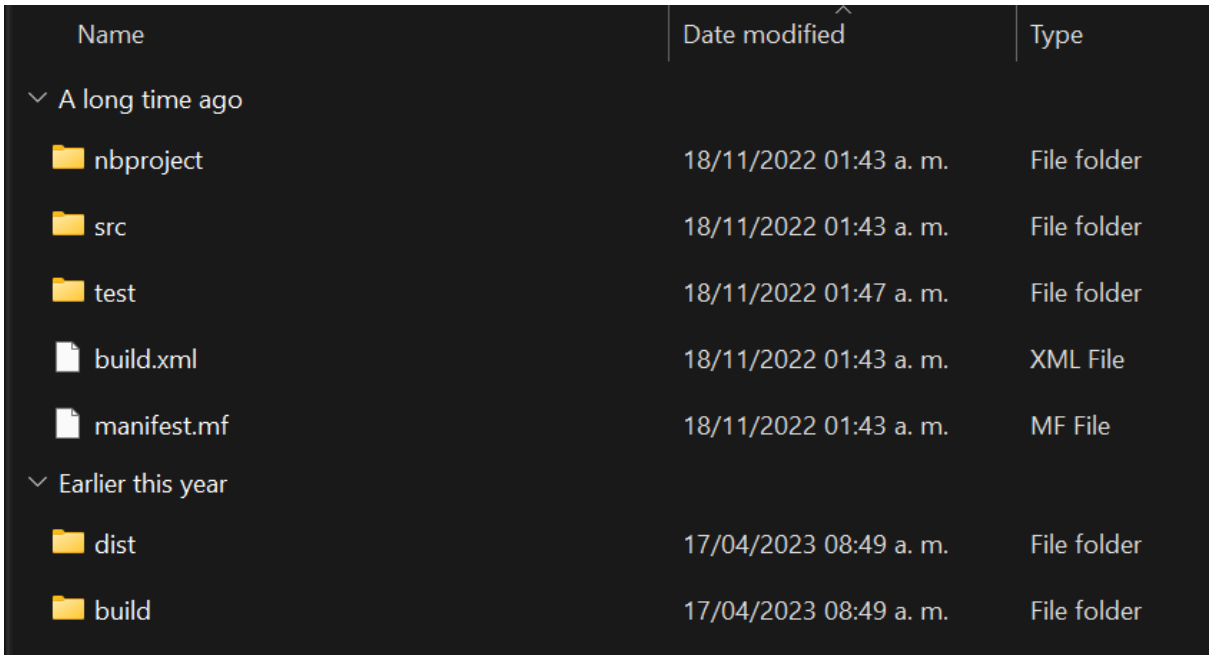
El sistema que has desarrollado en Java NetBeans es una aplicación de registro de usuarios que utiliza una base de datos MySQL para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar). La estructura del proyecto se organiza dentro de la carpeta src, que contiene subcarpetas como img y panel, donde se almacenan recursos gráficos y archivos relacionados con la interfaz gráfica del usuario (GUI).

Requisitos Previos:

Enumera los requisitos previos necesarios para trabajar en el proyecto. Esto podría incluir conocimientos básicos de Java, IDE (puede ser NetBeans) y cualquier otra herramienta específica que se utilice.
-JAVA
-MYSQL

Estructura del Proyecto:

Estos son todos los módulos que se usan en el sistema



Donde:

nbproject: Contiene los archivos y configuraciones específicos de NetBeans. Puedes ignorar esto en tu manual, ya que los usuarios finales generalmente no necesitan interactuar con estos archivos.

src: Aquí es donde se encuentran tus archivos fuente (código fuente) del proyecto. Asegúrate de entender qué clases y paquetes contiene tu aplicación.

test: Puede contener pruebas unitarias para tu código. No siempre es necesario incluir esto en el manual de usuario, a menos que los usuarios deban realizar pruebas específicas.

build.xml: Este archivo es utilizado por Apache Ant para compilar y construir tu proyecto. Los usuarios finales generalmente no necesitan interactuar con este archivo.

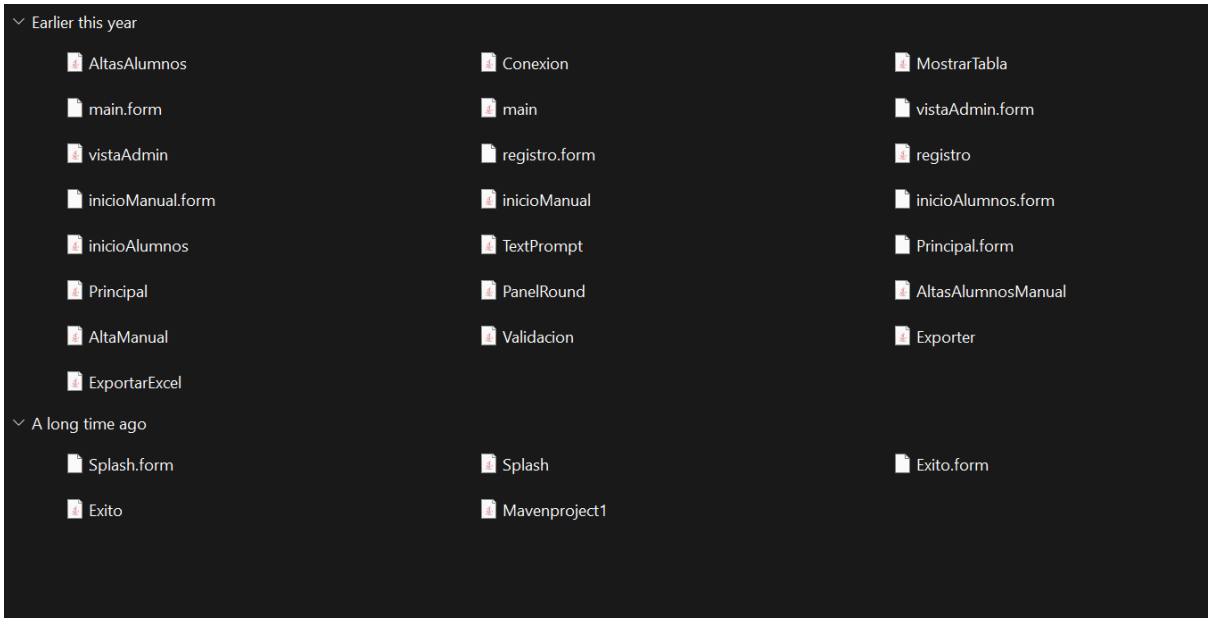
manifest.mf: Este archivo contiene información sobre la aplicación, como el nombre de la clase principal. No es necesario para el usuario final.

dist: Aquí es donde se colocan los archivos generados después de la compilación. Es posible que los usuarios finales deban buscar el archivo ejecutable de aquí para ejecutar la aplicación.

build: Directorio temporal utilizado durante el proceso de construcción. (Puedes ignorar esto en el manual)

Módulo src:

Dentro de este módulo vas a encontrar una carpeta llamada img la cual contiene todas las imágenes del sistema y otra carpeta llamada panel la cual contiene una variedad de archivos relacionados con la interfaz gráfica y la lógica del sistema.



AltasAlumnos:

este código maneja la inserción segura de registros de estudiantes en una base de datos MySQL, asegurando la integridad de los datos y proporcionando mensajes informativos al usuario.

Explicación del Funcionamiento:

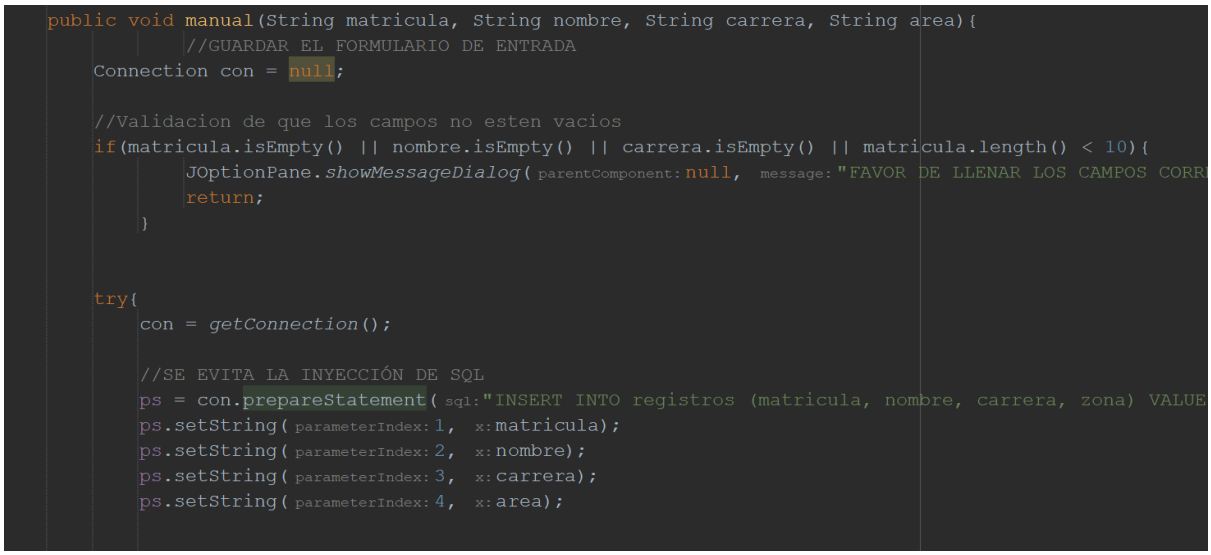
El código inicia definiendo constantes para la conexión a la base de datos, como la URL, el nombre de usuario y la contraseña. La clase `AltaManual` incluye un método estático `getConnection` que utiliza JDBC para establecer una conexión con la base de datos MySQL.

```
public static final String URL = "jdbc:mysql://localhost:3306/bibliotech";
public static final String USERNAME = "root";
public static final String PASSWORD = "";

PreparedStatement ps;
ResultSet rs;

public static Connection getConnection(){
    Connection con = null;

    try{
        Class.forName( className:"com.mysql.cj.jdbc.Driver");
        con = (Connection)DriverManager.getConnection( url:URL,  user:USERNAME,  password:PASSWORD);
        System.out.println( x:"CONEXION EXITOSA");
    } catch (Exception e){
        System.out.println("Error de conexion"+e);
    }
    return con;
}//Fin de la conexion
```

La función principal de esta clase es el método `manual`, que recibe información sobre la matrícula, nombre, carrera y área de un estudiante. Antes de procesar la

información, se realizan validaciones para asegurarse de que los campos no estén vacíos y que la matrícula tenga una longitud mínima requerida.

