El futuro digital es de todos

MinTIC

INTERFACES GRÁFICAS
Y BASES DE DATOS

Universidad Industrial de Santander

UIS

Misión TIC 2022

5.1. Introducción

Hasta este momento, todos los resultados de nuestras operaciones se han mostrado en la consola. Pero ahora es momento de introducirnos en las interfaces gráficas.

Las interfaces gráficas (GUI, del inglés *Graphical User Interface)* ofrecen al usuario botones, menú, barra de herramientas, ventas, cuadros de diálogo y similares, que conocemos en nuestra interacción como usuarios. Un ejemplo de interfaz gráfica o GUI, podría ser un procesador de texto como lo es Microsoft Word o el bloc de notas.

Estos elementos que componen la GUI, son manejados por eventos y se desarrollan usando bibliotecas o librerías. Estas librerías proporcionan al desarrollador un conjunto de herramientas para el desarrollo de las interfaces gráficas que son compatibles para todos los sistemas operativos de escritorio.

5.2. Swing como interfaz gráfica para Java

Swing es una biblioteca de clases que nos ofrece la capacidad de construir interfaces gráficas para aplicaciones en escritorio en Java.

Swing nos provee de dos elementos principales para la construcción de interfaces gráficas:

Contenedores: Como su nombre lo dice, contienen un conjunto de elementos o componentes.

Componentes: Elementos gráficos como botones, cuadros de texto que se pueden organizar en contenedores.

Existen tres tipos de contenedores de alto nivel: JFrame, JDialog y JApplet.

Ya que estos son de alto nivel, cualquier otro contenedor o componente debe construirse en su interior.

Para usar esta librería debemos importarla de la siguiente manera:

```
import javax.swing.*;
```

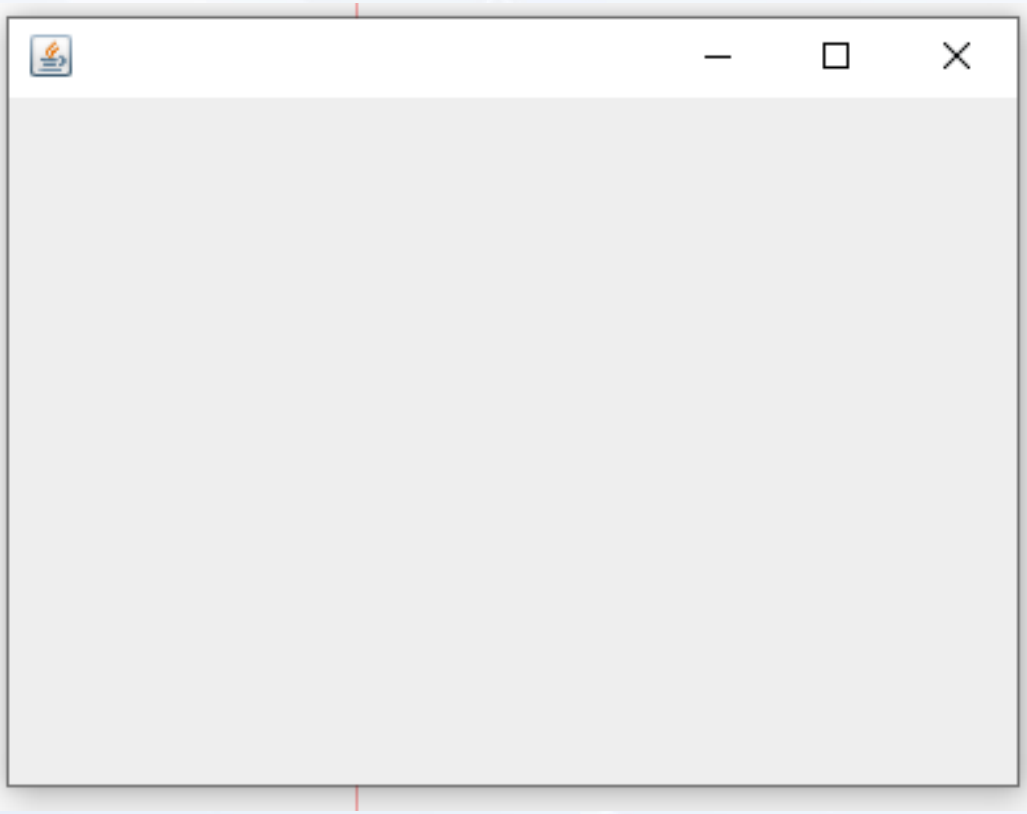
5.2.1. Ventanas con Swing

El componente más elemental que se necesita para construir una interfaz gráfica con *Swing* es la clase JFrame. Esta clase encapsula una ventana del sistema operativo de escritorio donde se ejecuta el programa (Linux, Microsoft Windows, Mac OS).

Un ejemplo de una ventana simple con Swing se puede hacer con el siguiente código

```
import javax.swing.*; // Se importa todas las clases necesarias para realizar interfaces gráficas
public class Datos extends JFrame{ // Creamos una clase que hereda de la clase JFrame
    public Datos() { //Constructor de la clase Datos
    }
    public static void main(String[] ar) {
        Datos formulario1=new Datos();
        formulario1.setBounds(5,10,200,150); // Este método nos permite definir la posición y
tamaño de la ventana
        formulario1.setVisible(true); // Este método nos permite definir la visibilidad de la
ventana
        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Este método nos permite
definir la operación a realizar una vez se cierre la ventana. En este caso salir de la aplicación.
    }
}
```

Lo cual nos creará una ventana de este estilo:



(Nota explicativa: En la imagen se observa una ventana del sistema operativo Windows con fondo gris y controles de minimizar, maximizar y cerrar)

En el constructor de la clase Formulario, deshabilitamos el Layout o diseño heredado. En el main una vez se crea un objeto tipo Formulario, usamos el método setBounds para ubicar la ventana o JFrame en la fila 10 y columna 5 de la pantalla de nuestro computador y con una altura de 150 pixeles y un ancho de 200 pixeles .

5.2.2. Etiquetas con Swing

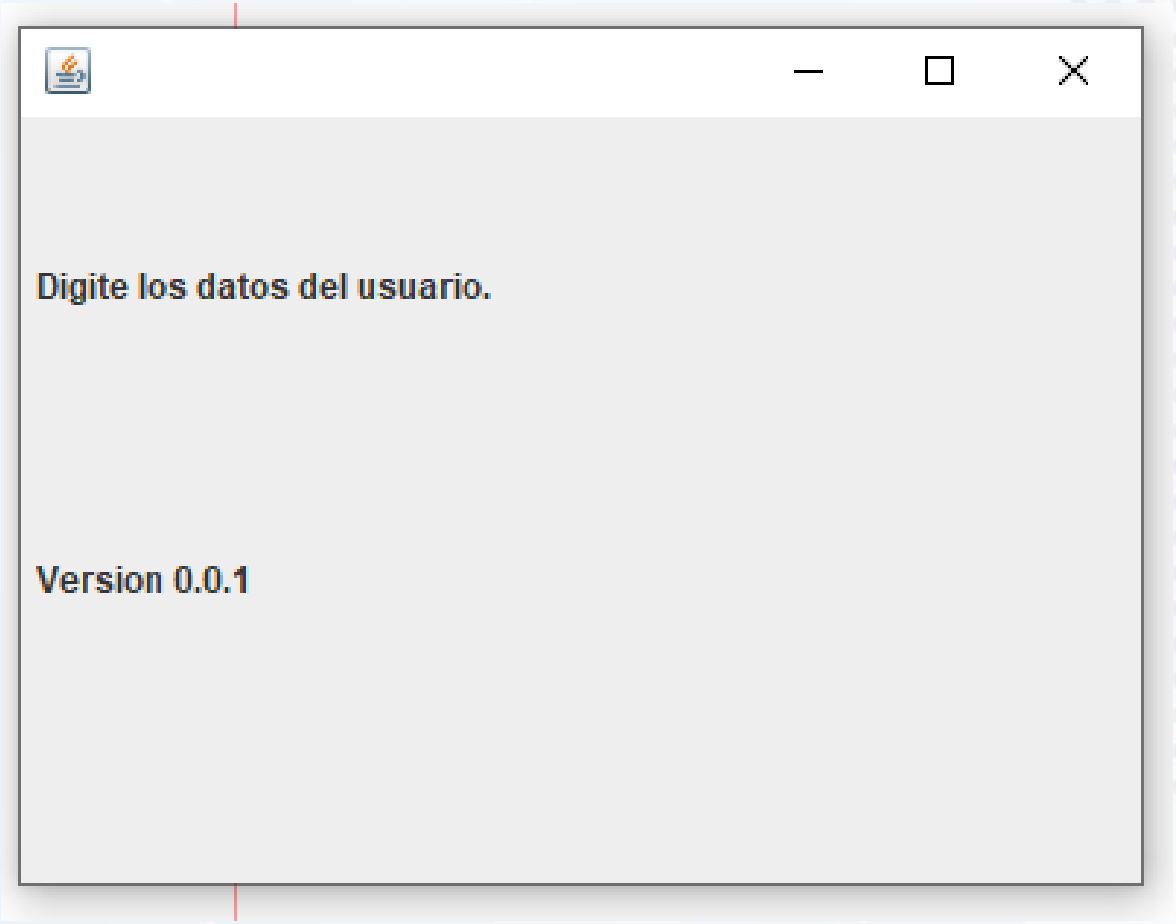
Las etiquetas nos permiten colocar un texto para describir o informar al usuario. La clase JLabel se encargará de proveernos esta utilidad. Un ejemplo para crear y agregar una etiqueta podría ser el siguiente:

```
label1=new JLabel("Digite los datos del usuario."); // Se crea un objeto con el texto que
tendrá la  etiqueta
label1.setBounds(5,50,150,15); // De manera similar a la ventana ubicamos la posición y
tamaño de la etiqueta
 add(label1); // Con este método agrega la etiqueta a la ventana o JFrame
```

