Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Aragón

Carrera: ICO

Nombre: Luis Alejandro Romero Martínez Grupo: 1159 Fecha: 5/06/23

Materia: programación Orientada a Objetos Semestre: II

Profesor: Jesús Hernández Cabrera.

Documentación del proyecto final

Ventana Princesas

Como primer punto empezaremos revisando la vista o la parte de la interfaz del programa el cual hereda de JFrame

```
public class VentanaPrincesas extends JFrame {
    4 usages
    private GridLayout layoutGeneral;

    //Objetos de panel 1
    4 usages
    private JLabel lblId;
    4 usages
    private JLabel lblNombre;
    4 usages
    private JLabel lblCuento;
    4 usages
    private JLabel lblColorVestido;
    4 usages
    private JLabel lblImagen;
    4 usages
    private JLabel lblAntagonista;
    6 usages
    private JTextField txtId;
    5 usages
    private JTextField txtNombre;
    5 usages
    private JTextField txtCuento;
    5 usages
    private JTextField txtColorVestido;
    5 usages
    private JTextField txtImagenUrl;
    5 usages
    private JTextField txtAntagonista;
    4 usages
    private JButton btnAgregar;
    18 usages
    private JPanel panel1;
```

En la primera imagen se aprecia las variables empleadas en como lo dice la primera línea comentada dentro del panel 1. El cual es el que requiere de mayor cantidad de objetos pues como se observa en la siguiente imagen los siguientes paneles van con muchísimos menos objetos

```
//Objetos Panel 2

4 usages
private JTable tblPrincesas;

8 usages
private JPanel panel2;
4 usages
private JButton btnCargar;
4 usages
private JScrollPane scroll;
//Objetos Panel 3

8 usages
private JPanel panel3;
4 usages
private JLabel lblfila;
4 usages
private JLabel lblImg;
//Objetos Panel 4
8 usages
private JPanel panel4;
4 usages
private JButton btnBorrar;
4 usages
private JButton btnActualizar;
```

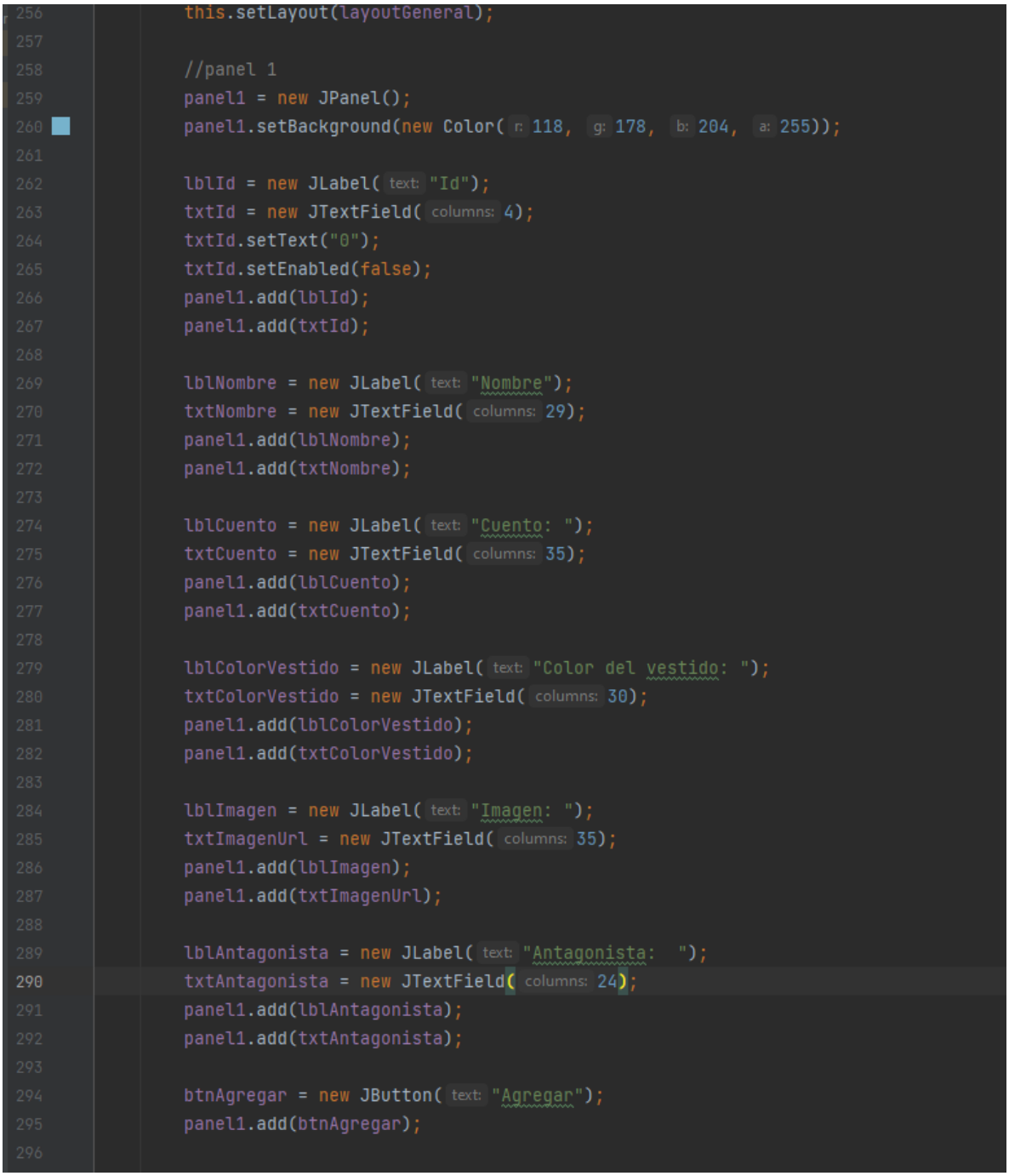
Después de la declaración de nuestros objetos que usaremos les declaramos los getters y setters desde la línea 44 a la 247 los cuales estas marcados por comentarios