Luego, se establece una conexión a la base de datos y se prepara una declaración SQL para insertar el registro en la tabla `registros`. Se utilizan parámetros en la consulta para evitar la inyección de SQL.



Después de ejecutar la inserción, se muestra un mensaje indicando si la operación fue exitosa.

Finalmente, independientemente del resultado de la inserción, se crea una instancia de las clases `inicioAlumnos` e `inicioManual`. La primera muestra una pantalla de registro automático, y la segunda se cierra, proporcionando así una transición en la interfaz de usuario.

Conexión:

Este código, se encarga de facilitar la conexión de una aplicación Java a una base de datos MySQL. Su función principal es como una especie de "puente" que permite que la aplicación envíe y reciba información de la base de datos de manera segura y confiable.

Establece la Conexión: Utilizando la información sobre la base de datos (nombre, ubicación, usuario y contraseña), el código establece una conexión para que la aplicación pueda almacenar y recuperar datos de la base de datos.

Manejo de Errores: Si ocurre algún problema durante la conexión, el código informa al usuario que algo salió mal, proporcionando mensajes comprensibles sobre la situación.

```
public Connection conectar(){
    try {
        Class.forName( className:driver);
        cx = DriverManager.getConnection(url+bd, user, password);
        System.out.println( x:"Conexion exitosa");
    } catch (ClassNotFoundException |SQLException ex) {
        System.out.println( x:"No se pudo establecer conexion");
        Logger.getLogger( name:Conexion.class.getName()).log( level:Level.SEVERE,  msg:null

    }
    return cx;
}

public void desconectar(){
    try {
        cx.close();
    } catch (SQLException ex) {
        Logger.getLogger( name:Conexion.class.getName()).log( level:Level.SEVERE,  msg:null
    }
```

MostrarTabla:

El código proporciona una estructura para la conexión con una base de datos MySQL y la generación de consultas SQL dinámicas basadas en los parámetros proporcionados. Esto facilita la obtención de datos específicos de la base de datos según los criterios proporcionados.

1. Conexión con la Base de Datos:

```
public class MostrarTabla {
    //Conexion con DB
    public static final String URL = "jdbc:mysql://localhost:3306/bibliotech";
    public static final String USERNAME = "root";
    public static final String PASSWORD = "";

    static Connection con = null;
    ResultSet rs;
```

Aquí se definen las constantes para la conexión a la base de datos MySQL. La URL especifica la ubicación de la base de datos, y las credenciales (nombre de usuario y contraseña) se proporcionan para autenticar la conexión.

```
    static Connection con = null;
    ResultSet rs;
```

Se declara una variable estática `con` de tipo `Connection` que se utilizará para establecer la conexión con la base de datos. También se declara una variable `rs` de tipo `ResultSet` para almacenar el resultado de una consulta SQL.

```
    public static Connection getConnection(){

        try{
            Class.forName( className:"com.mysql.cj.jdbc.Driver");
            con = (Connection)DriverManager.getConnection( url:URL,  user:USERNAME,  password:P
            System.out.println( x:"CONEXION EXITOSA");
        } catch (Exception e){
            System.out.println("Error de conexion"+e);
        }
        return con;
```

En este método estático llamado `getConnection`, se intenta cargar el controlador JDBC para MySQL y establecer la conexión utilizando las constantes definidas anteriormente. Si la conexión es exitosa, se imprime un mensaje de éxito; de lo contrario, se imprime un mensaje de error. La conexión se devuelve como resultado.

2. Generación de Consultas SQL:

```
public String consulta(String sql, String matricula, String zona, String fecha, Stri
    //CONSULTA DE DATOS DEPENDIENDO DE QUE CAMPOS CONTENGA

        //EN CASO DE QUE SOLO EL CAMPO DE MATRICULA TEGA DATOS
        if("".equals( anObject: fecha) && !"".equals( anObject: matricula) && "".equals( anOb

            sql = sql + " WHERE matricula = " + matricula + " ORDER BY nombre";
        }
        //EN CASO DE QUE SOLO EL CAMPO DE ZONA CONTENGA DATOS
        else if("".equals( anObject: fecha) && "".equals( anObject: matricula) && !"".equals

            zona = "'" + zona + "'";//MAÑOSA PARA QUE LA CONSULTA SQL DETECTARA LA VA
            sql = sql + " WHERE zona = " + zona + " ORDER BY nombre";
        }

        //EN CASO DE QUE SOLO EL CAMPO DE FECHA INICIO CONTENGA DATOS
        else if(!"".equals( anObject: fecha) && "".equals( anObject: fechafin) && "".equals
```

```
        //EN CASO DE QUE NINGUN CAMPO CONTENGA DATOS
        else if("".equals( anObject: matricula) && "".equals( anObject: zona) && "".equals( an
            sql = sql + " ORDER BY nombre";
        }

        //RETORNA LA CONSULTA SQL COMPLETA
        return sql;
}
```

Este método llamado `consulta` toma una consulta SQL base (`sql`) y varios parámetros relacionados con los campos de la base de datos (matrícula, zona, fecha, etc.). La lógica condicional dentro de este método construye dinámicamente la consulta SQL basándose en los valores proporcionados para los diferentes campos.

3. Ejemplo de Uso:

```
MostrarTabla mostrarTabla = new MostrarTabla();
String consultaSQL = mostrarTabla.consulta( sql: "SELECT * FROM tabla",  matricula:
```

Este código crea una instancia de la clase `MostrarTabla` y llama al método `consulta` para obtener una consulta SQL basada en ciertos parámetros. La consulta resultante puede ser utilizada para recuperar datos específicos de la base de datos.

main.form y main:

El código crea una interfaz gráfica para un programa de inicio de sesión. Utiliza componentes como campos de texto, botones e imágenes para proporcionar una experiencia de usuario. Los eventos son manejados para realizar acciones específicas en respuesta a las interacciones del usuario.

1. Creación de la Interfaz Gráfica:

El código define una clase llamada `main` que extiende `javax.swing.JFrame`. En este caso, `JFrame` se utiliza para crear la ventana principal de la aplicación.

```
public class main extends javax.swing.JFrame {

    public main() {
        initComponents();
        this.setLocationRelativeTo( c:null);

        //Se coloca el PlaceHolder
        TextPrompt users = new TextPrompt( text:" Ingrese su Usuario", component:txt_Use
        TextPrompt matri = new TextPrompt( text:" Ingrese su Contraseña", component:txt_
    }
```

Este método se encarga de inicializar todos los componentes de la interfaz gráfica. Los componentes son creados y configurados en este método, y se invoca automáticamente al instanciar la clase.

3. Configuración de Eventos y Comportamientos:

```
    private void txt_UserActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void btn_LimpiarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        txt_User.setText( t:"");
        txt_Password.setText( t:"");
    }

    private void btn_ListoActionPerformed(java.awt.event.ActionEvent evt) {
        //Se valida para pasar al apartado de administrador
        Validacion val = new Validacion();
        val.validacion( user:txt_User.getText(),  pass:txt_Password.getText());


    }
```

Estos métodos manejan eventos específicos que ocurren en la interfaz gráfica, como hacer clic en botones o ingresar texto. Cada método tiene un bloque de código que define cómo debe responder la interfaz ante ese evento.

4. Constructor y Configuración Inicial:

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
     Look and feel setting code (optional)
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new main().setVisible( b:true);
        }
    });
}
```

El constructor de la clase (`main()`) llama al método `initComponents()` para inicializar la interfaz y realiza algunas configuraciones adicionales, como centrar la ventana en la pantalla.

El método `main` es el punto de entrada de la aplicación. Inicia la interfaz dentro del hilo de despacho de eventos de Swing para garantizar un manejo seguro de la interfaz gráfica.

Si tienes alguna pregunta específica sobre alguna parte del código o si necesitas más detalles, no dudes en preguntar.

vistaAdmin.form y vistaAdmin:

Este código es parte de un programa en Java que utiliza el entorno de desarrollo de Swing para crear una interfaz gráfica de usuario (GUI). El programa parece ser una aplicación de gestión de registros relacionados con matrículas en una institución educativa o similar. A continuación, se proporciona una explicación detallada de las partes clave del código:

1. Importaciones de paquetes y clases:

```java
package panel;
import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;
import static panel.vistaAdmin.getConnection;
```

En esta sección, se importan diferentes clases y paquetes necesarios para la aplicación, como clases para la interfaz gráfica (Swing), manipulación de archivos, manipulación de bases de datos (JDBC), y otras utilidades.

2. Declaración de variables y conexión a la base de datos:

```java
public class vistaAdmin extends javax.swing.JFrame {


    //Conexion con DB
    public static final String URL = "jdbc:mysql://localhost:3306/bibliotech";
    public static final String USERNAME = "root";
    public static final String PASSWORD = "";

    PreparedStatement ps;
    ResultSet rs;
    static Connection con = null;
```

Se declaran variables para la conexión a la base de datos MySQL, como la URL, el nombre de usuario y la contraseña. También se declaran objetos relacionados con la conexión a la base de datos.

3. Método para obtener la conexión a la base de datos:

```java
public static Connection getConnection(){


    try{
        Class.forName(className: "com.mysql.cj.jdbc.Driver");
        con = (Connection)DriverManager.getConnection(url:URL, user:USERNAME, password:PASSWORD);
        System.out.println(x: "CONEXION EXITOSA");
    } catch (Exception e){
        System.out.println("Error de conexion"+e);
    }
    return con;
}//Fin de la conexion
```

Este método estático devuelve una conexión a la base de datos. Utiliza el controlador JDBC de MySQL para establecer la conexión.

4. Constructor de la clase `vistaAdmin`:

```java
public vistaAdmin() {
    initComponents();
    this.setLocationRelativeTo(c: null);


    //Se coloca el PlaceHolder
    TextPrompt matri = new TextPrompt(text: " INGRESAR MATRÍCULA",component: txt_matricula);
}
```

El constructor inicializa la interfaz gráfica, centra la ventana en la pantalla y crea un "TextPrompt" (un tipo de marcador de posición) para el campo de texto `txt_matricula`.

5. Método `mostrar` para visualizar datos en la tabla:

```java
public void mostrar (String tabla){
    String fecha, matricula, fechafin, zona, carrera;
    int num = 1;
    Connection con = null;
    String sql = "SELECT * FROM " + tabla;
    Statement st;
    System.out.println(x: sql);
    DefaultTableModel model = new DefaultTableModel();

    //SE COLOCA EL NOMBRE A LOS CAMPOS A MOSTRAR EN LA TABLA
    model.addColumn(columnName: "LISTADO");
    model.addColumn(columnName: "MATRÍCULA");
    model.addColumn(columnName: "NOMBRE");
    model.addColumn(columnName: "CARRERA");
    model.addColumn(columnName: "ZONA");
    model.addColumn(columnName: "FECHA");
    model.addColumn(columnName: "HORA DE ENTRADA");
    visor.setModel(dataModel: model);


    String [] datos = new String[7];
    try{
        con = getConnection();
        st = con.createStatement();
```

```
                /*SE COLOCA POR QUE SE ESTA CONSULTANDO UN ARREGLO DE DATOS
                EL NUMERO DE LA IZQUIERDA INDICA LA POSICIÓN DEL DATO EN LA TABLA*/
                datos[0] = Integer.toString(i: num);
                datos[1] = rs.getString(columnIndex:2);
                datos[2] = rs.getString(columnIndex:3);
                datos[3] = rs.getString(columnIndex:4);
                datos[4] = rs.getString(columnIndex:5);
                datos[5] = rs.getString(columnIndex:6);
                datos[6] = rs.getString(columnIndex:7);
                model.addRow(rowData:datos);
                num++;
            }
        }catch(Exception e){
            JOptionPane.showMessageDialog(parentComponent:null, message:"ERROR AL CONSULTAR INTENTELO NUEVAMENTE");
            System.err.println(x: e);
        }
    }
```

Este método realiza una consulta a la base de datos y muestra los resultados en una tabla en la interfaz gráfica. También se filtran los resultados según los valores ingresados en campos como `matricula`, `carrera`, `zona`, `fecha`, y `fechafin`.

6. Método principal (`main`):

```
    public static void main(String args[]) {

        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new vistaAdmin().setVisible(b: true);

            }
        });

    }
```

Este método principal inicia la aplicación al crear una instancia de la clase vistaAdmin` y hacerla visible.

7. Componentes de la interfaz gráfica:

Se definen y configuran varios componentes de la interfaz gráfica, como botones, campos de texto, etiquetas, y tablas. Se utilizan clases y métodos de Swing para diseñar y manejar la interfaz.

Registro.form y registro:

Este código es una implementación en Java de una interfaz gráfica de usuario (GUI) utilizando la biblioteca Swing para la creación de ventanas y componentes visuales. La aplicación parece estar relacionada con un sistema de registro de usuarios en una biblioteca, ya que hace referencia a la base de datos "bibliotech" y contiene campos como la matrícula, nombre y carrera.

A continuación, proporcionaré una descripción de las partes clave del código:

1. Importaciones:

```
1   package panel;
2   import java.sql.ResultSet;
3   import java.sql.Connection;
4   import java.sql.DriverManager;
5   import java.awt.Image;
6   import java.sql.PreparedStatement;
7   import javax.swing.Icon;
8   import javax.swing.ImageIcon;
9   import javax.swing.JLabel;
```

- Importa las clases necesarias para la manipulación de bases de datos (ResultSet, Connection, DriverManager, PreparedStatement).
- Importa clases de AWT y Swing para trabajar con imágenes y componentes visuales.

2. Declaración de variables de conexión a la base de datos:

```
18        public static final String URL = "jdbc:mysql://localhost:3306/bibliotech";
19        public static final String USERNAME = "root";
20        public static final String PASSWORD = "";
```

- Define constantes para la conexión a la base de datos MySQL.

3. Método de conexión a la base de datos:

```
25      public static Connection getConnection(){
26          Connection con = null;
27
28          try{
29              Class.forName(className: "com.mysql.cj.jdbc.Driver");
30              con = (Connection)DriverManager.getConnection(url:URL, user:USERNAME, password: PASSWORD);
31              System.out.println(x: "CONEXION EXITOSA");
32          } catch (Exception e){
33              System.out.println("Error de conexion"+e);
34          }
35          return con;
36      }
```

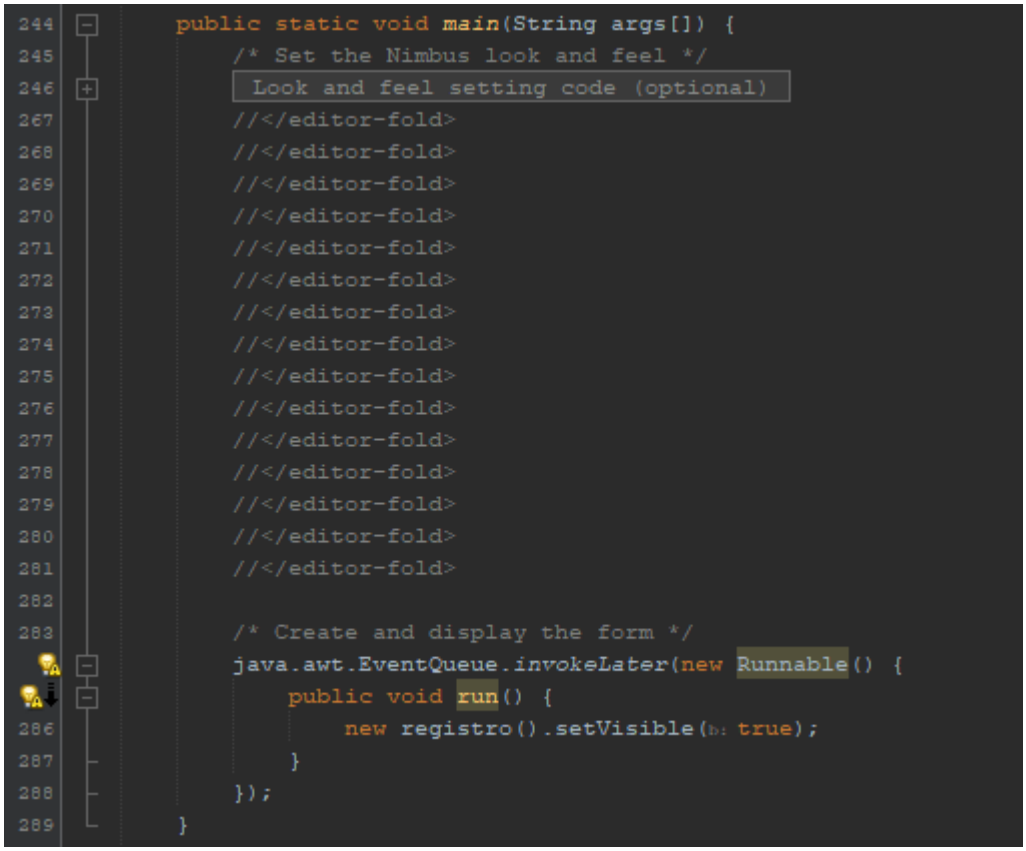
- Establece una conexión a la base de datos utilizando los valores de URL, nombre de usuario y contraseña definidos anteriormente.

4. Constructor de la clase:

```
40      public registro() {
41          initComponents();
42          this.setLocationRelativeTo(c: null);
43
44          //Se coloca el PlaceHolder
45          TextPrompt matri = new TextPrompt(text: " Ingrese su Matricula",component: txt_User);
46          TextPrompt nombre = new TextPrompt(text: " Ingrese su nombre",component: txt_nombre);
47          // TextPrompt carrera = new TextPrompt(" Ingrese su carrera",txt_carrera);
48      }
```

- Inicializa la interfaz gráfica y configura un "placeholder" en los campos de texto.

5. Método principal (main):



- Inicia la aplicación, creando una instancia de la clase `registro` y haciéndola visible.

6. Manejo de eventos:

- Hay métodos que manejan eventos, como el clic en botones

```
@SuppressWarnings("unchecked")
Generated Code

private void img_backMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

private void txt_nombreActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void txt_UserActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

private void btn_ListoActionPerformed(java.awt.event.ActionEvent evt) {...7 lines }

private void btn_LimpiarActionPerformed(java.awt.event.ActionEvent evt) {...5 lines

private void txt_UserKeyTyped(java.awt.event.KeyEvent evt) {...18 lines }

private void txt_nombreKeyTyped(java.awt.event.KeyEvent evt) {...9 lines }
```

7. Componentes visuales y ajuste de imágenes:

Utiliza varios componentes visuales de Swing como `JFrame`, `JPanel`, `JLabel`, JButton`, `JTextField`, `JComboBox`, etc., para diseñar la interfaz gráfica. y Utiliza un método `SetImageLabel` para ajustar imágenes a los tamaños de los `JLabel`.