Si mantenemos el código anterior y añadimos algunas etiquetas sería de la siguiente manera

```
import javax.swing.*;
public class Datos extends JFrame{

private JLabel label1,label2; // inicializamos los atributos donde se va almacenar el objeto tipo
JLabel

    public Datos() {
        label1=new JLabel("Digite los datos del usuario.");
        label1.setBounds(5,50,300,15);
        add(label1);
        label2=new JLabel("Version 0.0.1");
        label2.setBounds(5,150,150,15);
        add(label2);
    }
    public static void main(String[] ar) {
        Datos formulario1=new Datos();
        formulario1.setBounds(5,10,200,150);
        formulario1.setVisible(true);
        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Lo cual nos creará una ventana de este estilo:



(Nota explicativa: En la imagen se observa una ventana del sistema operativo Windows con fondo gris, dos etiquetas que muestran el texto programado y controles de minimizar, maximizar y cerrar)

5.2.3. Paneles con Swing

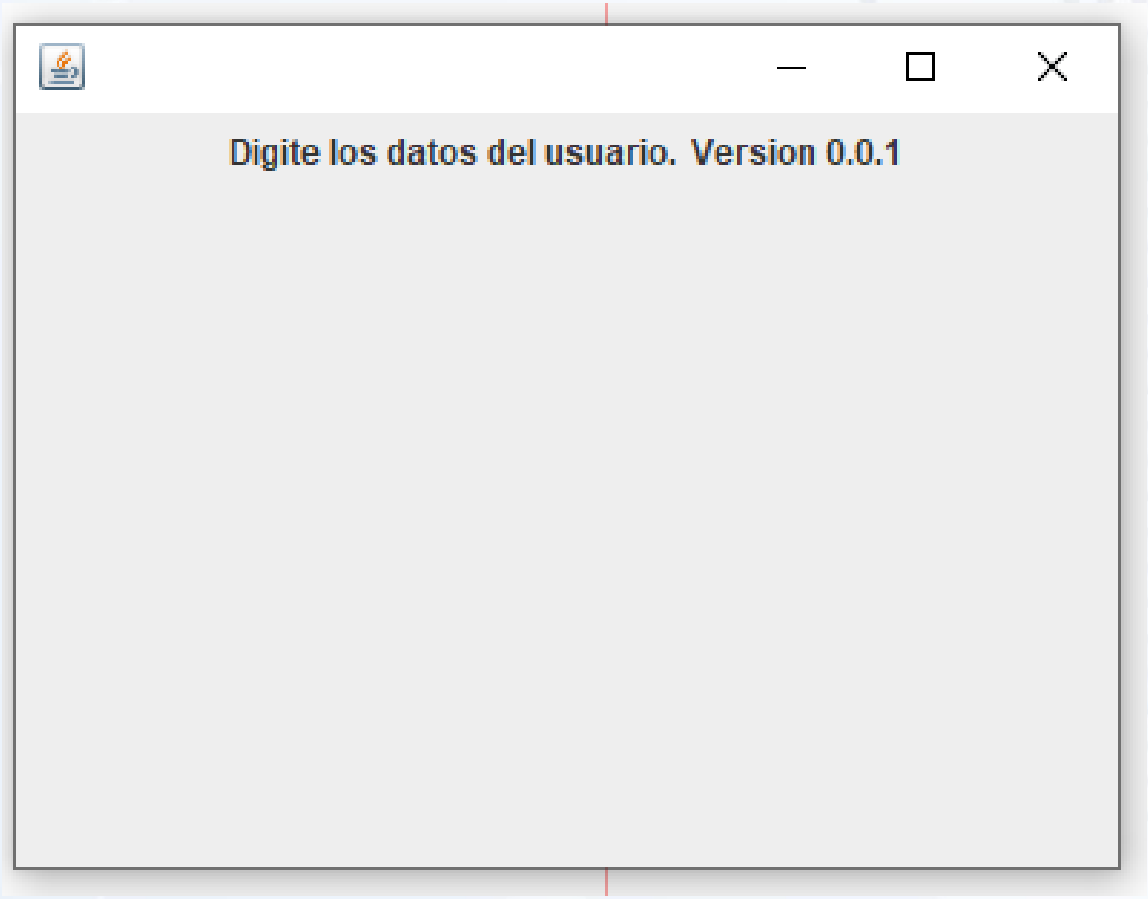
Los paneles son contenedores donde es posible ubicar elementos dentro de él para facilitar la movilización de varios elementos con mover solamente el panel. La clase JPanel nos proveerá de esta utilidad. Un ejemplo para crear un panel y agregar algunas etiquetas podría ser el siguiente:

```
   JPanel panel1 = new JPanel(); //Se crea una instancia de la clase JPanel
     label1=new JLabel("Digite los datos del usuario."); // Etiqueta creada para agregar al
panel
     panel1.add(label1);  //Se agrega la etiqueta al panel
     label2=new JLabel("Version 0.0.1");    // Segunda etiqueta creada para agregar al panel
     panel1.add(label2); // Se agrega la segunda etiqueta al panel
     add(panel1); // Agregar el panel al JFrame o ventana principal
```

Si mantenemos el código anterior y añadimos el panel, sería de la siguiente manera

```
import javax.swing.*;
public class Datos extends JFrame{

private JLabel label1,label2;

    public Datos() {
         JPanel panel1 = new JPanel();
         label1=new JLabel("Digite los datos del usuario.");
         panel1.add(label1);
         label2=new JLabel("Version 0.0.1");
         panel1.add(label2);
         add(panel1);
    }
    public static void main(String[] ar) {
        Datos formulario1=new Datos();
        formulario1.setBounds(5,10,200,150);
        formulario1.setVisible(true);
        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Lo cual nos creará una ventana de este estilo:



(Nota explicativa: En la imagen se observa una ventana del sistema operativo Windows con fondo gris, dos etiquetas que muestran el texto programado con un diferente orden que el anterior ejemplo y controles de minimizar, maximizar y cerrar)

5.2.4. Entrada de datos

La clase JTextField permite al usuario ingresar datos en un campo de texto. Un JFrame con una entrada de texto se podría codificar de la siguiente manera:

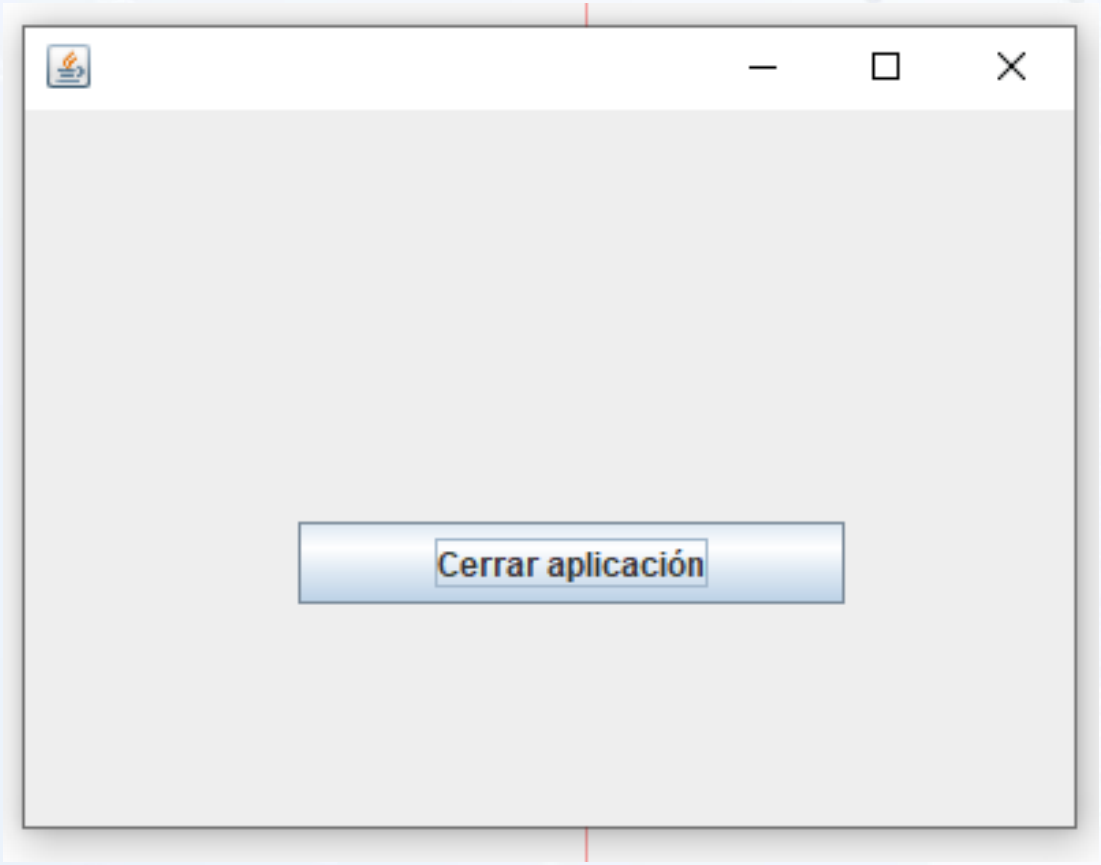
```
    campoTexto1=new JTextField(); // Se crea la instancia de la clase JTextField
      campoTexto1.setBounds(110,20,140,15); // Define la posición y el tamaño del
componente
      add(campoTexto1); // Se agrega el componente a la ventana o JFrame
```

5.2.5. Botones con Swing

La clase JButton nos mostrará un botón para que el usuario pueda presionarlo. La forma de agregar un botón a un JFrame o ventana es similar al proceso realizado con las etiquetas o JLabel. Un JFrame con un botón se podría codificar de la siguiente manera:

```
import javax.swing.*;
public class Datos extends JFrame{

JButton boton1;

    public Datos() {
          setLayout(null); // Define el diseño de la ventana. En este caso ninguno para mostrar el botón.
          boton1=new JButton("Cerrar aplicación"); // Se crea la instancia de la clase JButton
          boton1.setBounds(100,150,200,30);  //Se define la posición del botón en la ventana
          add(boton1); // Se agrega el botón a la ventana o JFrame
    }
    public static void main(String[] ar) {
        Datos formulario1=new Datos();
        formulario1.setBounds(5,10,200,150);
        formulario1.setVisible(true);
        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Lo cual nos creará una ventana de este estilo:



(Nota explicativa: En la imagen se observa una ventana del sistema operativo Windows con fondo gris, un botón con el texto "cerrar aplicación" y controles de minimizar, maximizar y cerrar)

5.2.6. Interacción a través de eventos

En *Swing*, en el momento en que un usuario realiza una interacción con la aplicación, se ejecuta un evento. Para que un determinado componente pueda realizar una acción de respuesta a esta interacción, se debe definir un "escuchador" o listener con al menos un método específico que se ejecutará al activarse un evento determinado

Las siguientes instrucciones crean un botón que una vez el usuario oprime, se imprime un mensaje en la consola:

```
JButton pulsar=new JButton("un botón para pulsar");
pulsar.addActionListener(new ActionListener()
{
     public void actionPerformed(ActionEvent e)
         {
           System.out.println("el botón fue
presionado");
          }
})
```

Para juntar todo lo que hemos visto vamos a pensar que estamos en una empresa de desarrollo software y nos piden realizar una implementación de una interfaz gráfica para cambiar el nombre de la ventana de una aplicación más grande.

Los requerimientos son los siguientes:

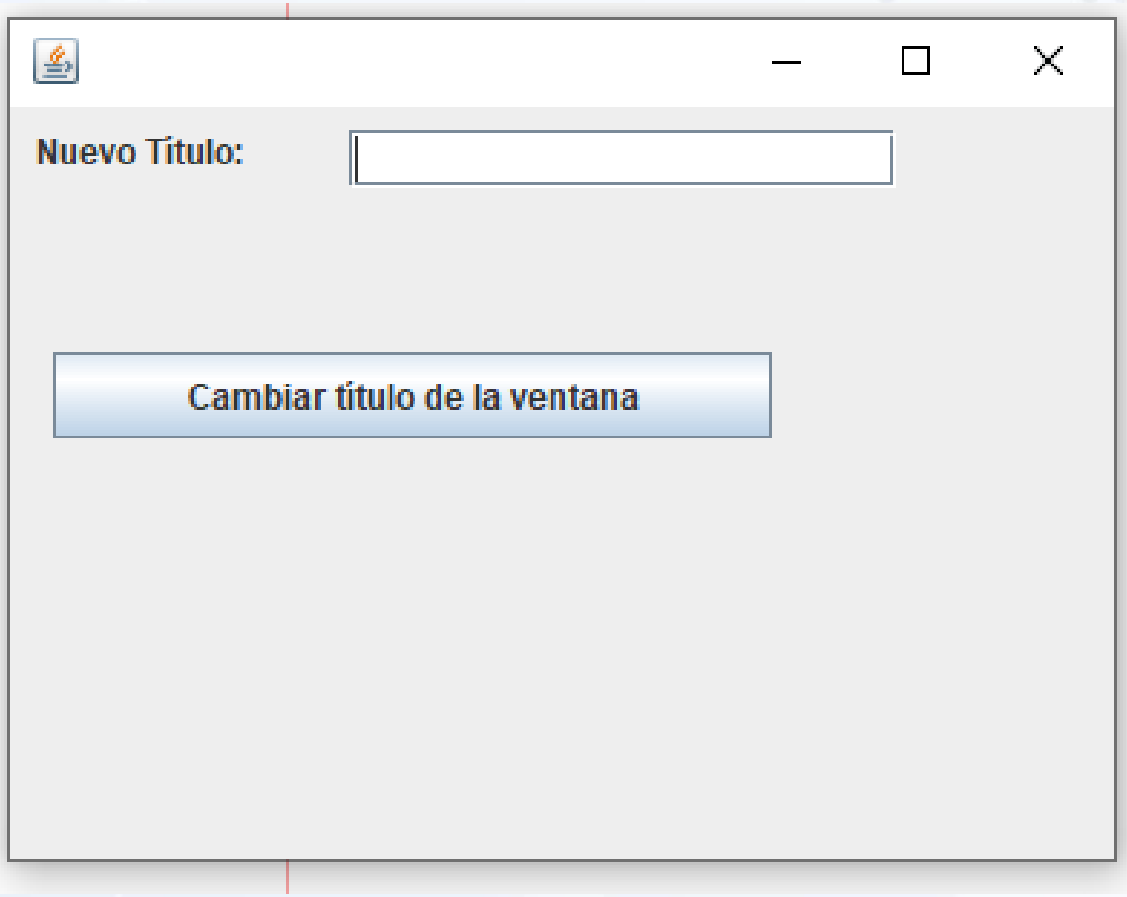
•En una ventana de una altura de 150 pixeles y un ancho de 200 pixeles crear una etiqueta llamada "Nuevo Título".

•En frente de la etiqueta hay un campo de entrada de datos para recibir datos del usuario.

•Debajo de la etiqueta y el campo de entrada de datos un botón con el texto "Cambiar título de la ventana". Asegúrese que el texto pueda leerse en el botón

•Cuando el botón se pulse el nombre de la ventana debe cambiar al ingresado por el usuario en el campo de entrada de datos.

Se debe ver de la siguiente manera:

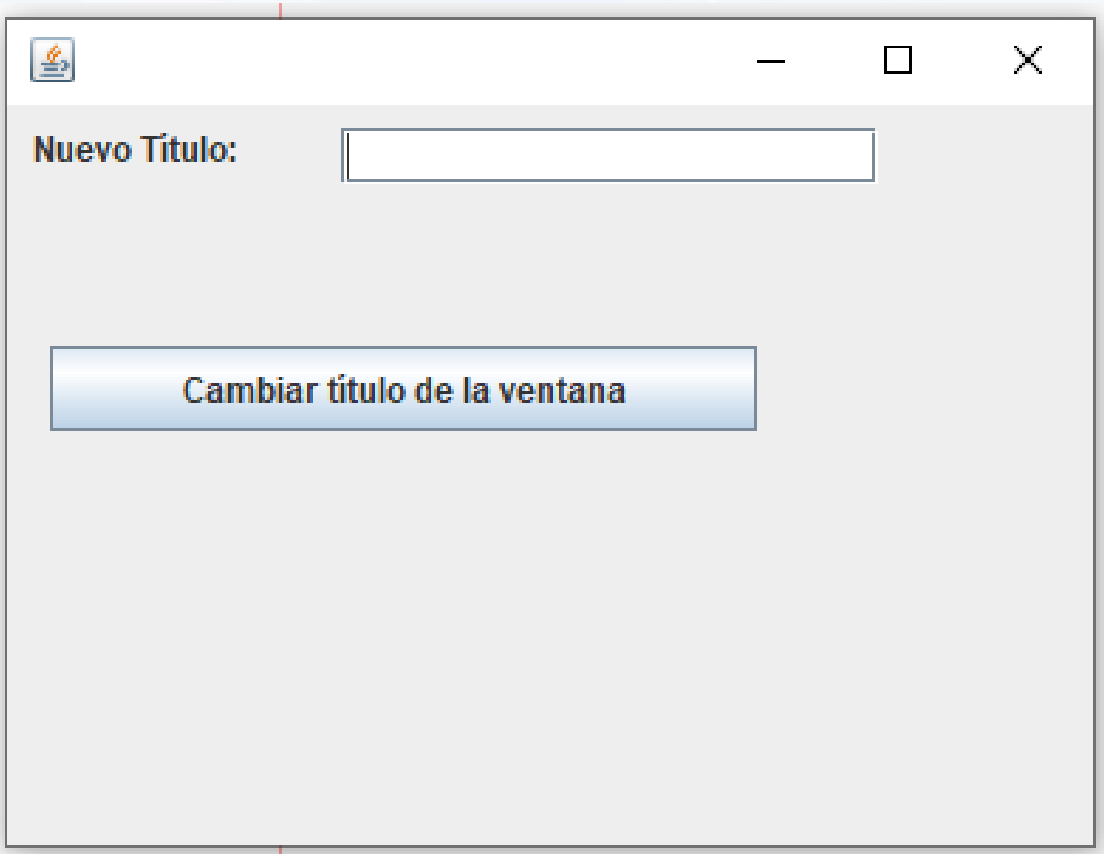


(Nota explicativa: En la imagen se observa una ventana del sistema operativo Windows con fondo gris, un botón con el texto "cambiar título de la ventana", un cuadro de texto y una etiqueta.)

Finalmente, podemos utilizar todos los componentes para crear esa interfaz gráfica simple:

```
import javax.swing.*;
import java.awt.event.*;


public class Datos extends JFrame implements ActionListener{

private JTextField campoTexto; // Se define la variable para el campo de texto
private JLabel etiqueta1; // Se define la variable para la etiqueta
private JButton botonCambiar; // Se define la variable para el botón
public Datos(){
     setLayout(null);
     etiqueta1=new JLabel("Nuevo Título:");
     etiqueta1.setBounds(9,5,150,20);
     add(etiqueta1);
     campoTexto=new JTextField();
     campoTexto.setBounds(118,8,190,20);
     add(campoTexto);
     botonCambiar=new JButton("Cambiar título de la ventana");
     botonCambiar.setBounds(15,85,250,30);
     add(botonCambiar);
     botonCambiar.addActionListener(this); //Agrega el listener al botón para detectar los eventos de la
aplicación
```

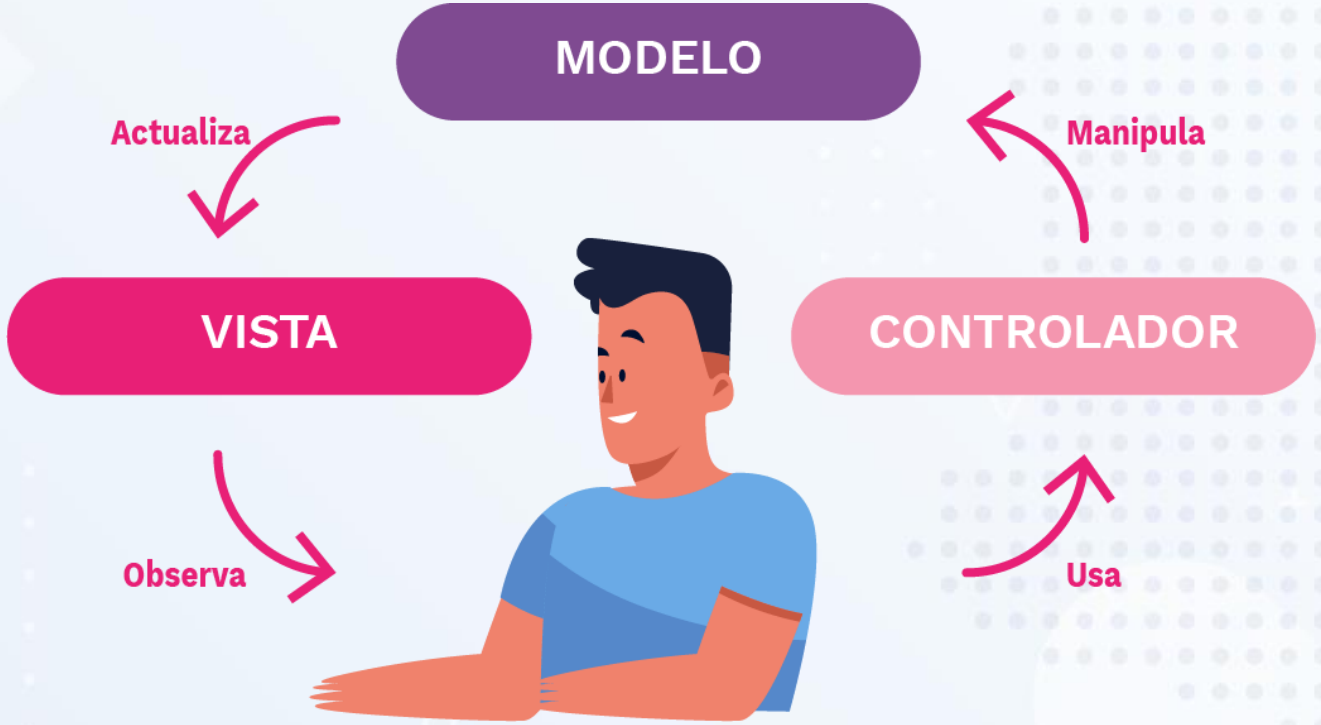
```
}
public void actionPerformed(ActionEvent e) {  // Método que se ejecuta cuando un evento es
realizado
     if (e.getSource()==botonCambiar) { // Comprobamos si el evento pertenece al botón creado
“botonCambiar”
          String tituloCampo=campoTexto.getText(); // Obtenemos el string guardado en el campo
de texto “campoTexto”
          setTitle(tituloCampo); // Definimos el título de la ventana de la aplicación con el
texto obtenido del campo de texto
     }
}
  public static void main(String[] ar) {
     Datos formulario1=new Datos();
     formulario1.setBounds(5,10,200,150);
     formulario1.setVisible(true);
     formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```



(Nota explicativa: En la imagen se observa una ventana del sistema operativo Windows con fondo gris, un botón con el texto "cambiar título de la ventana", un cuadro de texto y una etiqueta.)