Y para comenzar a acomodar todo creamos el constructor el cual nos pedirá el titulo de la ventana y al igual que en la declaración de los objetos dividimos por partes el programa para tener un mayor orden.

```
//General
layoutGeneral = new GridLayout( rows: 2,  cols: 2);
this.setLayout(layoutGeneral);
```

En la anterior imagen se puede ver como primero se declara el Grid layout donde crearemos la cuadricula para poner cada panel y se la agregamos a la ventana

Como siguiente paso procedemos a crear el panel 1 darle formato y crear todos los objetos que serán usados dentro de este para al final ser agregados al panel.



```
256         this.setLayout(layoutGeneral);
257
258         //panel 1
259         panel1 = new JPanel();
260         panel1.setBackground(new Color( r: 118,  g: 178,  b: 204,  a: 255));
261
262         lblId = new JLabel( text: "Id");
263         txtId = new JTextField( columns: 4);
264         txtId.setText("0");
265         txtId.setEnabled(false);
266         panel1.add(lblId);
267         panel1.add(txtId);
268
269         lblNombre = new JLabel( text: "Nombre");
270         txtNombre = new JTextField( columns: 29);
271         panel1.add(lblNombre);
272         panel1.add(txtNombre);
273
274         lblCuento = new JLabel( text: "Cuento: ");
275         txtCuento = new JTextField( columns: 35);
276         panel1.add(lblCuento);
277         panel1.add(txtCuento);
278
279         lblColorVestido = new JLabel( text: "Color del vestido: ");
280         txtColorVestido = new JTextField( columns: 30);
281         panel1.add(lblColorVestido);
282         panel1.add(txtColorVestido);
283
284         lblImagen = new JLabel( text: "Imagen: ");
285         txtImagenUrl = new JTextField( columns: 35);
286         panel1.add(lblImagen);
287         panel1.add(txtImagenUrl);
288
289         lblAntagonista = new JLabel( text: "Antagonista:  ");
290         txtAntagonista = new JTextField( columns: 24);
291         panel1.add(lblAntagonista);
292         panel1.add(txtAntagonista);
293
294         btnAgregar = new JButton( text: "Agregar");
295         panel1.add(btnAgregar);
296
```

Se repite el paso para los paneles 2,3,4, se crean los paneles se les da formato, se crean los objetos y son agregados a su panel correspondiente los cuales también se encuentran separados por comentarios

```
297
298         //panel 2
299         panel2 = new JPanel();
300         panel2.setBackground(new Color( r: 234,  g: 108,  b: 108));
301         panel2.setLayout(new FlowLayout());
302         btnCargar = new JButton( text: "Cargar");
303         panel2.add(btnCargar);
304         tblPrincesas = new JTable();
305         scroll = new JScrollPane(tblPrincesas);
306         panel2.add(scroll);
307
308
309         //panel 3
310         panel3 = new JPanel();
311         panel3.setBackground(new Color( r: 142,  g: 204,  b: 105,  a: 255));
312         panel3.setLayout(new FlowLayout());
313         lblfila = new JLabel( text: "Foto:");
314         panel3.add(lblfila);
315         lblImg = new JLabel( text: "...");
316         panel3.add(lblImg);
317
318         //panel 4
319         panel4 = new JPanel();
320         panel4.setBackground(new Color( r: 137,  g: 211,  b: 168,  a: 255));
321         panel4.setLayout(new FlowLayout());
322         btnActualizar = new JButton( text: "Actualizar");
323         btnBorrar = new JButton( text: "Borrar");
324         panel4.add(btnActualizar);
325         panel4.add(btnBorrar);
326
327
```