```
private void SetImageLabel(JLabel labelName, String root){...6 lines }


// Variables declaration - do not modify
private javax.swing.JButton btn_Limpiar;
private javax.swing.JButton btn_Listo;
private javax.swing.JComboBox<String> cb_carrera;
private javax.swing.JLabel img_back;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel7;
private javax.swing.JPanel jPanel1;
private Clases.PanelRound panelRound2;
private javax.swing.JTextField txt_User;
private javax.swing.JTextField txt_nombre;
// End of variables declaration
```

En resumen, el código parece formar parte de una aplicación de registro de usuarios en una biblioteca, donde los usuarios pueden ingresar su matrícula, nombre y carrera. La aplicación utiliza una base de datos MySQL para almacenar la información de los usuarios.

inicioManual.forn y inicioManual:

El código funciona para qusuarioue los pueden ingresar su matrícula, nombre, área y carrera. La aplicación utiliza una base de datos MySQL para almacenar y recuperar la información de los usuarios

A continuación, proporcionaré una descripción de las partes clave del código:

1. Importaciones:

```
3  import java.sql.ResultSet;
4  import java.sql.Connection;
5  import java.sql.DriverManager;
6  import java.awt.Image;
7  import java.sql.PreparedStatement;
8  import javax.swing.Icon;
9  import javax.swing.ImageIcon;
10 import javax.swing.JLabel;
11 import javax.swing.JOptionPane;
```

- Importa las clases necesarias para la manipulación de bases de datos (ResultSet, Connection, DriverManager, PreparedStatement).
- Importa clases de AWT y Swing para trabajar con imágenes y componentes visuales.
- Importa `JOptionPane` para mostrar mensajes de diálogo.

2. Declaración de variables de conexión a la base de datos:

```
20      public static final String URL = "jdbc:mysql://localhost:3306/bibliotech";
21      public static final String USERNAME = "root";
22      public static final String PASSWORD = "";
```

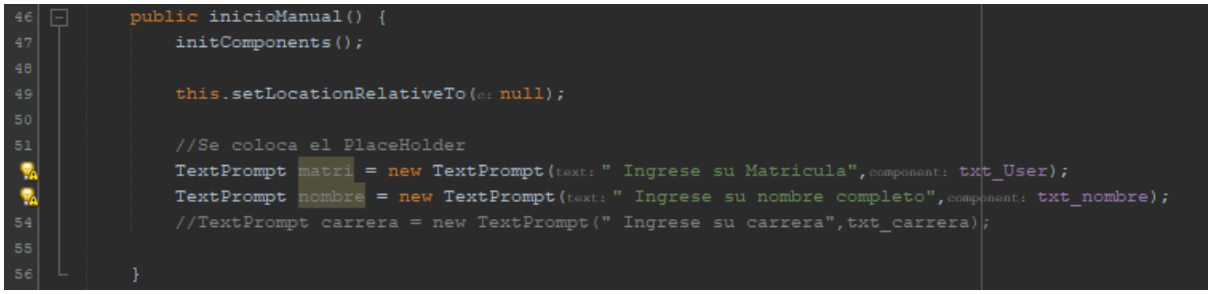
- Define constantes para la conexión a la base de datos MySQL.

3. Método de conexión a la base de datos:

```
29      public static Connection getConnection(){
30          Connection con = null;
31
32          try{
33              Class.forName(className: "com.mysql.cj.jdbc.Driver");
34              con = (Connection)DriverManager.getConnection(url:URL, user: USERNAME, password: PASSWORD);
35              System.out.println(x: "CONEXION EXITOSA");
            } catch (Exception e){
37              System.out.println("Error de conexion"+e);
38          }
39          return con;
40      }
```

- Establece una conexión a la base de datos utilizando los valores de URL, nombre de usuario y contraseña definidos anteriormente.

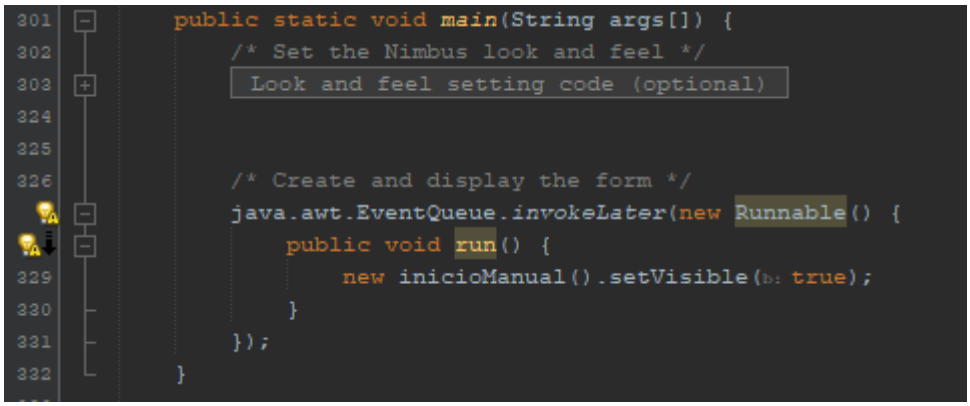
4. Constructor de la clase:



```
46      public inicioManual() {
47          initComponents();
48
49          this.setLocationRelativeTo(c: null);
50
51          //Se coloca el PlaceHolder
            TextPrompt matri = new TextPrompt(text: " Ingrese su Matricula",component: txt_User);
            TextPrompt nombre = new TextPrompt(text: " Ingrese su nombre completo",component: txt_nombre);
54          //TextPrompt carrera = new TextPrompt(" Ingrese su carrera",txt_carrera);
55
56      }
```

- Inicializa la interfaz gráfica y configura un "placeholder" en los campos de texto.

5. Método principal (main):



```
301     public static void main(String args[]) {
302         /* Set the Nimbus look and feel */
303         Look and feel setting code (optional)
324
325
326         /* Create and display the form */
            java.awt.EventQueue.invokeLater(new Runnable() {
                public void run() {
329                 new inicioManual().setVisible(b: true);
330             }
331         });
332     }
```

- Inicia la aplicación, creando una instancia de la clase `inicioManual` y haciéndola visible.

6. Manejo de eventos:

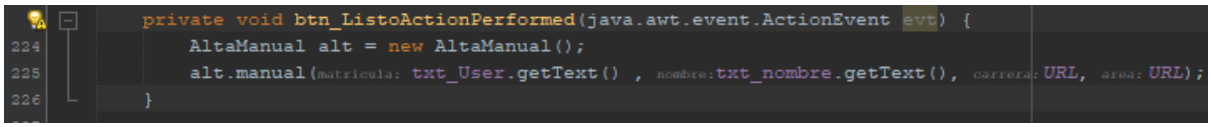
- Hay métodos que manejan eventos, como el clic en botones (`btn_LimpiarActionPerformed`, `btn_ListoActionPerformed`, `btn_buscarActionPerformed`) y acciones en campos de texto (`txt_UserKeyTyped`, `txt_nombreKeyTyped`).

7. Componentes visuales:
- Utiliza varios componentes visuales de Swing como `JFrame`, `JPanel`, `JLabel`, `JButton`, `JTextField`, `JComboBox`, etc., para diseñar la interfaz gráfica.

8. Ajuste de imágenes:

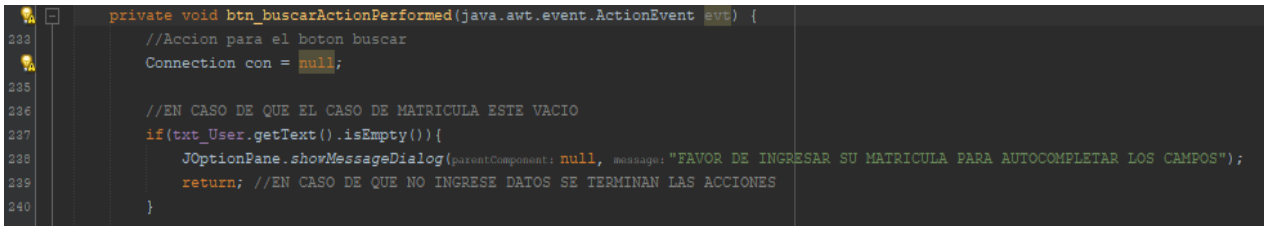
- Utiliza un método `SetImageLabel` para ajustar imágenes a los tamaños de los `JLabel`.

9. Acción del botón "Listo":



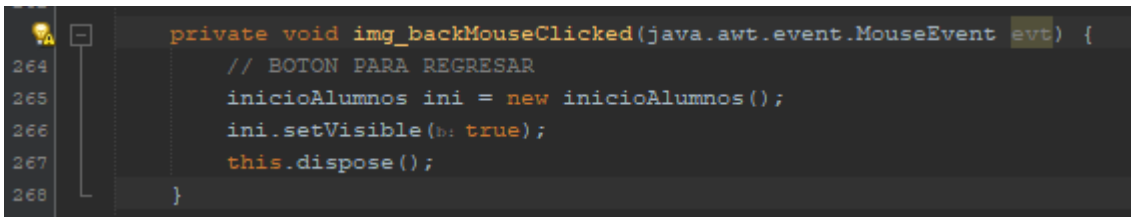
- Llama al método `manual` de la clase `AltaManual` pasando la matrícula y el nombre como parámetros.

10. Acción del botón "Buscar":



- Realiza una búsqueda en la base de datos utilizando la matrícula ingresada y autocompleta los campos si se encuentra un resultado.

11. Acción del botón de regreso:



inicioAlumnos.form y inicioAlumnos:

Este código crea otra interfaz gráfica para la aplicación, específicamente para la pantalla de inicio de sesión de alumnos. Al igual que el código anterior, utiliza componentes de Swing y maneja eventos para proporcionar una experiencia de usuario interactiva.

1. Creación de la Interfaz Gráfica y inicialización de Componentes:

La clase `inicioAlumnos` también extiende de `javax.swing.JFrame`, lo que significa que es otra interfaz gráfica.

```
public class inicioAlumnos extends javax.swing.JFrame {
    public inicioAlumnos() {
        initComponents();
        //SE COLOCA EL JFRAME EN EL CENTRO DE LA PANTALLA
        this.setLocationRelativeTo( c:null);

        //Se coloca el PlaceHolder
       TextPrompt matri = new TextPrompt( text:" FAVOR DE ESCANEAR SU MATRICULA", component:txt
    }
```

Este método se encarga de inicializar todos los componentes de la interfaz gráfica. Al igual que en el código anterior, los componentes son creados y configurados en este método.

3. Configuración de Eventos y Comportamientos:

```
private void img_backMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

private void btn_inicioManualActionPerformed(java.awt.event.ActionEvent evt) {...3 li

private void btn_inicioManualMouseClicked(java.awt.event.MouseEvent evt) {...6 lines

private void btn_registroActionPerformed(java.awt.event.ActionEvent evt) {...6 lines

private void txt_matriculaKeyTyped(java.awt.event.KeyEvent evt) {...17 lines }

private void txt_matriculaActionPerformed(java.awt.event.ActionEvent evt) {...3 lines

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {...9 lines }
```

Estos métodos manejan eventos específicos, como clics en botones, ingreso de texto, etc.

4. Constructor y Configuración Inicial:

```
public class inicioAlumnos extends javax.swing.JFrame {
    public inicioAlumnos() {
        initComponents();
        //SE COLOCA EL JFRAME EN EL CENTRO DE LA PANTALLA
        this.setLocationRelativeTo( c:null);

        //Se coloca el PlaceHolder
       TextPrompt matri = new TextPrompt( text:" FAVOR DE ESCANEAR SU MATRICULA",
    }
```

El constructor de la clase (`inicioAlumnos()`) llama al método `initComponents()` para inicializar la interfaz y realiza algunas configuraciones adicionales, como centrar la ventana en la pantalla. También se agrega un "PlaceHolder" al campo de matrícula.

5. Ejecución de la Aplicación:

```
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new inicioAlumnos().setVisible( b:true);
        }
    });
```

El método `main` es el punto de entrada de la aplicación y muestra la interfaz gráfica al ejecutar el programa.

textPrompt:

Este código define la clase `TextPrompt` facilita la adición de prompts a componentes de texto en una interfaz gráfica de usuario (GUI) de Java, brindando opciones de personalización en términos de cuándo y cómo se muestran estos prompts.

A continuación, desglosaré los elementos clave del código:

```
public TextPrompt(String text, JTextComponent component) {
        this(text, component,  show: Show.ALWAYS);
}

public TextPrompt(String text, JTextComponent component, Show show) {...20 lines

/**
 * Convenience method to change the alpha value of the current foreground
 * Color to the specifice value.
 *
 * @param alpha
 *            value in the range of 0 - 1.0.
```

- La clase tiene dos constructores. El primero toma un texto y un componente de texto como parámetros, utilizando `Show.ALWAYS` como valor predeterminado para el parámetro `show`.
- El segundo constructor permite especificar el comportamiento de cuándo se mostrará el prompt a través del parámetro `show`.

2. Método `checkForPrompt`:

```
    private void checkForPrompt() {
        // Text has been entered, remove the prompt

        if (document.getLength() > 0) {
            setVisible( aFlag:false);
            return;
        }

        // Prompt has already been shown once, remove it

        if (showPromptOnce && focusLost > 0) {
            setVisible( aFlag:false);
            return;
        }
```

```
    // Check the Show property and component focus to determine if the
    // prompt should be displayed.

    if (component.hasFocus()) {
        if (show == Show.ALWAYS || show == Show.FOCUS_GAINED)
            setVisible( aFlag:true);
        else
            setVisible( aFlag:false);
    } else {
        if (show == Show.ALWAYS || show == Show.FOCUS_LOST)
            setVisible( aFlag:true);
        else
            setVisible( aFlag:false);
    }
```

`

- Este método se encarga de verificar si el prompt debe ser visible en función de diversas condiciones, como si el documento asociado al componente de texto está vacío, si el prompt ya se mostró una vez, y la configuración de `show` (siempre, al obtener el enfoque, o al perder el enfoque).

3. Métodos para Cambios en Apariencia:

```
    public void changeAlpha(int alpha) {
            alpha = alpha > 255 ? 255 : alpha < 0 ? 0 : alpha;

            Color foreground = getForeground();
            int red = foreground.getRed();
            int green = foreground.getGreen();
            int blue = foreground.getBlue();

            Color withAlpha = new Color( r:red,  g:green,  b:blue,  a:alpha);
            super.setForeground( fg:withAlpha);
    }
```

```
     */
    public void changeStyle(int style) {
            setFont( font:getFont().deriveFont(style));
    }
```

- Estos métodos proporcionan la capacidad de cambiar la transparencia del color de primer plano (`changeAlpha`), cambiar el estilo de la fuente (`changeStyle`), y son métodos de conveniencia para ajustar la apariencia del prompt.

4. Implementación de Interfaces (`FocusListener` y `DocumentListener`):

- La clase implementa los métodos de estas interfaces para responder a eventos de enfoque y cambios en el documento asociado al componente de texto.

5. Getters y Setters:

```
    public Show getShow() {
            return show;
    }

    /** Set the prompt Show property to control when the promt is shown ...12 lines
    public void setShow(Show show) {
            this.show = show;
    }

    /** Get the showPromptOnce property ...5 lines */
    public boolean getShowPromptOnce() {
            return showPromptOnce;
    }

    /** Show the prompt once ...8 lines */
    public void setShowPromptOnce(boolean showPromptOnce) {
            this.showPromptOnce = showPromptOnce;
    }
```

Principal.form y `Principal`:

La clase `Principal` establece el panel principal de la interfaz gráfica, proporcionando botones para acceder a otras partes del programa. La transparencia y la configuración de eventos de clic en los botones están diseñadas para mejorar la apariencia y la funcionalidad de la interfaz.

1. Constructor:

```
    /** Creates new form Principal */
    public Principal() {
        initComponents();
        this.setLocationRelativeTo( c:null);


        //Transparencia de botones (NO BORRAR)
        trasparenciaButton();
    }
```

- El constructor inicializa los componentes de la interfaz llamando a `initComponents()`, establece la ubicación relativa del marco principal y configura la transparencia de los botones.

3. Método `btn_adminMouseClicked`:

```
private void btn_adminMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    main fra = new main();
    fra.setVisible( b:true);
    this.dispose();
}
```

- Este método se activa cuando se hace clic en el botón `btn_admin`. Crea una instancia de la clase `main` (presumiblemente otra parte de tu programa) y la hace visible, mientras oculta el marco principal actual.

4. Método `btn_alumnosMouseClicked`:

```
    private void btn_alumnosMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        inicioAlumnos ini = new inicioAlumnos();
        ini.setVisible( b:true);
        this.dispose();
    }
```

Similar al método anterior, este método se activa al hacer clic en el botón `btn_alumnos`. Crea una instancia de la clase `inicioAlumnos` y la hace visible, mientras oculta el marco principal actual.

5. Método `trasparenciaButton`:

```
    //Le da transparencia a los botones
    public void trasparenciaButton(){
        btn_admin.setOpaque( isOpaque: false);
        btn_admin.setContentAreaFilled( b: false);
        btn_admin.setBorderPainted( b: false);
        btn_alumnos.setOpaque( isOpaque: false);
        btn_alumnos.setContentAreaFilled( b: false);
        btn_alumnos.setBorderPainted( b: false);
        btn_inicio.setOpaque( isOpaque: false);
        btn_inicio.setContentAreaFilled( b: false);
        btn_inicio.setBorderPainted( b: false);
    }
```

Este método configura la transparencia de los botones (`btn_admin`, `btn_alumnos`, y `btn_inicio`). Hace que los botones sean transparentes y sin borde para mejorar la apariencia en la interfaz gráfica.

6. Método `SetImageLabel`:

```
//Adapta la imagen al tamaño del jlabel
private void SetImageLabel(JLabel labelName, String root){
    ImageIcon image = new ImageIcon( filename: root);
    Icon icon = new ImageIcon( image: image.getImage().getScaledInstance( width: labelName.g
    labelName.setIcon(icon);
    this.repaint();
}
```

- Este método toma un `JLabel` y la ruta de una imagen como parámetros, luego ajusta la imagen al tamaño del `JLabel` y la establece como ícono en el `JLabel`.

7. Método `main`:

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
     Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Principal().setVisible( b:true);
        }
    });
}
```

- El método `main` es el punto de entrada del programa. Crea una instancia de la clase `Principal` y la hace visible.

PanelRound:

Esta clase `PanelRound` proporciona un panel con esquinas redondeadas, y puedes personalizar el radio de redondeo para cada esquina de forma independiente. Puedes usar esta clase como una alternativa visualmente atractiva a un panel rectangular estándar en tus aplicaciones de interfaz gráfica en Java.

1. Atributos:

```
private int roundTopLeft = 0;
private int roundTopRight = 0;
private int roundBottomLeft = 0;
private int roundBottomRight = 0;
```

Estos son atributos que representan el radio de redondeo para cada una de las cuatro esquinas del panel. Los valores predeterminados son cero, lo que significa esquinas cuadradas.

2. Constructores:

```
public PanelRound() {
    setOpaque( isOpaque:false);
}
```

Este constructor inicializa la clase. Configura el panel como no opaco, lo que permite ver a través de él.

3. Métodos de Acceso:

```
public int getRoundTopLeft() {...3 lines }

public void setRoundTopLeft(int roundTopLeft) {...4 lines }

public int getRoundTopRight() {...3 lines }

public void setRoundTopRight(int roundTopRight) {...4 lines }

public int getRoundBottomLeft() {...3 lines }

public void setRoundBottomLeft(int roundBottomLeft) {...4 lines }

public int getRoundBottomRight() {...3 lines }

public void setRoundBottomRight(int roundBottomRight) {...4 lines }
```

4. Método `paintComponent`:

```
protected void paintComponent(Graphics grphcs) {
    Graphics2D g2 = (Graphics2D) grphcs.create();
    g2.setRenderingHint( hintKey: RenderingHints.KEY_ANTIALIASING, hintValue: RenderingHir
    g2.setColor( c: getBackground());
    Area area = new Area( s: createRoundTopLeft());
    if (roundTopRight > 0) {
        area.intersect(new Area( s: createRoundTopRight()));
    }
    if (roundBottomLeft > 0) {
        area.intersect(new Area( s: createRoundBottomLeft()));
    }
    if (roundBottomRight > 0) {
        area.intersect(new Area( s: createRoundBottomRight()));
    }
    g2.fill( s: area);
    g2.dispose();
    super.paintComponent( g: grphcs);
}
```

Este método se encarga de pintar el componente. Crea un objeto `Graphics2D` para pintar con antialiasing (suavizado de bordes). Luego, crea y llena un área que representa la forma del panel con esquinas redondeadas.

5. Métodos Privados:

```
    private Shape createRoundTopLeft() {...10 lines }

    private Shape createRoundTopRight() {...10 lines }

    private Shape createRoundBottomLeft() {...10 lines }

    private Shape createRoundBottomRight() {...10 lines }
}
```

Estos métodos crean y devuelven áreas que representan las esquinas redondeadas del panel. Cada método tiene en cuenta el radio de redondeo para la esquina correspondiente.

6. Configuración de las esquinas redondeadas:

En el método `paintComponent`, se crean áreas para cada esquina y se intersectan para obtener la forma final del panel con las esquinas redondeadas configuradas.

7. Actualización en Cambios de Radio:

Los métodos de acceso (`setRoundTopLeft`, `setRoundTopRight`, etc.) permiten cambiar dinámicamente el radio de redondeo de cada esquina, lo que resulta en una actualización visual del panel.

AltasAlumnosManual` y `AltaManual:

La clase `AltasAlumnosManual` está diseñada para gestionar el alta de alumnos en una base de datos MySQL en una aplicación de interfaz gráfica en Java. Proporciona un método para ingresar datos de alumnos, verificando que la matrícula y el nombre no estén vacíos. En caso de éxito, muestra un mensaje y redirige a la interfaz de inicio de alumnos; de lo contrario, notifica sobre un error en la operación. Este código utiliza JDBC para la conexión a la base de datos y componentes de Swing, proporcionando una interacción visual en la interfaz de usuario.

