

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

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

```
private JTable tblPrincesas;
private JPanel panel2;
private JButton btnCargar;
private JScrollPane scroll;
private JPanel panel3;
private JLabel lblfila;
private JLabel lblImg;
private JPanel panel4;
private JButton btnBorrar;
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.

```
this.setLayout(layoutGeneral);
                panel1 = new JPanel();
                panel1.setBackground(new Color( r: 118, g: 178, b: 204, a: 255));
260
                txtId = new JTextField( columns: 4);
                txtId.setEnabled(false);
                lblCuento = new JLabel( text: "Cuento: ");
                txtCuento = new JTextField( columns: 35);
                lblColorVestido = new JLabel( text: "Color del vestido: ");
                txtColorVestido = new JTextField( columns: 30);
                panel1.add(txtColorVestido);
                txtImagenUrl = new JTextField( columns: 35);
                txtAntagonista = new JTextField( columns: 24);
```

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

```
panel2 = new JPanel();
                panel2.setBackground(new Color( r 234, g: 108, b: 108));
300
                panel2.setLayout(new FlowLayout());
                btnCargar = new JButton( text: "Cargar");
                panel2.add(btnCargar);
                tblPrincesas = new JTable();
                scroll = new JScrollPane(tblPrincesas);
                panel3 = new JPanel();
311
                panel3.setBackground(new Color( r: 142, g: 204, b: 105, a: 255));
                panel3.setLayout(new FlowLayout());
                lblfila = new JLabel( text: "Foto:");
                lblImg = new JLabel( text: "...");
                panel4 = new JPanel();
320
                panel4.setBackground(new Color( r: 137, g: 211, b: 168, a: 255));
                panel4.setLayout(new FlowLayout());
                btnActualizar = new JButton( text: "Actualizar");
```

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.

```
public void limpiar(){
    txtNombre.setText("");
    txtCuento.setText("");
    txtColorVestido.setText("");
    txtImagenUrl.setText("");
    txtAntagonista.setText("");
}
```

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

```
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;

14
    2 usages
    public PrincesasDisney() {
    16
```

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.

```
public PrincesasDisney() {
}

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

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

```
public interface InterfazDAO {

1 usage 1 implementation
public abstract boolean insertar(Object obj) throws SQLException;

1 usage 1 implementation
boolean updte(Object obj, int Id) throws SQLException;

1 no usages 1 implementation
public abstract boolean delete(int id) throws SQLException;
2 usages 1 implementation
public abstract ArrayList obtenerTodo() throws SQLException;
no usages 1 implementation
public abstract ArrayList obtenerTodo() throws SQLException;
no usages 1 implementation
public abstract Object buscarPorId(String id) throws SQLException;

14 ol public abstract Object buscarPorId(String id) throws SQLException;

15 }
```

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.

```
GOverride
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: 3, ((PrincesasDisney) obj).getColorVestido());
pstm.setString( parameterIndex: 3, ((PrincesasDisney) obj).getColorVestido());
pstm.setString( parameterIndex: 4, ((PrincesasDisney) obj).getColorVestido());
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.

```
### String sqtUpdate = "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( parameterIndec: 1, ((PrincesasDisney) obj).getNombre());
pstm.setString( parameterIndec: 3, ((PrincesasDisney) obj).getCuento());
pstm.setString( parameterIndec: 3, ((PrincesasDisney) obj).getCuento());
pstm.setString( parameterIndec: 4, ((PrincesasDisney) obj).getNombre());
pstm.setString( parameterIndec: 5, ((PrincesasDisney) obj).getCuento());
pstm.setString( parameterIndec: 5, ((PrincesasDisney) obj).getAntagonista());

return 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

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.

```
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.

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

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.

```
public void cargarDatos() {

try {
    ArrayList<PrincesasDisney> tirar = pdao.obtenerTodo();

datos = pdao.obtenerTodo();

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.

```
public boolean agregarPrincesa(PrincesasDisney princesa) {

boolean resultado = false;

try {

   if (pdao.insertar(princesa)) {

    datos.add(princesa);

    resultado = true;

} else {

resultado = false;

}

resultado = false;

}

catch (SQLException e) {

   System.out.println(e.getMessage());

}

return resultado;
}
```

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.

```
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 id y lo único que hace es mandar a llamar el método delete y para que la maquina sepa que se borrara se requiere el id. 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. Así 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.

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.

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 **getlmagen** 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.