Para finalizar la parte de la vista solo se le da formato a la ventana y se agregan todos los paneles a la ventana.

```
        //Parte final

        this.getContentPane().add(panel1,  index: 0);
        this.getContentPane().add(panel2,  index: 1);
        this.getContentPane().add(panel3,  index: 2);
        this.getContentPane().add(panel4,  index: 3);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setSize( width: 1000,  height: 900);
        this.setVisible(true);
    }
```

A lo largo del programa se requirio la creación de un metodo que vaciara los textfields el cual se declara sobre esta clase para ser más practico y simple de realizar nombrado limpiar.

```
339     public void limpiar(){
340         txtNombre.setText("");
341         txtCuento.setText("");
342         txtColorVestido.setText("");
343         txtImagenUrl.setText("");
344         txtAntagonista.setText("");
345     }
346
347 }
```

Princesas Disney

En esta clase lo único que tenemos es lo que será la base de nuestra tabla por así decirlo pues son los atributos de las princesas de Disney por así decirlo generales

```java
public class PrincesasDisney {
    4 usages
    private int id;
    5 usages
    private String nombre;
    5 usages
    private String cuento;
    5 usages
    private String colorVestido;
    6 usages
    private String imagenUrl;
    5 usages
    private String antagonista;

    2 usages
    public PrincesasDisney() {
    }
```

Lo único que se realizo en esta clase son sus getters y setters así como su constructor por defecto y el sobre cargado, además de usar uno donde no se requiera el id para que este no se vea afectado en ciertos procesos.

```java
    2 usages
    public PrincesasDisney() {
    }

    no usages
    public PrincesasDisney(String nombre, String cuento, String colorVestido, String imagenUrl, String antagonista) {
        this.nombre = nombre;
        this.cuento = cuento;
        this.colorVestido = colorVestido;
        this.imagenUrl = imagenUrl;
        this.antagonista = antagonista;
    }

    2 usages
    public PrincesasDisney(int id, String nombre, String cuento, String colorVestido, String imagenUrl, String antagonista) {
        this.id = id;
        this.nombre = nombre;
        this.cuento = cuento;
        this.colorVestido = colorVestido;
        this.imagenUrl = imagenUrl;
        this.antagonista = antagonista;
    }
```

Por ultimo dentro de esta clase tenemos un método el cual nos ayuda en el proceso del cual a través de la url de la tabla se regresara una imagen. Este se llamo getImagen.

```
1 usage
94    public ImageIcon getImagen() throws MalformedURLException {
95
96        URL urlImage = new URL(this.imagenUrl);
97        return new ImageIcon(urlImage);
98
99    }
```

Conexión Singleton

Para esta clase lo único que hice fue copiar el código para poder establecer la conexión con nuestra bases de datos además de agregar la implementación para ayudar a la conexión a ser realizada puesto que sin esta no corre. Aquí no se realizo ningún cambio.

Interfaz DAO

En el interfaz DAO no ningún cambio al ser implementado en la siguiente clase ese se quedo tal cual con los métodos abstractos.

```
6  public interface InterfazDAO {
       1 usage  1 implementation
7      public abstract boolean insertar(Object obj) throws SQLException;
8
9
       1 usage  1 implementation
10     boolean updte(Object obj, int Id) throws SQLException;
11
       no usages  1 implementation
12     public abstract boolean delete(int id) throws SQLException;
       2 usages  1 implementation
13     public abstract ArrayList obtenerTodo() throws SQLException;
       no usages  1 implementation
14     public abstract Object buscarPorId(String id) throws SQLException;
15 }
16
```