1. Importaciones:

```
package panel;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
```

`

- Se importan las clases necesarias para la gestión de bases de datos y la interfaz gráfica (JOptionPane).

2. Definición de la Clase y Constantes de Conexión a la Base de Datos:

```
 */
public class AltasAlumnosManual {
    public static final String URL = "jdbc:mysql://localhost:3306/bibliotech";
    public static final String USERNAME = "root";
    public static final String PASSWORD = "";

    PreparedStatement ps;
    ResultSet rs;
```

- Se declara la clase `AltasAlumnosManual` que se utiliza para gestionar altas de alumnos en la base de datos.
- Se definen las constantes para la URL de la base de datos, el nombre de usuario y la contraseña.

3. Variables de SQL:

```
PreparedStatement ps;
ResultSet rs;
```

- Se declaran variables para preparar consultas SQL y manejar resultados.

5. Método Estático de Conexión a la Base de Datos:

```
public static Connection getConnection(){
    Connection con = null;

    try{
        Class.forName( className: "com.mysql.cj.jdbc.Driver");
        con = (Connection)DriverManager.getConnection( url: URL,  user: USERNAME,  password:
        System.out.println( x: "CONEXION EXITOSA");
    } catch (Exception e){
        System.out.println("Error de conexion"+e);
    }
    return con;
}
```

- Se define un método estático `getConnection` que devuelve una conexión a la base de datos MySQL. Se utiliza el controlador JDBC de MySQL.

6. Método para Altas de Alumnos:

```
public void altasAlumnosManual(String matricula, String nombre, String carrera){
    Connection con = null;

    if(matricula.isEmpty() || nombre.isEmpty()){
        JOptionPane.showMessageDialog( parentComponent: null,  message: "FAVOR DE LLENAR L
        return; //EN CASO DE QUE NO INGRESE DATOS SE TERMINAN LAS ACCIONES
    }

    try{

        con = getConnection();

        //SE EVITA LA INSERCIÓN SQL
        ps = con.prepareStatement( sql: "INSERT INTO alumnos (matricula, nombre, car
        ps.setString( parameterIndex: 1,  x: matricula);
        ps.setString( parameterIndex: 2,  x: nombre);
        ps.setString( parameterIndex: 3,  x: carrera);

      int res =  ps.executeUpdate();
```

```
        if(res > 0){
            JOptionPane.showMessageDialog( parentComponent: null,  message: "USUARIO REGISTRADO CON EXITO");
            inicioAlumnos ini = new inicioAlumnos();
            ini.setVisible( b: true);


        }
        else{
            JOptionPane.showMessageDialog( parentComponent: null,  message: "ERROR AL GUARDAR A LA PERSONA");
        }

        con.close();

    } catch(Exception e){
        System.err.println( x: e);
    }
}
```

- Se define el método `altasAlumnosManual` que realiza la inserción de un nuevo alumno en la base de datos.
- Se verifica que la matrícula y el nombre no estén vacíos, mostrando un mensaje si es así.
- Se obtiene la conexión a la base de datos utilizando el método `getConnection`.
- Se prepara y ejecuta la consulta SQL para insertar un nuevo alumno.
- Se muestra un mensaje de éxito o error en función del resultado de la inserción.
- Se cierra la conexión a la base de datos.

Validación:

Este código se enfoca en la validación de credenciales para permitir el acceso al área de administradores en una aplicación Java con base de datos MySQL.

1. Constantes de Conexión a la Base de Datos y Variables SQL:

```
 */
public class Validacion {
    //Conexion con DB
    public static final String URL = "jdbc:mysql://localhost:3306/bibliotech";
    public static final String USERNAME = "root";
    public static final String PASSWORD = "";

    PreparedStatement ps;
    ResultSet rs;
```

- Se definen las constantes para la URL de la base de datos, el nombre de usuario y la contraseña. Además, se declaran variables para preparar consultas SQL y manejar resultados.

2. Método Estático de Conexión a la Base de Datos (`getConnection`):

```
public static Connection getConnection(){
    Connection con = null;

    try{
        Class.forName( className:"com.mysql.cj.jdbc.Driver");
        con = (Connection)DriverManager.getConnection( url:URL,  user:USERNAME,  passwo
        System.out.println( x:"CONEXION EXITOSA");
    } catch (Exception e){
        System.out.println("Error de conexion"+e);
    }
    return con;
}//Fin de la conexion
```

- Se define un método estático `getConnection` que devuelve una conexión a la base de datos MySQL. Utiliza el controlador JDBC de MySQL.

3. Método de Validación de Usuario y Contraseña (`validacion`):

```
public void validacion(String user, String pass){
     Connection con = null;
    //En caso de que no se ingrese ningun dato
    if(user.equals( anObject:"") || pass.equals( anObject:"")){
        JOptionPane.showMessageDialog( parentComponent:null,  message:"CAMPOS FALTAN
    }
    else{
        try{
           con = getConnection();
           ps = con.prepareStatement("SELECT usuario, pass FROM administradore
           rs =  ps.executeQuery();
            if(rs.next()){
                //en caso de ser correcta y validar desplegar la pantalla de a
                vistaAdmin admin = new vistaAdmin();
                admin.setVisible( b:true);
            }
            //En caso de que el usuario o contraseña no sean validos
            else{
```

```
                return;
            }

        } catch(Exception e){

            System.err.println( x:e);
        }
    }
}
```

- Se define el método `validación` que verifica la validez de las credenciales de usuario.
- Verifica que el usuario y la contraseña no estén vacíos, mostrando un mensaje si es así.

- Obtiene la conexión a la base de datos utilizando el método `getConnection`.
- Prepara y ejecuta una consulta SQL para verificar las credenciales en la tabla de administradores.
- Si las credenciales son correctas, se muestra la pantalla de administrador. En caso contrario, se muestra un mensaje de error.

Exporter:

Esta clase `Exporter` permite exportar datos de una o más tablas Swing a un archivo de hoja de cálculo Excel mediante la utilización de la biblioteca JExcelApi.

1. Importaciones:

```
package panel;

import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.List;
import javax.swing.JTable;
import jxl.Workbook;
import jxl.write.Label;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import jxl.write.WriteException;
```

- Se importan las clases necesarias para la manipulación de archivos, la representación de tablas (JTable) y la creación de hojas de cálculo Excel (jxl).

2. Clase Exporter: y atributos de la clase:

```
public class Exporter {

    private File file;
    private List tabla;
    private List nom_files;
```

- Se define la clase `Exporter` que se encargará de exportar tablas a un archivo Excelm la clase tiene tres atributos: `file` (archivo de destino), `tabla` (lista de tablas a exportar) y `nom_files` (lista de nombres para las hojas de Excel).

3. Constructor:

```
public Exporter(File file, List tabla, List nom_files) throws Exception {

this.file = file;
this.tabla = tabla;
this.nom_files = nom_files;
if(nom_files.size()!=tabla.size()){
    throw new Exception( message: "Error");
}
```

- Se define un constructor que recibe un archivo, una lista de tablas y una lista de nombres.
- Si el tamaño de las listas `nom_files` y `tabla` no coincide, se lanza una excepción.

4. Método de Exportación:y logica de exportación:

```
public boolean export() throws WriteException {
try {
    DataOutputStream out = new DataOutputStream(new FileOutputStream(file));
WritableWorkbook w = Workbook.createWorkbook(out);
for (int index = 0; index < tabla.size(); index++) {
  JTable table = (JTable) tabla.get(index);
  WritableSheet s = w.createSheet((String) nom_files.get(index), 0);
  for (int i = 0; i < table.getColumnCount(); i++) {
    String NomCol = table.getColumnName( column: i);
    s.addCell(new Label(i, 0,NomCol));

    for (int j = 0; j < table.getRowCount(); j++) {
        Object object = table.getValueAt( row: j,  column: i );
        s.addCell(new Label(i, j + 1, String.valueOf( obj: object))); //lo pone en la
    }
  }
}
w.write();
w.close();
out.close();
```

- Captura y manejo de excepciones en caso de errores durante la exportación.

Splash:

Este código Java representa una pantalla de bienvenida (Splash Screen) con una imagen y un mensaje de carga. La ventana se cierra automáticamente después de un breve período, dando paso a la ventana principal de la aplicación.

1. Importaciones:

```
package panel;
import com.sun.awt.AWTUtilities;
import java.awt.Image;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
```

- Se importan las clases necesarias, incluyendo algunas relacionadas con la interfaz gráfica y la manipulación de imágenes.

2. Definición de la Clase y atributos de la clase:

```
public static void main(String args[]) {
      / Variables declaration - do not modify
      private javax.swing.JLabel img_Splash;
      private javax.swing.JLabel jLabel1;
      private javax.swing.JLabel jLabel2;
      private javax.swing.JPanel jPanel1;
    // End of variables declaration
    /* Set the Nimbus look and feel */
     Look and feel setting code (optional)
```

- Se define la clase `Splash` que extiende `JFrame` e implementa la interfaz `Runnable`. La implementación de `Runnable` sugiere que esta clase podría ejecutarse en un hilo separado.y Se declaran los atributos y componentes de la interfaz gráfica.

3. Constructor:

```
public Splash() {
    initComponents();
    //Colocacion y ajustacion de las imagenes en los jlabels (Se ajustan al tam
   //SetImageLabel(img_Splash,"src\\img\\Splash.jpg");
   // SetImageLabel(img_cargando,"src\\img\\cargando.gif");

    this.setLocationRelativeTo( c:null);
    AWTUtilities.setWindowOpaque(this, false);
    tiempo = new Thread( task:this);
    tiempo.start();
}
```

El constructor inicializa componentes, configura la ubicación de la ventana, establece la transparencia y arranca un hilo (`Thread`) para la animación o temporizador.

4. Método `SetImageLabel`:

```
public void SetImageLabel(JLabel labelName, String root){
    ImageIcon image = new ImageIcon( filename: root);
    Icon icon = new ImageIcon( image: image.getImage().getScaledInstance( width: label
    labelName.setIcon(icon);
    this.repaint();
}


// Variables declaration - do not modify
private javax.swing.JLabel img_Splash;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JPanel jPanel1;
// End of variables declaration
```

- Este método ajusta la imagen al tamaño del `JLabel`.