Si colocamos un texto en el campo en blanco y oprimimos el botón, el título de la ventana cambiará con el texto introducido. De esta manera, se está interactuando y respondiendo al usuario.

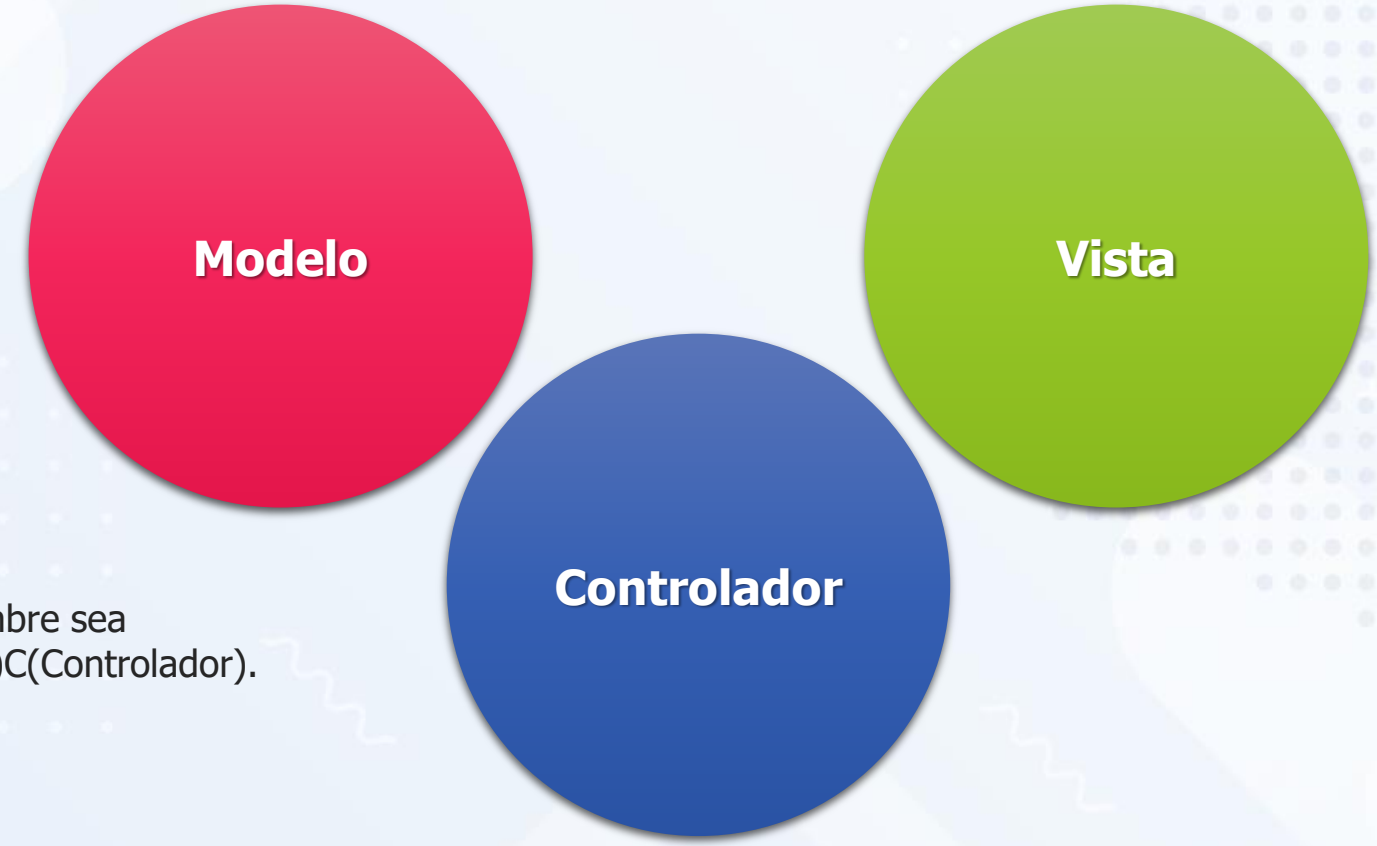
5.3. Modelo MVC con swing



(Nota explicativa: Se observa el modelo MVC desde la interacción del usuario con el controlador, pasando por el modelo y finalizando en la vista para volver a comenzar el ciclo)

El modelo MVC es un diseño de software que propone diferenciar el código programado en relación al rol que cumple en la aplicación.

Se proponen los siguientes roles del código en una aplicación MVC:



De allí que su nombre sea M(Modelo)V(Vista)C(Controlador).

Modelo

El modelo es la representación de la lógica de toda la aplicación. Validaciones, consultas y gestión del acceso a la base de datos, son ejecutados por el modelo. Se conoce como la lógica del negocio

Vista

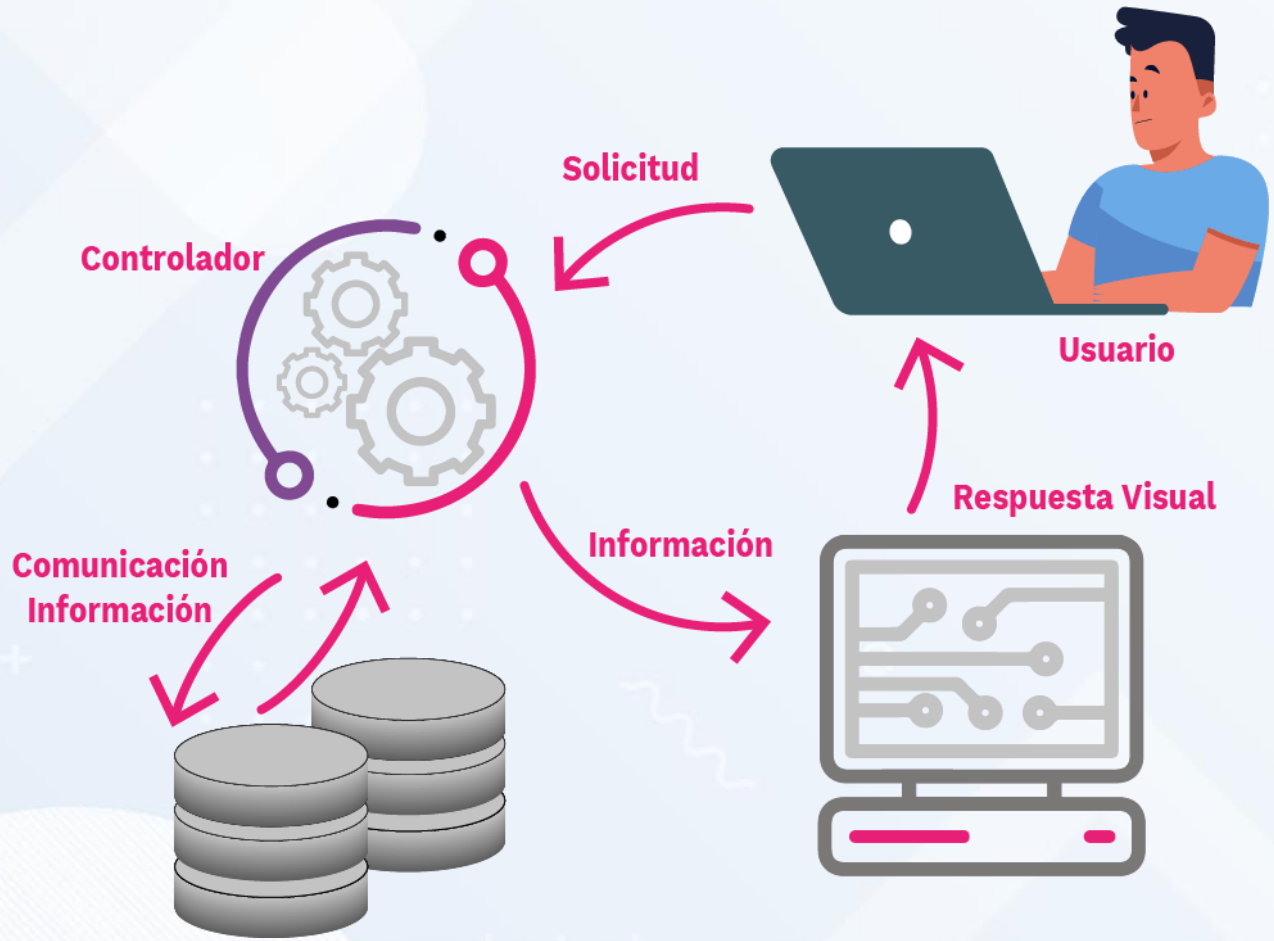
Es la representación del modelo (información y lógica del negocio) en un formato apropiado para interactuar, es decir, la interfaz de usuario.