Princesas Disney DAO

Para esta clase implemente todos los métodos visto pero los principalmente usados fueron el de Insertar: que lo que pide es que es un objeto el cual se insertara como un nuevo registro a la base de datos.

```java
@Override
public boolean insertar(Object obj) throws SQLException {
    String sqlInsert = "INSERT INTO PrincesasDisney(Nombre,Cuento,ColorVestido,Imagen,Antagonista) VALUES(?,?,?,?,?)";
    int rowCount = 0;


    PreparedStatement pstm = ConexionSingleton.getInstance( baseDatos: "PrincesasDisney.db").getConnection().prepareStatement(sqlInsert);
    pstm.setString( parameterIndex: 1, ((PrincesasDisney) obj).getNombre());
    pstm.setString( parameterIndex: 2, ((PrincesasDisney) obj).getCuento());
    pstm.setString( parameterIndex: 3, ((PrincesasDisney) obj).getColorVestido());
    pstm.setString( parameterIndex: 4, ((PrincesasDisney) obj).getImagenUrl());
    pstm.setString( parameterIndex: 5, ((PrincesasDisney) obj).getAntagonista());
    rowCount = pstm.executeUpdate();
    return rowCount > 0;
}
```

El actualizar: El cual lo que hace es sobre escribir la información dentro de un registro solo requiere de un objeto y del id del registro que se remplazara.

```java
@Override
public boolean updte(Object obj,int Id) throws SQLException {
    String sqlUpdate = "UPDATE PrincesasDisney SET Nombre = ?, Cuento = ?, ColorVestido = ?, Imagen = ?, Antagonista = ? WHERE Id = "+Id+";";
    int rowCount = 0;


    PreparedStatement pstm = ConexionSingleton.getInstance( baseDatos: "PrincesasDisney.db").getConnection().prepareStatement(sqlUpdate);


    pstm.setString( parameterIndex: 1, ((PrincesasDisney) obj).getNombre());
    pstm.setString( parameterIndex: 2, ((PrincesasDisney) obj).getCuento());
    pstm.setString( parameterIndex: 3, ((PrincesasDisney) obj).getColorVestido());
    pstm.setString( parameterIndex: 4, ((PrincesasDisney) obj).getImagenUrl());
    pstm.setString( parameterIndex: 5, ((PrincesasDisney) obj).getAntagonista());


    rowCount = pstm.executeUpdate();

    return rowCount > 0;
}
```

Y el de obtener todo: El cual lo que hace es mandar a llamar a todos los registros de la base de datos e insertarlos dentro de un array list para después ser mostrados en la tabla

```java
@Override
public ArrayList obtenerTodo() throws SQLException {
    String sql = "SELECT * FROM PrincesasDisney";
    ArrayList<PrincesasDisney> resultado = new ArrayList<>();

    Statement stm = ConexionSingleton.getInstance( baseDatos: "PrincesasDisney.db").getConnection().createStatement();
    ResultSet rst = stm.executeQuery(sql);
    while (rst.next()){
        resultado.add(new PrincesasDisney(rst.getInt( columnIndex: 1),rst.getString( columnIndex: 2),rst.getString( columnIndex: 3),rst.getString( columnIndex: 4),
                rst.getString( columnIndex: 5),rst.getString( columnIndex: 6)));
    }
```

Pero igual tuve que agregar un método llamado “cambiarPorCeldas” este pide un String que se insertara en la tabla, además como el nombre del método lo dice se requiere la id de la fila y la columna para así poder identificar la celda exacta de cual se cambiara el dato, cabe aclarar que este método no permite que se modifiquen la columna de la “Id”.

Lo primero que hace este método es declarar la variable que de sqlUpdate donde se dicta la sentencia que dará la orden a la base de datos, así como la variable del prepared statement la cual reconocerá nuestra base de datos y se encargara de mandar la primera variable a la base de datos y también se declara la variable que detecta si hubo un cambio en las columnas el rowCount.

```
1 usage
public boolean cambiarPorCeldas(String datoActualizado, int Id,int colIndex) throws SQLException {
    String sqlUpdate;
    PreparedStatement pstm;
    int rowCount;
```