7. Método `run`:

```
public void run() {
    if
            (tiempo != null){
        try {
            Thread.sleep( millis: 5000);
            this.dispose();
            new Principal().setVisible( b: true);
        } catch (InterruptedException ex) {
            Logger.getLogger( name: Splash.class.getName()).log( level: Level.SEVERE
        }
    }
    tiempo = null;
}
```

- Este método implementa la interfaz `Runnable`. Espera durante 5 segundos y luego cierra la ventana de bienvenida, mostrando la ventana principal (`Principal`).

Exportar Excel:

El código permite a los usuarios exportar datos de una tabla a un archivo de Excel, proporcionando una interfaz gráfica para seleccionar la ubicación del archivo y abriendo automáticamente el archivo Excel después de la exportación.

1. Importaciones:

```
package panel;

import java.awt.Desktop;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import javax.swing.JFileChooser;
import javax.swing.JTable;
import javax.swing.filechooser.FileNameExtensionFilter;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
```

- Se importan las clases necesarias para la manipulación de archivos, la interfaz gráfica (`JFileChooser`), y las clases de Apache POI, que son utilizadas para trabajar con archivos de Excel.

2. Método `exportarExcel`:

```
public void exportarExcel(JTable t) throws IOException {
    JFileChooser chooser = new JFileChooser();
    FileNameExtensionFilter filter = new FileNameExtensionFilter( description: "Archivos
    chooser.setFileFilter(filter);
    chooser.setDialogTitle( dialogTitle: "Guardar archivo");
    chooser.setAcceptAllFileFilterUsed( b: false);
    if (chooser.showSaveDialog( parent: null) == JFileChooser.APPROVE_OPTION) {
        String ruta = chooser.getSelectedFile().toString().concat( str: ".xlsx");
        try {
            File archivoXLS = new File( pathname: ruta);
            if (archivoXLS.exists()) {
                archivoXLS.delete();
            }
            archivoXLS.createNewFile();
            //Workbook libro = new XSSFWorkbook();
            Workbook libro = new XSSFWorkbook();
            FileOutputStream archivo = new FileOutputStream( file: archivoXLS);
            Sheet hoja = libro.createSheet("Mi hoja de trabajo 1");
```

- Este método realiza los siguientes pasos:
  - Configura un cuadro de diálogo para que el usuario elija la ubicación donde se guardará el archivo Excel.
  - Crea un archivo Excel (`.xlsx`).
  - Crea un libro de trabajo Excel y una hoja de trabajo.
  - Crea una fila para los encabezados de la tabla.
  - Llena el archivo Excel con los datos de la tabla (`JTable`).
  - Guarda y cierra el archivo Excel.
  - Abre el archivo Excel recién creado utilizando la aplicación predeterminada asociada con archivos Excel en el sistema.

- En caso de excepciones (por ejemplo, `IOException` o `NumberFormatException`), imprime un mensaje y relanza la excepción.

Modulo nbproject:

- project.xml: Este archivo contiene la configuración del proyecto NetBeans, incluyendo información sobre las dependencias y configuraciones específicas del proyecto.

- build-impl.xml: Archivo XML que contiene scripts Ant generados automáticamente por NetBeans para compilar y construir el proyecto.

- genfiles.properties: Este archivo mantiene las rutas a archivos generados durante el proceso de construcción del proyecto.

- project.properties: Propiedades específicas del proyecto, como configuraciones de compilación y opciones del entorno de ejecución.

Prívate:

- private.properties: Propiedades privadas del proyecto, utilizadas para almacenar información sensible o configuraciones específicas de un desarrollador.

- private.xml: Configuraciones privadas, que pueden incluir ajustes específicos de NetBeans.

Modulo dist:

- Archivos JAR: Bibliotecas externas necesarias para el funcionamiento de la aplicación. Estas bibliotecas son empaquetadas con la aplicación para facilitar la distribución.

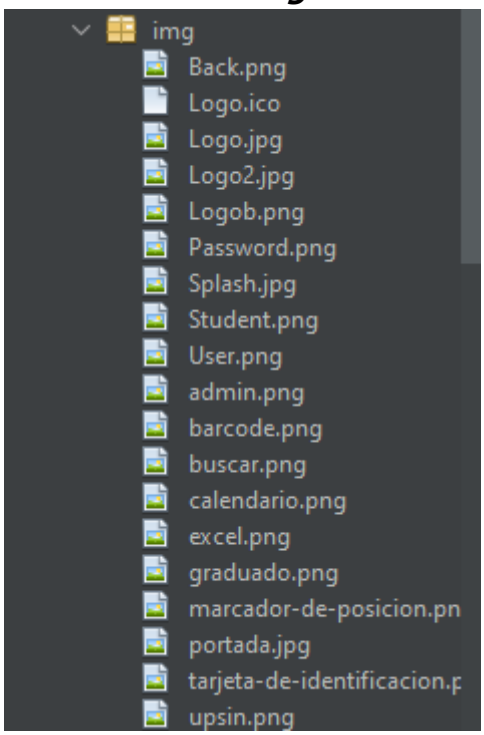
- Bibliotech.jar: Este archivo JAR es el ejecutable principal de la aplicación y contiene toda la funcionalidad necesaria.

Modulo build:

- PanelRound.class: Este archivo es la representación compilada de la clase`PanelRound`. Contiene el código ejecutable para esa clase específica.

- img:

- Lista de imágenes:



Estos archivos contienen las imágenes utilizadas por la aplicación. Pueden incluir iconos, fondos y otros recursos visuales.

- Lista de paneles: Estos archivos son las representaciones compiladas de las clases en el paquete `panel`. Cada clase puede representar un componente visual o lógica específica de la interfaz de usuario.

Dependencias y Bibliotecas:

En el desarrollo de la aplicación "Bibliotech" se hace uso de diversas dependencias y bibliotecas para facilitar la implementación de funcionalidades específicas. A continuación, se detallan algunas de las principales dependencias incluidas en el proyecto:

1. absoluteLayout.jar:

Esta biblioteca proporciona capacidades de diseño de interfaz gráfica sin la necesidad de un gestor de diseño. Permite colocar componentes en posiciones absolutas dentro de un contenedor.

2. jcalendar-1.4.jar:

La biblioteca JCalendar ofrece componentes de interfaz gráfica de usuario (GUI) que facilitan la selección de fechas en aplicaciones Java. Esto es útil para la gestión de eventos temporales en la aplicación "Bibliotech".

3. jd-2.8.jar:

JD-Core es una biblioteca que proporciona funciones de descompilación para archivos de clase Java. Aunque no se utiliza directamente en tiempo de ejecución, puede ser útil durante el desarrollo y la depuración para comprender el código de bibliotecas externas.

4. mysql-connector-java-8.x.x.jar:

Este conector JDBC permite la comunicación entre la aplicación Java y la base de datos MySQL. Facilita la ejecución de consultas y la gestión de la interacción con la base de datos "Bibliotech".

5. poi-5.2.3.jar:

Apache POI es una biblioteca que proporciona clases para manipular documentos de Microsoft Office, como hojas de cálculo Excel. En "Bibliotech", se utiliza para exportar datos a archivos Excel.

6. rsScaleLabel.jar:
Esta biblioteca ofrece un componente de etiqueta escalable que puede ser útil para el diseño y presentación de información en la interfaz gráfica de usuario.

7. pol-5-2-3.jar:
Biblioteca que podría estar relacionada con aspectos gráficos o de presentación en la interfaz de usuario, aunque se necesita más información específica sobre su implementación en "Bibliotech".

Estas dependencias y bibliotecas externas se integran en el proyecto para enriquecer y ampliar la funcionalidad de "Bibliotech", facilitando tareas como la manipulación de fechas, la exportación de datos y la interacción con una base de datos MySQL.

Configuración del Entorno de Desarrollo

La correcta configuración del entorno de desarrollo es esencial para garantizar la eficiencia y el funcionamiento adecuado del proyecto "Bibliotech". A continuación, se describen los elementos clave en la configuración del entorno:

1. Proyecto NetBeans:
"Bibliotech" está desarrollado en el entorno de desarrollo NetBeans. NetBeans proporciona un conjunto de herramientas robustas para el desarrollo de aplicaciones Java, facilitando la creación, depuración y mantenimiento del código fuente.

2. Estructura del Proyecto:
La estructura del proyecto sigue las convenciones estándar de NetBeans. La carpeta `nbproject` contiene archivos relacionados con la configuración del proyecto, como `project.xml`, `build-impl.xml`, `genfiles.properties`, y `project.properties`.

3. Gestión de Dependencias:
Las dependencias externas del proyecto, como bibliotecas y JARs, se gestionan en la carpeta `dist/lib`. Estos archivos son esenciales para la ejecución de la aplicación y se incluyen en el proyecto para garantizar la portabilidad.

4. Exportación de Datos y Bibliotecas Externas:
Se utiliza la carpeta `dist` para almacenar archivos generados durante el proceso de construcción, como el archivo ejecutable JAR (`Bibliotech.jar`). Además, se encuentran bibliotecas externas necesarias para el funcionamiento de la aplicación.

5. Ejecución de la Aplicación:
Para ejecutar "Bibliotech", se recomienda utilizar el archivo JAR generado. La ejecución se realiza mediante el comando `java -jar Bibliotech.jar`. Además, la interfaz gráfica principal se inicia desde la clase `Splash` como pantalla de bienvenida.

6. Configuración de Librerías:
Descripción:Las librerías externas utilizadas en el proyecto deben estar configuradas adecuadamente en NetBeans. Esto garantiza que las clases y métodos de estas bibliotecas estén disponibles para su uso en el código.

7. Exportación a Excel:
La funcionalidad de exportar datos a Excel se realiza utilizando la biblioteca Apache POI. La configuración adecuada de esta biblioteca en el proyecto asegura la correcta generación de archivos Excel.

8. Configuración de Imágenes y Recursos:
La carpeta `build/classes/img` contiene recursos visuales utilizados por la interfaz de usuario. Se debe prestar atención a la correcta configuración de estos recursos para garantizar su disponibilidad durante la ejecución.

9. Configuración de Base de Datos:
- Descripción: La aplicación se conecta a una base de datos MySQL, y la configuración de la conexión se realiza en las clases relevantes. Asegúrese de que los parámetros de conexión, como URL, usuario y contraseña, estén configurados correctamente.