Controlador

Responde a los eventos realizados por el usuario desde la interfaz y llama los métodos correspondientes a las respuestas de estos eventos. Por ejemplo, solicitar una información. También es posible para el controlador, enviar comunicaciones a la vista. Es por esto que se dice que el controlador es el intermediario entre la vista y el modelo.

Interacción entre los roles

A continuación, ilustramos cómo interactúan estos roles.

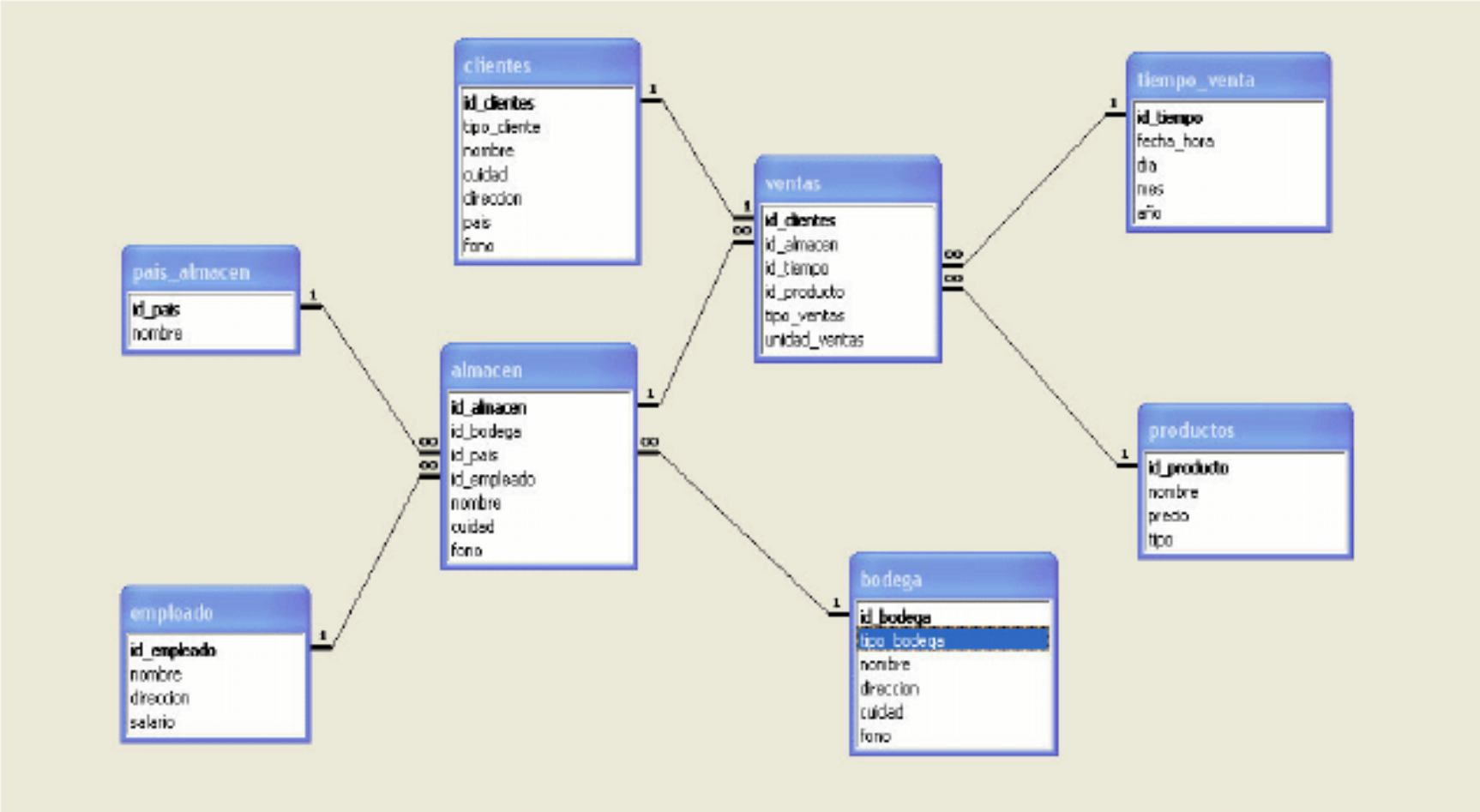


(Nota explicativa: Se tiene de primera mano al usuario, este usuario hace una solicitud al controlador,quien se comunica con el modelo para obtener la información, una vez procesada la información, el modelo le devuelve esa información procesada al controlador, luego el controlador va a enviar esta información a la vista. Finalmente, el usuario va a tener una respuesta visual de su solicitud)

5.4. Bases de datos relacionales

Las bases de datos son un elemento clave para el funcionamiento de las aplicaciones y dada su importancia, se abordará el tema para tener conocimiento de algunas de sus características. Actualmente, existen 2 tipos de bases de datos; Relacionales y no relacionales. Para esta unidad, centraremos nuestra atención en las relacionales y sus tipos de datos

Base de datos relacional: Los tipos de bases relacionales se caracterizan por almacenar datos que están relacionados entre sí y proporcionan puntos de acceso para su consulta. Además su diseño está basado en la representación en tablas, que mediante un modelo relacional, muestra de forma intuitiva y directa la representación de sus datos.



(Nota explicativa: se muestra en la figura, un ejemplo de la representación de una base de datos con tablas relacionadas)

En una base de datos relacional sus registros o tuplas son identificados por una o varias claves de acceso para encontrar un registro específico para modificarlo, consultarlo o eliminarlo.

(Nota explicativa: se muestra en la figura, un ejemplo de 3 tablas relacionadas con los datos de un empleado y su relación con los datos de nóminas y vacaciones)

5.4.1. Tipos de datos

Existen diferentes tipos de datos para almacenar en las bases de datos, pero centraremos la atención en los siguientes permitidos para *Sqlite,* que será nuestro sistema de gestión de base de datos:

Ejemplos de tipos de datos para creación de tablas o expresiones de cambios de tipo
INT,INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB
BLOB (Tipo de dato no especificado)
REAL DOUBLE DOUBLE PRECISION FLOAT
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME

tomado de https://www.sqlite.org/datatype3.html

5.4.2. Motores y sistemas gestores de bases de datos

Los motores de base de datos y los sistemas gestores, están estrechamente relacionados, esto debido a que el motor de base de datos es el componente de software principal para que un sistema de gestión logre realizar operaciones de creación , lectura, actualización y eliminación de datos; lo que se conoce como un CRUD, proveniente de las iniciales de las palabras **C**reate, **R**ead**, U**pdate y **D**elete

Principales sistemas de gestión de bases de datos

(Nota explicativa ; Se muestran en la imágen algunos de los principales sistemas de gestión de bases de datos, Oracle,SQL Server, MySQL, Cassandra y postgreSQL)

Administración de bases de datos

(Nota explicativa: En la imagen se muestran algunos de los software para administración de bases de datos , DBeaver, DB Browser,MySQL Workbench, con phpMyAdmin)

5.4.3. SQLite

Para el desarrollo de este ciclo, utilizaremos *SQLite* que es un sistema de gestión de bases de datos liviano y que no demanda mayor dificultad para utilizar debido a que es compatible con las características ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), y que basa su funcionamiento en acceso a una base de datos tipo archivo portable, ideal para aplicaciones que no requieran un proceso independiente para comunicarse con el programa principal .

Para este ciclo, la única herramienta que debemos instalar es *DB Browser for SQLite,* la cual puede ser descargada en el siguiente enlace:

https://sqlitebrowser.org/

5.4.3.1. Operaciones a bases de datos:

Lenguaje de control de datos DCL

Este lenguaje proporcionado por el sistema de gestión de datos facilita comandos SQL, que principalmente son utilizados por los administradores de bases de datos para gestionar operaciones como:

- Asignación de permisos para usuarios mediante el comando GRANT
- Eliminación de permisos a usuarios mediante el comando REVOKE
- Conceder o denegar para las tareas CONNECT, SELECT,INSERT, UPDATE,DELETE y USAGE

Lenguaje de manejo DML

Estas operaciones están relacionadas con lo que se denomina un CRUD (Create, Read, Update, Delete)
El lenguaje de manipulación de datos está definido por las tareas mostradas en la siguiente tabla:

Sigla CRUD	Operación	Comando	Estructura
Create	Creación de un nuevo registro en base de datos	INSERT	INSERT INTO nombreDeTabla(columna1, columna2, columna3, ...) VALUES (valor1, valor2, valor3, ...);
Read	Lectura de registros de base de datos	SELECT	SELECT columna1, columna2, ... FROM nombreDeTabla; SELECT * FROM table_name;
Update	Actualización de registros de base de datos	UPDATE	UPDATE nombreDeTabla SET columna1= valor1, columna2= valor2, ... WHERE condition;
Delete	Eliminación de registros de base de datos	DELETE	DELETE FROM nombreDeTabla WHERE condicion;

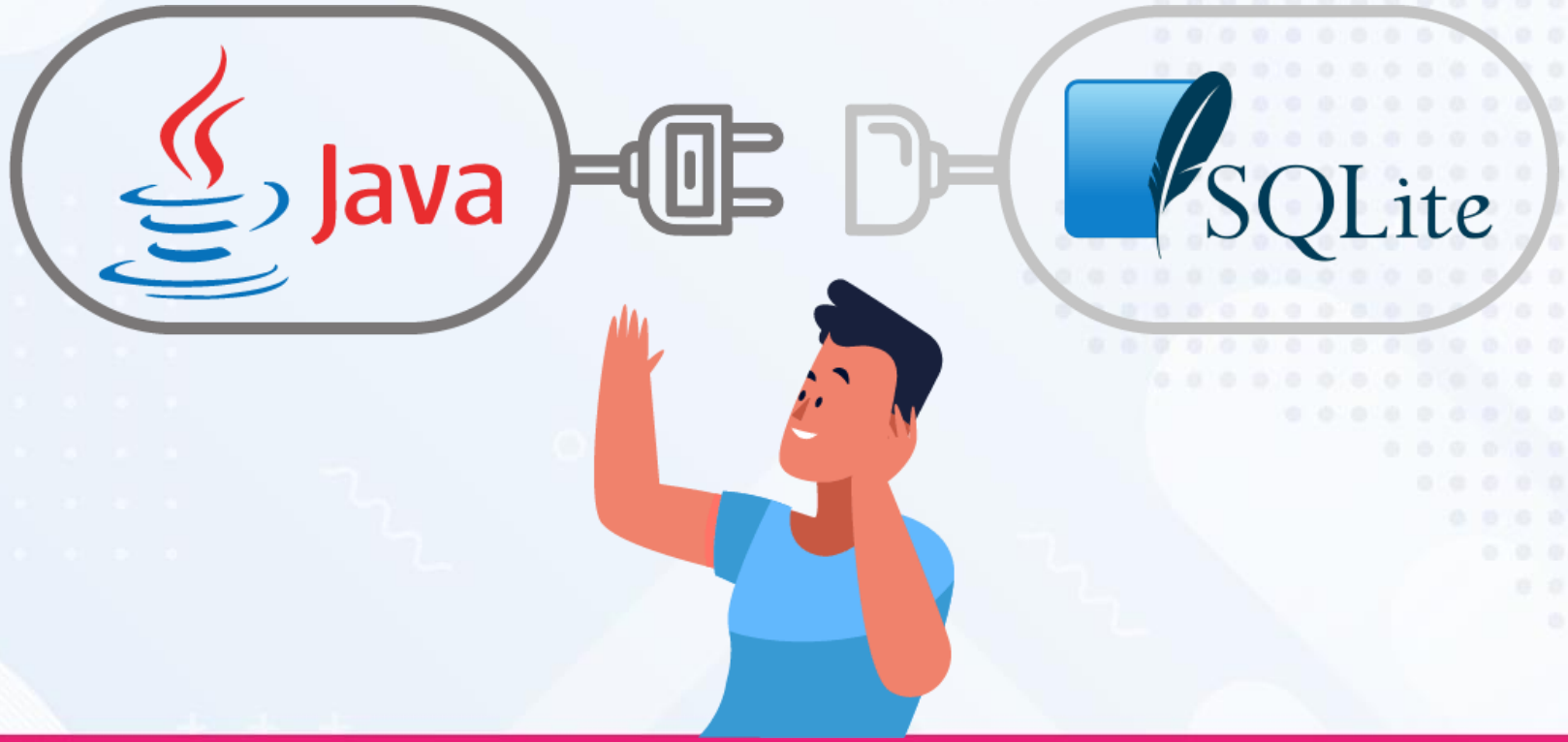
Lenguaje de definición DDL

Este lenguaje proporcionado por el sistema de gestión de datos, facilita comandos SQL, que principalmente son utilizados por los administradores de bases de datos para definir las estructuras que almacenarán los datos mediante operaciones como :

- Creación de nuevas bases de datos y tablas mediante los comandos CREATE, CREATE TABLE
- Agregar eliminar o modificar columnas de una tabla existente mediante el comando ALTER
- Eliminación de bases de datos, tablas o índices mediante el comando DROP

Lenguaje de definición de datos (Consultas a bases de datos).

Para realizar las operaciones a base de datos utilizaremos una Conexión JDBC. *Java Database Connector,* como se muestra en la siguiente figura es un puente que permite conectar la aplicación java con la base de datos.



Para realizar la conexión a base de datos y transacciones utilizaremos la siguiente clase Java

```
import java.sql.*;
import java.util.logging.*;

public class ConexionBD {

    // Configuracion de la conexion a la base de datos
    private String url = "";
    public Connection con = null;
    private Statement stmt = null;
    private ResultSet rs = null;

    //Constructor sin parmetros
    public ConexionBD() {

        url = "jdbc:sqlite:demo1.db";
        try {
            // Realizar la conexion
            con = DriverManager.getConnection(url);
            if (con != null) {
                DatabaseMetaData meta = con.getMetaData();
                System.out.println("Base de datos conectada " + meta.getDriverName());
            }
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }

    }
```

```
//Retornar la conexin
    public Connection getConnection() {
        return con;
    }

    //Cerrar la conexin
    public void closeConnection(Connection con) {
        if (con != null) {
            try {
                con.close();
            } catch (SQLException ex) {
                Logger.getLogger(ConexionBD.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    // Mtodo que devuelve un ResultSet de una consulta (tratamiento de SELECT)
    public ResultSet consultarBD(String sentencia) {
        try {
            stmt = con.createStatement();
            rs = stmt.executeQuery(sentencia);
        } catch (SQLException sqlex) {
            System.out.println(sqlex.getMessage());
        } catch (RuntimeException rex) {
            System.out.println(rex.getMessage());
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        return rs;
    }
```