Después de esto ahora si empezamos creando un switch el cual dependiendo de la columna seleccionada este realizara pequeños cambios según el numero de columna que se le asigne.

```
switch (colIndex){

    case 1:
        sqlUpdate = "UPDATE PrincesasDisney SET Nombre = ? WHERE Id = "+Id+";";
        rowCount = 0;
        pstm = ConexionSingleton.getInstance( baseDatos: "PrincesasDisney.db").getConnection().prepareStatement(sqlUpdate);
        pstm.setString( parameterIndex: 1, (datoActualizado));
        rowCount = pstm.executeUpdate();

        return rowCount > 0;


    case 2:
        sqlUpdate = "UPDATE PrincesasDisney SET Cuento = ? WHERE Id = "+Id+";";
        rowCount = 0;
        pstm = ConexionSingleton.getInstance( baseDatos: "PrincesasDisney.db").getConnection().prepareStatement(sqlUpdate);
        pstm.setString( parameterIndex: 1, (datoActualizado));
        rowCount = pstm.executeUpdate();

        return rowCount > 0;
    case 3:
        sqlUpdate = "UPDATE PrincesasDisney SET ColorVestido = ? WHERE Id = "+Id+";";
        rowCount = 0;
        pstm = ConexionSingleton.getInstance( baseDatos: "PrincesasDisney.db").getConnection().prepareStatement(sqlUpdate);
        pstm.setString( parameterIndex: 1, (datoActualizado));
        rowCount = pstm.executeUpdate();

        return rowCount > 0;
    case 4:
        sqlUpdate = "UPDATE PrincesasDisney SET Imagen = ? WHERE Id = "+Id+";";
        rowCount = 0;
        pstm = ConexionSingleton.getInstance( baseDatos: "PrincesasDisney.db").getConnection().prepareStatement(sqlUpdate);
        pstm.setString( parameterIndex: 1, (datoActualizado));
        rowCount = pstm.executeUpdate();

        return rowCount > 0;
    case 5:
        sqlUpdate = "UPDATE PrincesasDisney SET Antagonista = ? WHERE Id = "+Id+";";
        rowCount = 0;
        pstm = ConexionSingleton.getInstance( baseDatos: "PrincesasDisney.db").getConnection().prepareStatement(sqlUpdate);
        pstm.setString( parameterIndex: 1, (datoActualizado));
        rowCount = pstm.executeUpdate();

        return rowCount > 0;


}

return false;
}
```

En cada método es casi lo mismo que el método de actualizar, la única diferencia es que dependiendo la columna se hace el cambio en el SET del sqlUpdate pues es la columna a modificar y más adelante lo único que se hace es pedir la id para identificar que de que fila se modificara esa columna dándonos la manera de así poder mover las celdas que queramos y lo único que hace es regresar si hubo un cambio en las filas y en caso de no hacerlo manda un false.

Modelo Tabla Princesa

Esta tabla se diseñó como lo hemos estado viendo a través del semestre, primero se implementó el table model para poder sobrescribir los métodos que nos permitirán darle forma a nuestra tabla. Se declara la vista y Princesas DAO para poder usar el contenido de estas de una manera más fácil y nuestra constante que serán nuestro número de columnas que usaremos.

También declaramos un constructor vacío que contenga el array de datos y nuestras princesas dao donde están los métodos y pues se crea la tabla con el diseño visto en clase

De la línea 29 a la 152 no hay gran cambio solo se cambian los nombres de las columnas y el tipo de dato que reciben estas.

Pero dentro de esta clase se crearon múltiples métodos que serán usados dentro del controlador el primer método que tenemos es el de “cargarDatos” el cual manda a llamar el método “obtenerTodo” (De la clase PrincesasDisneyDAO) pero antes lo meto dentro de un array list que contiene objetos de tipo Princesa Disney.

```
153    public void cargarDatos() {
154
155        try {
156            ArrayList<PrincesasDisney> tirar = pdao.obtenerTodo();
157
158            datos = pdao.obtenerTodo();
159        } catch (SQLException e) {
160            System.out.println(e.getMessage());
161        }
162    }
163
```

El siguiente método que tenemos es el de agregarPrincesas a la tabla y a la base de datos puesto que primero nos pide un objeto de tipo Princesa Disney. Para luego mandar a llamar desde la clase PrincesasDisneyDAO el método insertar al cual se le asigna objeto que se pidió anteriormente y este nos arroja un booleano para saber si se completo o no además de que este método se nos pide tratar la excepción por lo que lo metemos en un try / catch para evitar errores.

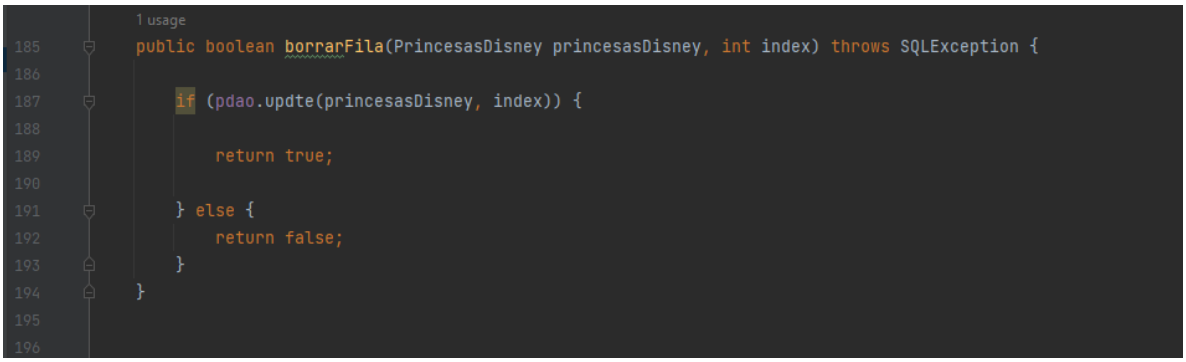
```
164    public boolean agregarPrincesa(PrincesasDisney princesa) {
165        boolean resultado = false;
166        try {
167            if (pdao.insertar(princesa)) {
168                datos.add(princesa);
169                resultado = true;
170            } else {
171
172                resultado = false;
173            }
174        } catch (SQLException e) {
175            System.out.println(e.getMessage());
176        }
177        return resultado;
178    }
```