```
try {
    int index = this.vista.getTblPrincesas().getSelectedRow();
    PrincesasDisney tmp = modelo.getPrincesaAtIndex(index);
    this.vista.getLblImg().setIcon(tmp.getImagen());
} catch (MalformedURLException ex) {
    throw new RuntimeException(ex);
}
this.vista.getLblImg().setText("");
}
```

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 mandar a llamar el método

borrarFila y se le asigna el id que habíamos sacado para así eliminar dicho registro de la base de datos y así actualizar la tabla para mostrase en el programa

```
if(e.getSource() == this.vista.getBtmBornar()){
    int index = this.vista.getBtmBornar()){
    int index = this.vista.getBtmBornar()){
        int new = this.vista.getBtmBornar()}{
        int new = this.vista.getBtmBornar()){
        int new = this.vista.getBtmBornar()}{
        int new = this.g
```

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.

Main

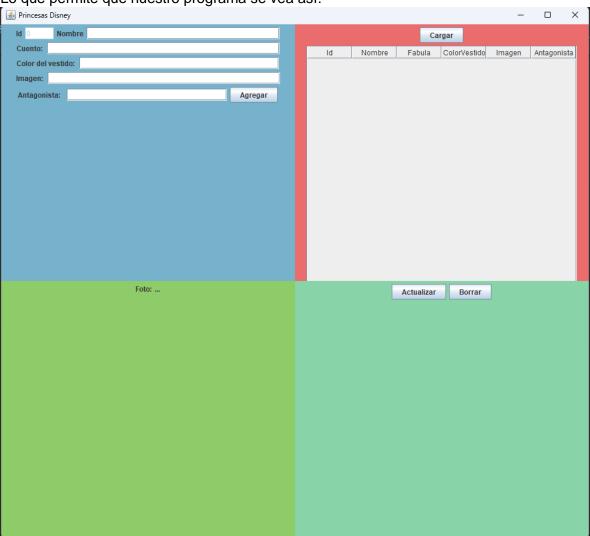
Por último está el main donde lo único que se hace es crear la ventana del programa y después se crea el controlador y se le asigna la vista para poder crear una vez mas el modelo vista controlador y que así opere de mejor manera el programa y sea más ordenado.

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

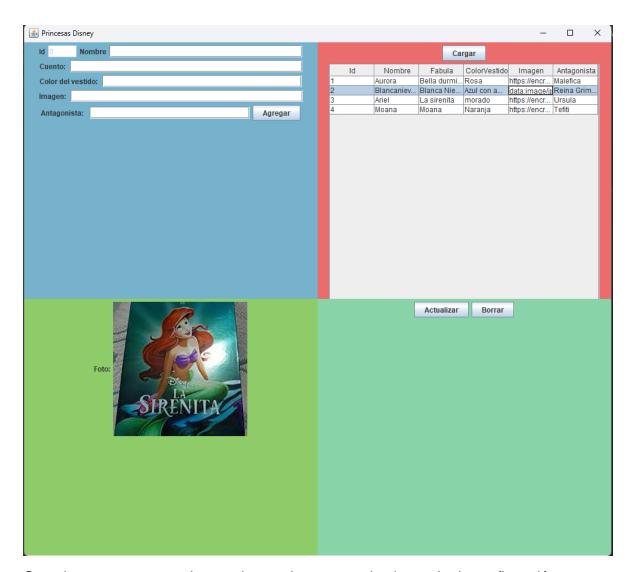
    ControladorPrincesas controlador = new ControladorPrincesas(vista);

    ControladorPrincesas controlador = new ControladorPrincesas(vista);
}
```

Lo que permite que nuestro programa se vea así:

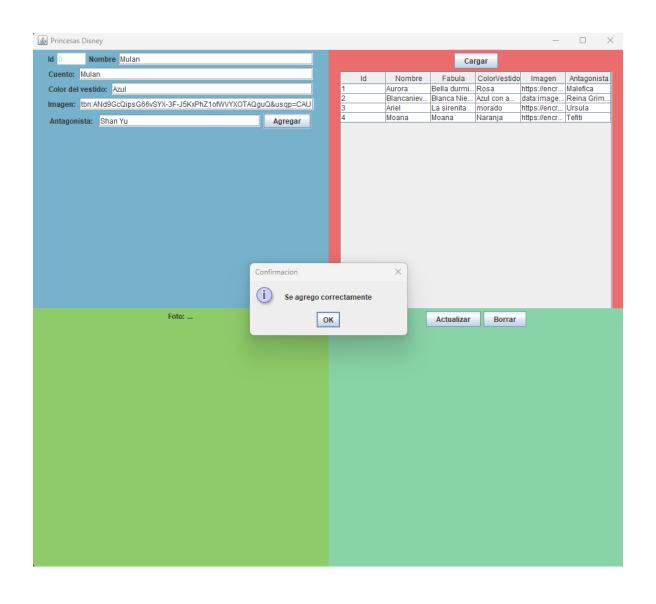


Hasta que se le da al boton de cargar para mostrar el contenido de la base de datos:



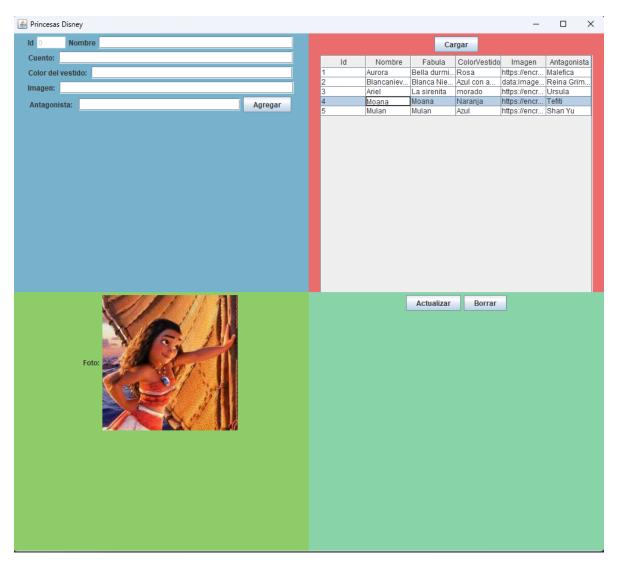
Cuando se agrega un registro se le manda un mensaje al usuario de confirmación y por ultimo se actualiza la base de datos.



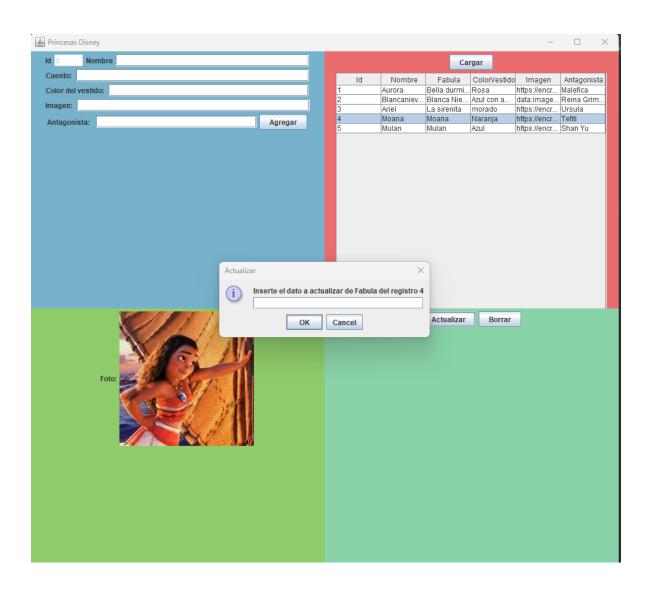


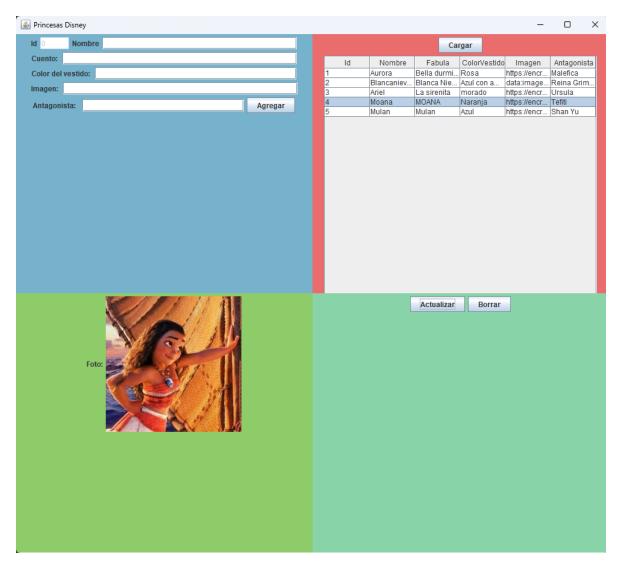


Y así se ve cuando se selecciona una fila de la tabla para mostrar la foto de la princesa:



Así se ve cuando se ocupa el botón de actualizar, se pide el dato a actualizar y se muestra de que celda se cambiaran los datos:





Por ultimo así se ve el botón de borrar para en este caso borrar el registro 6 y al darle confirmar automáticamente se actualiza la tabla y se muestra el cambio:



Cargar

ld	Nombre	Fabula	ColorVestido	Imagen	Antagonista
1	Aurora	Bella durmi	Rosa	https://encr	Malefica
2	Blancaniev	Blanca Nie	Azul con a	data:image	Reina Grim
3	Ariel	La sirenita	morado	https://encr	Ursula
4	Moana	MOANA	Naranja	https://encr	Tefiti
5	Mulan	Mulan	Azul	https://encr	Shan Yu