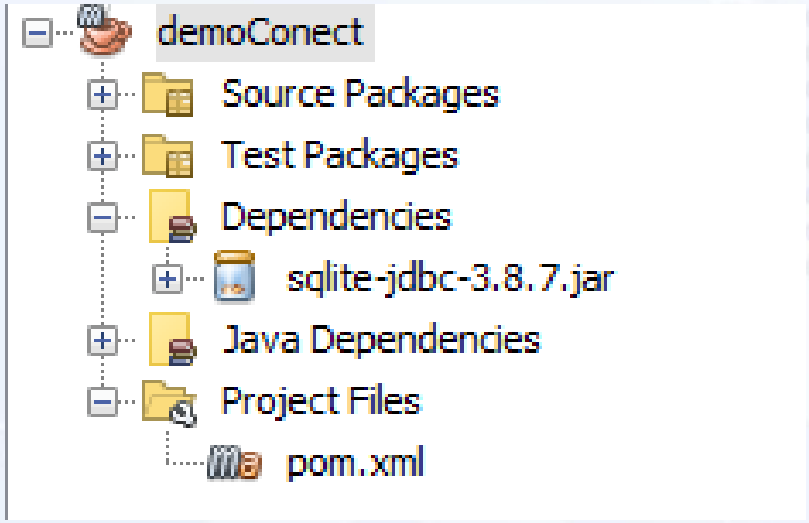
```
// Metodo que realiza un INSERT y devuelve TRUE si la operacin fue existosa
    public boolean insertarBD(String sentencia) {
        try {
            stmt = con.createStatement();
            stmt.execute(sentencia);
        } catch (SQLException | RuntimeException sqlex) {
            System.out.println("ERROR RUTINA: " + sqlex);
            return false;
        }
        return true;
    }

    public boolean borrarBD(String sentencia) {
        try {
            stmt = con.createStatement();
            stmt.execute(sentencia);
        } catch (SQLException | RuntimeException sqlex) {
            System.out.println("ERROR RUTINA: " + sqlex);
            return false;
        }
        return true;
    }
```

```
// Mtodo que realiza una operacin como UPDATE, DELETE, CREATE TABLE, entre otras
   // y devuelve TRUE si la operacin fue existosa
   public boolean actualizarBD(String sentencia) {
       try {
           stmt = con.createStatement();
           stmt.executeUpdate(sentencia);
       } catch (SQLException | RuntimeException sqlex) {
           System.out.println("ERROR RUTINA: " + sqlex);
           return false;
       }
       return true;
   }

   public boolean setAutoCommitBD(boolean parametro) {
       try {
           con.setAutoCommit(parametro);
       } catch (SQLException sqlex) {
           System.out.println("Error al configurar el autoCommit " + sqlex.getMessage());
           return false;
       }
       return true;
   }

   public void cerrarConexion() {
       closeConnection(con);
   }
```

```
public boolean commitBD() {
        try {
            con.commit();
            return true;
        } catch (SQLException sqlex) {
            System.out.println("Error al hacer commit " + sqlex.getMessage());
            return false;
        }
    }

    public boolean rollbackBD() {
        try {
            con.rollback();
            return true;
        } catch (SQLException sqlex) {
            System.out.println("Error al hacer rollback " + sqlex.getMessage());
            return false;
        }
    }
}
```

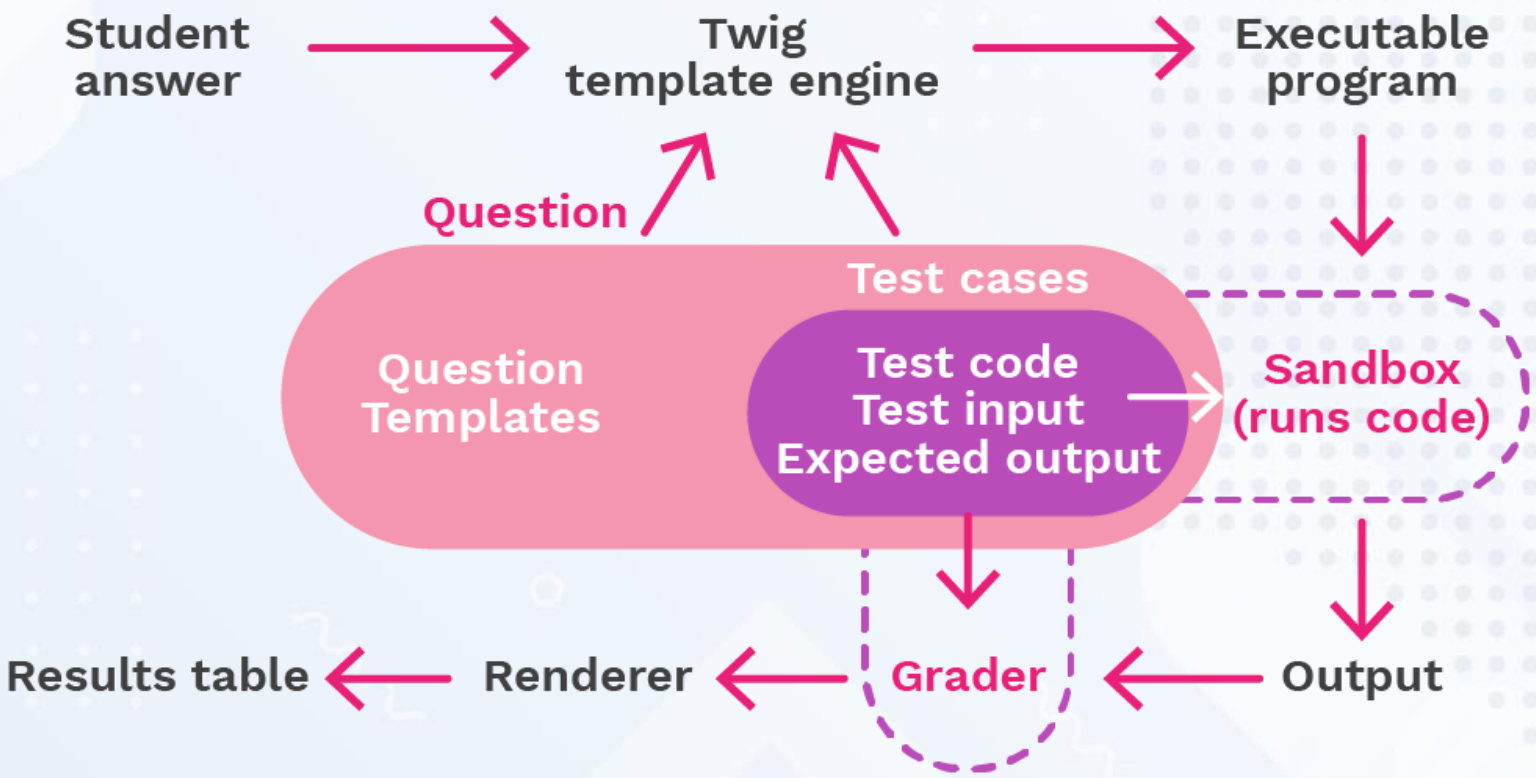
Para este ciclo, se entregará un archivo en formato **.db** con tablas previamente creadas y la configuración del driver de conexión específico en las dependencias del proyecto Java Maven, el cual se deberá abrir en el entorno de desarrollo integrado IDE Apache Netbeans 12.3
XML para agregar al archivo pom.xml del proyecto maven

```
<dependencies>
        <dependency>
            <groupId>org.xerial</groupId>
            <artifactId>sqlite-jdbc</artifactId>
            <version>3.8.7</version>
        </dependency>
  </dependencies>
```



5.5. Pruebas unitarias

Una prueba unitaria, es una forma de comprobar el funcionamiento específico de una unidad de código. y para nuestro ejercicio práctico en este ciclo, apoyaremos esta tarea mediante la herramienta *coderunner* dispuesta en el aula virtual de aprendizaje.



Material de lectura complementario

https://www.sqlite.org/datatype3.html
https://www.oracle.com/co/database/what-is-a-relational-database/
https://www.w3schools.com/sql/default.asp