El siguiente método es el de getPrincesasAtIndex el cual nos pide un índice de alguna columna y lo que hace este es regresar el dato que se encuentra en ese indice.

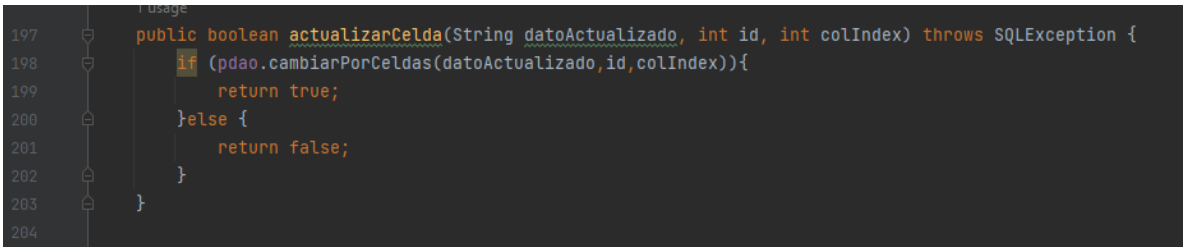
```
        3 usages
180    public PrincesasDisney getPrincesaAtIndex(int i) {
181
182        return datos.get(i);
183    }
184
```

Ya casi al final como penúltimo método tenemos el de borrarFila el cual nos pide un objeto de tipo PrincesasDisney y un id y lo único que hace es mandar a llamar el método update

y para que la maquina sepa que se actualizara se no requiere el id y el objeto que se insertara en ese registro o por el que sobrescribirá. También regresa un booleano para ver si funciono.



Como ultimo metodo tenemos el de actualizarCelda el cual nos pide 3 cosas, la primera es la cadena del dato que se actualizara, luego el id y el numero de la columna. Este metodo manda a llamar el metodo cambiarPorCeldas de la clase PrincesasDisneyDAO y se le otorgan estos 3 datos para que nuestro metodo. Asi como los demas metodos este regresa un booleano para saber si funciono.



Controlador Princesas

Por último tenemos la clase de controlador en la cual para iniciar se implementa el MouseAdapter puesto que solo usaremos ese método y los demás no, procedemos a declarar la vista y el modelo para poder completar nuestro modelo vista controlador y que se conecten. Después se crea un constructor que nos pide la vista y se le declaran todos los botones y objetos que implementaran este mouseListener así como el modelo de donde sacaremos los métodos.

```
3 usages
public class ControladorPprincesas extends MouseAdapter {

    38 usages
    private VentanaPrincesas vista;
    16 usages
    private ModeloTablaPrincesa modelo;


    1 usage
    public ControladorPrincesas(VentanaPrincesas vista) {
        this.vista = vista;
        this.vista.getTblPprincesas().addMouseListener( l: this);
        this.vista.getBtnCargar().addMouseListener( l: this);
        this.vista.getBtnAgregar().addMouseListener( l: this);
        modelo = new ModeloTablaPrincesa();
        this.vista.getTblPprincesas().setModel(modelo);
        this.vista.getBtnActualizar().addMouseListener( l: this);
        this.vista.getBtnBorrar().addMouseListener( l: this);

    }
```

Entonces ya luego mandamos a llamar el único método que no interesa el mouseClicked y para saber de donde viene la señal se hace con if comparando el objeto que nos interesa con la señal dentro de la variable e.