La configuración del entorno del proyecto "Bibliotech" es crucial para facilitar el desarrollo, la ejecución y el mantenimiento del software. Siguiendo estas pautas, los desarrolladores pueden asegurarse de que el entorno esté correctamente establecido para una experiencia de desarrollo eficiente.

Arquitectura del Sistema:

La arquitectura del sistema "Bibliotech" está diseñada para proporcionar una estructura sólida y modular que facilita la comprensión, el mantenimiento y la escalabilidad del software. A continuación, se describen los aspectos clave de la arquitectura del sistema:

1. Modelo-Vista-Controlador (MVC):

- Descripción: La arquitectura sigue el patrón MVC, que divide la aplicación en tres componentes principales: el Modelo (que gestiona los datos y la lógica de negocio), la Vista (que maneja la interfaz de usuario) y el Controlador (que actúa como intermediario entre el Modelo y la Vista). Este enfoque mejora el modularidad y la mantenibilidad.

2. Interfaz Gráfica de Usuario (GUI):

- Descripción: La interfaz de usuario se implementa utilizando componentes de la biblioteca Swing de Java. La estructura de la GUI está compuesta por múltiples paneles, cada uno diseñado para una función específica. El uso de paneles ayuda a organizar y presentar de manera eficiente la información al usuario.

3. Clases Principales:

- Descripción: La clase `Splash` sirve como pantalla de bienvenida al iniciar la aplicación, mientras que la clase `Principal` representa la ventana principal de la aplicación. Se utilizan otros paneles, como `PanelRound` para elementos visuales con esquinas redondeadas, proporcionando una estética atractiva.

4. Gestión de Usuarios:

- Descripción: La aplicación maneja la gestión de usuarios, diferenciando entre administradores y alumnos. La autenticación se realiza a través de la clase `Validacion`, que verifica las credenciales ingresadas por los administradores al iniciar sesión.

5. Conexión a Base de Datos:

- Descripción: La aplicación se conecta a una base de datos MySQL para almacenar y recuperar datos. La clase `AltasAlumnosManual` se encarga de gestionar las altas de alumnos en la base de datos, mientras que la clase `Validacion` valida las credenciales de los administradores.

6. Exportación a Excel:

- Descripción: La funcionalidad de exportar datos a Excel se logra mediante la clase `Exporter`, que utiliza la biblioteca JExcelApi. Esto permite a los usuarios generar archivos Excel a partir de los datos mostrados en las tablas de la aplicación.

7. Configuración del Proyecto:

- Descripción: La estructura del proyecto se organiza en paquetes lógicos, como `panel`, para mantener la claridad y la organización. La carpeta `img` almacenas imágenes utilizadas en la interfaz, y la carpeta `dist` contiene archivos generados durante la construcción.

8. Dependencias Externas:

- Descripción: El proyecto utiliza varias bibliotecas externas, como Apache POI para la exportación a Excel y JExcelApi para la manipulación de hojas de cálculo. La correcta gestión de estas dependencias es fundamental para el funcionamiento adecuado de la aplicación.

La arquitectura del sistema "Bibliotech" se centra en la modularidad, la eficiencia y la facilidad de mantenimiento. Este diseño permite a los desarrolladores comprender y ampliar el sistema de manera efectiva, mientras que los usuarios experimentan una interfaz intuitiva y funcionalidades robustas.

Modelo de Datos:

El modelo de datos de "Bibliotech" define la estructura y las relaciones entre las entidades que representan la información almacenada en la base de datos. A continuación, se detallan las principales entidades y sus relaciones:

1. Tabla de Alumnos:

- Descripción: Esta tabla almacena la información relativa a los alumnos registrados en el sistema.
  - `Matrícula`: Identificador único para cada alumno.
  - `Nombre`: Nombre del alumno.
  - `Carrera`: Carrera académica del alumno.

2. Tabla de Administradores:

- Descripción: Contiene los datos de los administradores encargados de gestionar la aplicación.
- Atributos:
- `Usuario`: Nombre de usuario del administrador.
- `Contraseña`: Contraseña asociada al usuario administrador.

3. Relación entre Alumnos y Administradores:

- Descripción: La relación establece que múltiples alumnos pueden estar registrados en el sistema, mientras que hay un conjunto limitado de administradores encargados de gestionar la aplicación.
- Tipo de Relación: Uno a muchos (1:N).

4. Exportación a Excel:

- Descripción: La exportación a Excel implica la generación de archivos Excel que contienen datos específicos de las tablas mostradas en la interfaz.
- Estructura del Archivo Excel:
- Una hoja de trabajo por cada tabla exportada.
- En cada hoja, las columnas representan los nombres de las columnas de la tabla.
- Los datos se llenan debajo de las columnas correspondientes.

El modelo de datos refleja la organización lógica de la información dentro del sistema "Bibliotech". Está diseñado para admitir operaciones como el registro de alumnos, la autenticación de administradores y la exportación de datos a Excel. Mantener la coherencia y la integridad de estos datos es esencial para garantizar el correcto funcionamiento de la aplicación.

Flujo de Trabajo

El "Flujo de Trabajo" en el sistema "Bibliotech" describe las secuencias de acciones y procesos que los usuarios pueden llevar a cabo durante su interacción con la aplicación. A continuación, se presenta un desglose del flujo de trabajo para las principales funcionalidades del sistema:

1. Registro de Alumnos:

- El usuario accede a la interfaz de registro de alumnos.
- Ingresa la matrícula, nombre y carrera del alumno.
- Presiona el botón de registro.
- El sistema valida la información y almacena los datos en la base de datos.
- Se muestra un mensaje de confirmación.

2. Inicio de Sesión de Administradores:

- El administrador accede a la pantalla de inicio de sesión.
- Ingresa su nombre de usuario y contraseña.
- Presiona el botón de inicio de sesión.
- El sistema valida las credenciales y redirige al administrador a la interfaz de administración.

3. Administración de Alumnos:

- El administrador visualiza la lista de alumnos registrados.
- Puede realizar operaciones como editar, eliminar o buscar alumnos.
- Al realizar cambios, el sistema actualiza la base de datos y refleja los resultados en la interfaz.

4. Exportación a Excel:

- Desde la interfaz de administración, el administrador selecciona la opción de exportar a Excel.
- El sistema genera archivos Excel con la información de las tablas relevantes.
- Se proporciona un enlace para descargar los archivos Excel generados.

5. Inicio de Sesión de Alumnos:

- El alumno accede a la pantalla de inicio de sesión.
- Ingresa su matrícula y otros datos de autenticación.
- Después de la validación, se redirige al alumno a la interfaz personalizada.

6. Visualización de Datos por Alumnos:

- Los alumnos pueden ver información específica relacionada con su matrícula y registro.
- La interfaz muestra datos como el nombre y la carrera del alumno.

El flujo de trabajo en "Bibliotech" se ha diseñado para ser intuitivo y eficiente, proporcionando a los usuarios la capacidad de realizar acciones específicas según su rol en el sistema. Cada paso en el flujo de trabajo está cuidadosamente estructurado para garantizar una experiencia coherente y sin problemas.

Codificación y Estándares

La sección de "Codificación y Estándares" en el contexto del proyecto "Bibliotech" abarca las prácticas y pautas seguidas durante el desarrollo del software para garantizar consistencia, legibilidad y mantenibilidad del código. A continuación, se destacan algunos de los aspectos clave relacionados con la codificación y los estándares adoptados:

1. Convenios de Nomenclatura:

Se sigue un conjunto consistente de convenciones de nomenclatura para variables, métodos y clases. Esto mejora la comprensión del código y facilita la colaboración entre desarrolladores.

2. Comentarios y Documentación:

El código está documentado de manera exhaustiva utilizando comentarios claros y descriptivos. La documentación se adhiere a estándares reconocidos, proporcionando información sobre la funcionalidad y el propósito de las clases y métodos.

3. Manejo de Excepciones:
Se implementa un manejo adecuado de excepciones para mejorar la robustez del código. Las excepciones se gestionan de manera coherente en todo el sistema, garantizando un comportamiento predecible y facilitando la depuración.

4. Organización de Paquetes:
Los archivos y paquetes se organizan de manera lógica y estructurada. Se siguen prácticas recomendadas para la organización de archivos, lo que facilita la navegación y la comprensión de la estructura del proyecto.

5. Seguridad:
Se han aplicado prácticas de seguridad para proteger contra vulnerabilidades conocidas. Esto incluye la validación de datos de entrada, el uso de consultas parametrizadas en bases de datos y otras medidas para prevenir ataques comunes.

6. Revisión de Código:
El equipo de desarrollo lleva a cabo revisiones regulares de código para garantizar la calidad y adherencia a los estándares establecidos. Se fomenta la colaboración y se realizan correcciones según sea necesario.

7. Compatibilidad y Portabilidad:
El código se desarrolla teniendo en cuenta la compatibilidad y portabilidad entre diferentes entornos y versiones de bibliotecas. Esto garantiza que la aplicación pueda ejecutarse de manera eficiente en diversos sistemas.

Estos estándares y prácticas de codificación contribuyen a un código base sólido, fácil de mantener y escalable a medida que evoluciona el proyecto "Bibliotech". La adhesión a estos estándares promueve la eficiencia del equipo de desarrollo y contribuye a la calidad general del software.

Pruebas:
Se recomienda hacer pruebas constantes de registro de usuarios y alumnos nuevos para ver que con el tiempo no se deteriore el programa y se revise que siga funcionando de la misma forma que desde un inicio.

Tareas, prácticas Y metas Sugeridas para futuras actualizaciones:
Recomendamos que realicen tareas constantes de actualización de base de datos y conexiones con código, también fuera un gran avance que investigaran como implementar una adaptación a distintos dispositivos y no solo dependa de una sola máquina, también practiquen la seguridad de base de datos y no se arriesguen a robo de información en algún punto.

Soporte y Contacto:
En caso de requerir ayuda o tener alguna duda y no puedas resolverlas puedes acudir a los creadores originales del código y solicitar apoyo.

Contactos:

- Jorge Rendon
  6692495240
  2021030600@upsin.edu.mx

- Manuel Ramírez
  6691220185
  2021030209@upsin.edu.mx

- Bernardo Rosas
  6691270200
  2021030109@upsin.edu.mx