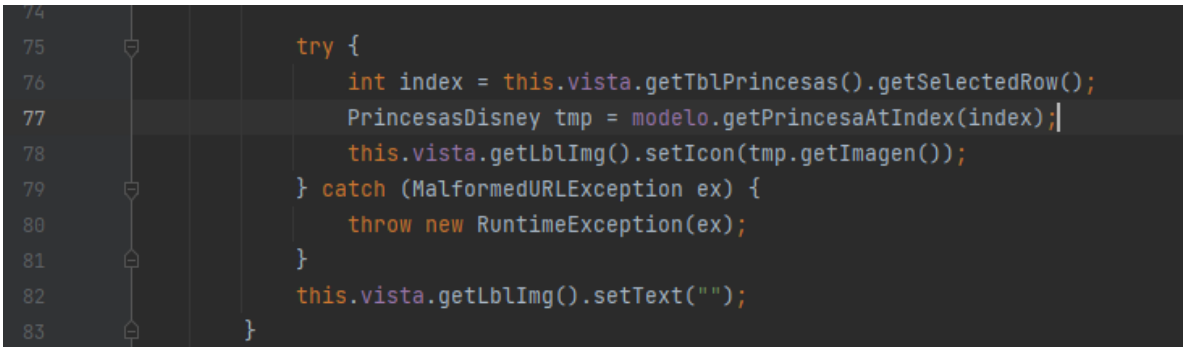
Primero le dimos su acción al botón de cargar lo que hace es mandar a llamar el método de cargarDatos de la clase de ModeloTablaPrincesas para luego mandar ese modelo a la tabla y por último se actualiza esta tabla para mostrar el contenido de la base de datos en la tabla de nuestro programa.

```
    @Override
    public void mouseClicked(MouseEvent e) {
        if(e.getSource() == this.vista.getBtnCargar()){


                modelo.cargarDatos();
                this.vista.getTblPrincesas().setModel(modelo);
                this.vista.getTblPrincesas().updateUI();



        }
```

El siguiente botón al que le dimos su acción al botón de agregar el cual lo que hace es recuperar la información que se encuentra en los text fields para ser asignados dentro de un objeto de tipo princesas para después mandar a llamar el método de agregarPrincesa y si este se cumple mande un cuadro de dialogo donde se diga que se completó con éxito la acción y además se actualice la tabla y muestre la implantación de nuestro registro. Además de que cuando acabe se limpien nuestros campos para poder ser llenados de nuevo.

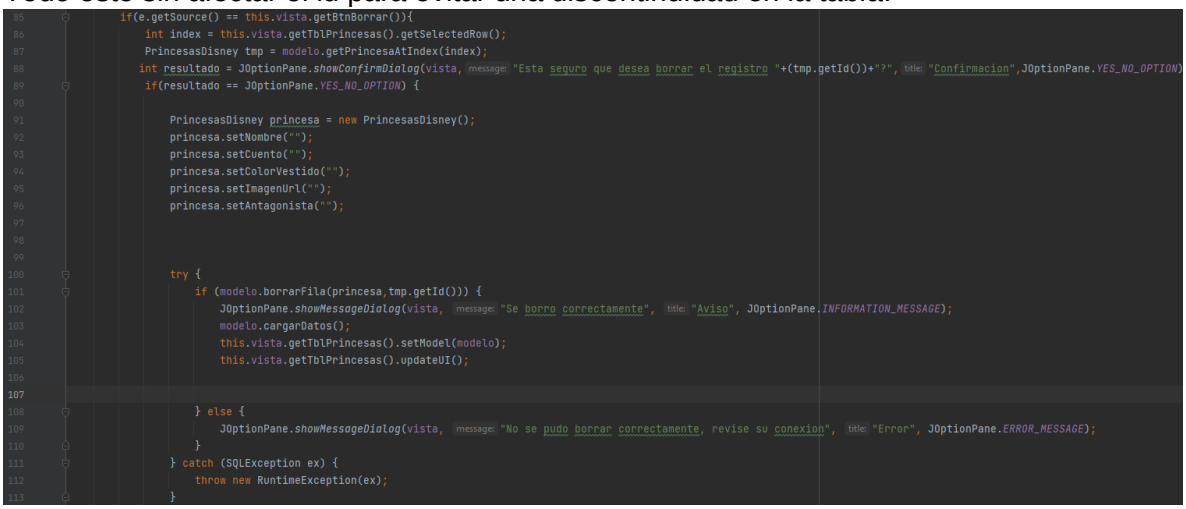
El siguiente evento se comparó con la tabla para asignar el evento de seleccionar una fila y nos regresara una imagen esto se hizo primero sabiendo que fila se seleccionó de la tabla convierto ese dato en un tipo entero llamado index para luego con ese indice mandar a llamar el método de getPrincesaAtIndex lo cual nos regresaba el dato del arraylist en esa posición para así meterlo en una variable temporal.

Esa variable temporal serviría para de ahí sacar la url de esa princesa en especifico y dársela al método de getImagen de la clase PrincesasDisney que lo que hace es transformar esa cadena en un tipo de dato de URL para así poder retornar la imagen e insertarla dentro de un label asignado al panel 3. Todo esto dentro de un catch que se nos solicita por ser una excepción atendida.



El siguiente botón fue el de borrar el cual lo primero que hace es sacar el id de la fila seleccionada de manera interna mientras le pregunta al usurario si esta seguro de querer borrar el registro seleccionado mostrándole la id de dicha fila. Una vez que se confirma se manda un booleano el cual si es true lo que hace este es Crear un objeto de tipo princesa con string vacíos para ser otorgados al método que se manda a llamar de borrarFila que

necesita un objeto de tipo Princesa Disney y el id de la fila al que se le borraran los datos los cuales ya tenemos y ese método regresa un booleano para saber si este se logro y así avisarle al usuario con un cuadro de dialogo que confirme que fue exitoso o que algo fallo. Todo esto sin afectar el id para evitar una discontinuidad en la tabla.



```
if(e.getSource() == this.vista.getBtnBorrar()){
    int index = this.vista.getTblPrincesas().getSelectedRow();
    PrincesasDisney tmp = modelo.getPrincesaAtIndex(index);
    int resultado = JOptionPane.showConfirmDialog(vista, message: "Esta seguro que desea borrar el registro "+(tmp.getId())+"?", title: "Confirmacion",JOptionPane.YES_NO_OPTION)
    if(resultado == JOptionPane.YES_NO_OPTION) {

        PrincesasDisney princesa = new PrincesasDisney();
        princesa.setNombre("");
        princesa.setCuento("");
        princesa.setColorVestido("");
        princesa.setImagenUrl("");
        princesa.setAntagonista("");



        try {
            if (modelo.borrarFila(princesa,tmp.getId())) {
                JOptionPane.showMessageDialog(vista, message: "Se borro correctamente", title: "Aviso", JOptionPane.INFORMATION_MESSAGE);
                modelo.cargarDatos();
                this.vista.getTblPrincesas().setModel(modelo);
                this.vista.getTblPrincesas().updateUI();


            } else {
                JOptionPane.showMessageDialog(vista, message: "No se pudo borrar correctamente, revise su conexion", title: "Error", JOptionPane.ERROR_MESSAGE);
            }
        } catch (SQLException ex) {
            throw new RuntimeException(ex);
        }
```

El ultimo boton asignado es el de actualizar el cual te pregunta que es lo que vas a meter en la celda por medio del cuadro de dialogo y de manera simultanea este programa lo que hace es sacar la id de la fila seleccionada y el numero de columan para dar con la celda para todo esto otorgarselo al metodo de actualizarCelda que nos pide estos tres datos para ya solo esperar el booleano para poder ver si se logro o no y asi solo cargarlo al modelo y del modelo a la tabla para ser actualizada y mostrado en el programa.

```
if (e.getSource() == this.vista.getBtnActualizar()) {
    int index = this.vista.getTblPrincesas().getSelectedRow();
    PrincesasDisney tmp = modelo.getPrincesaAtIndex(index);
    int id = tmp.getId();
    String datoActualizado = JOptionPane.showInputDialog(vista, message: "Inserte el dato a actualizar de "+
            this.vista.getTblPrincesas().getColumnName(vista.getTblPrincesas().getSelectedColumn())+" del registro "
            +id, title: "Actualizar",JOptionPane.INFORMATION_MESSAGE);


    int colIndex = this.vista.getTblPrincesas().getSelectedColumn();

    System.out.println(colIndex+""+datoActualizado+""+id);

    try {
        if(modelo.actualizarCelda(datoActualizado,id,colIndex)){
            System.out.println("se logro");
            modelo.cargarDatos();
            this.vista.getTblPrincesas().setModel(modelo);
            this.vista.getTblPrincesas().updateUI();

        }else{
            System.out.println("no se logro");
        }
    } catch (SQLException ex) {
        throw new RuntimeException(ex);
    }
```

Main

Por último está el main donde lo único que se hace es crear la ventana del programa y después se crea el controlador y se le asigna la vista para poder crear una vez mas el

modelo vista controlador y que así opere de mejor manera el programa y sea más ordenado.

```java
no usages
public class Main {
    no usages
    public static void main(String[] args) {
     VentanaPrincesas vista = new VentanaPrincesas( title: "Princesas Disney");

        ControladorPrincesas controlador = new ControladorPrincesas(vista);



    }
}
```

Lo que permite que nuestro programa se vea así: