GENERALITAT VALENCIANA
Conselleria d'Educació, Universitats i Ocupació
Institut Educació Secundària El Caminàs
Unió Europea
Fons Social Europeu
El FSE inverteix en el teu futur

Interfaces gráficas Java - SWING

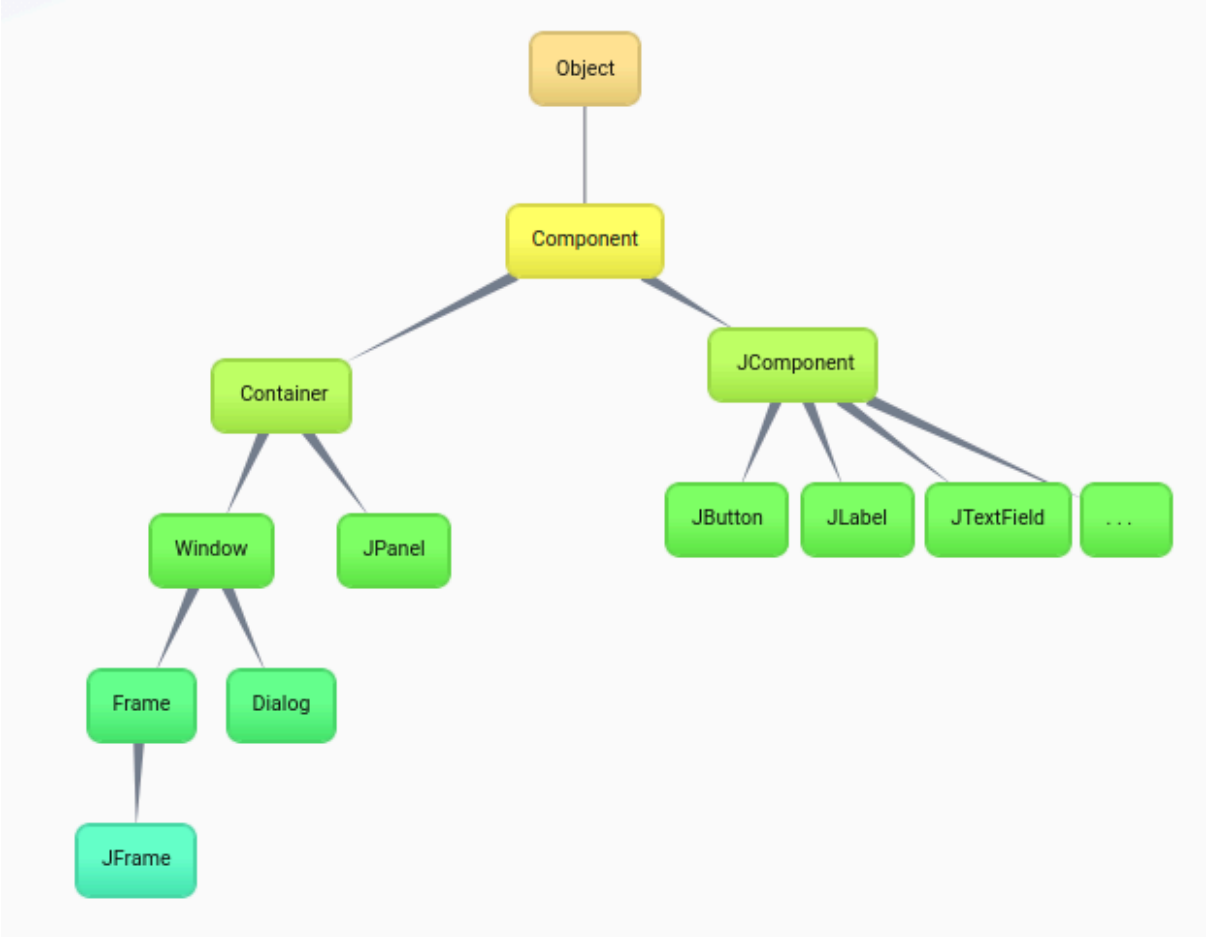
ÍNDICE

1. Introducción

Swing es una biblioteca de Java para crear interfaces gráficas de usuario (GUI). Forma parte de **Javax.swing** y permite crear ventanas, botones, etiquetas, cuadros de texto y más. Algunas de sus características son:

- Es parte del **JDK** (no requiere instalación extra).
- Está basado en **AWT (Abstract Window Toolkit)**, pero más flexible ya que al dibujar sus propios elementos gráficos es **independiente del sistema operativo**.
- Usa el concepto de **contenedores y componentes**.

Para entender su funcionamiento, es necesario tener una visión global de las clases de las que desciende, esta es la estructura general:



- **`Object`**: Clase base de todos los objetos en Java.
- **`Component`** (java.awt.Component): Todos los elementos gráficos en Swing derivan de esta clase.
- **`Container`** (java.awt.Container): Puede contener otros componentes (`JPanel`, `JFrame`, etc.).
- **`Window`** (java.awt.Window): Representa una ventana sin bordes (`Frame`, `Dialog`).
- **`Frame`** (java.awt.Frame): Ventana principal con barra de título.
- **`JFrame`** (javax.swing.JFrame): Ventana principal en Swing (Versión mejorada de `Frame`).
- **`Dialog`** (java.awt.Dialog)
- **`JPanel`** (javax.swing.JPanel): Va dentro de un `JFrame` para organizar elementos.
- **`JComponent`** (javax.swing.JComponent): Base de componentes interactivos:
    1. **JLabel** (javax.swing.JLabel): Etiqueta de texto
    2. **JButton** (javax.swing.JButton): Botón clickeable
    3. **JTextField** (javax.swing.JTextField): Campo de texto
    4. **JPasswordField** (javax.swing.JPasswordField): Campo de texto con caracteres ocultos.
    5. **JScrollPane** (javax.swing.JScrollPane): Panel con scroll
    6. **JTextArea** (javax.swing.JTextArea): Área de texto multilínea
    7. **JCheckBox** (javax.swing.JCheckBox): Casilla de verificación
    8. **JRadioButton** (javax.swing.JRadioButton): Botón de opción
    9. **JComboBox** (javax.swing.JComboBox): Menú desplegable
    10. **JTable** (javax.swing.JTable): Tabla de datos
    11. **JList** (javax.swing.JList): Lista de elementos
    12. **JMenuBar** (javax.swing.JMenuBar): Barra de menú
    13. **JToolBar** (javax.swing.JToolBar): Barra de herramientas
    14. **JCalendar** (de la biblioteca JCalendar) no incluida en Swing pero se puede agregar como una librería externa: Selección de fechas de manera visual mediante un calendario.

2. Ventanas con Swing (JFrame)

Para crear una ventana con Swing se necesita crear una clase hija de la clase `JFrame` y luego un objeto de la clase que se ha creado. En el constructor de la clase es importante:

- Indicar el tamaño que tendrá nuestra ventana.
- Indicar la acción a realizar cuando se cierre.
- Cambiar su estado a visible.

Un ejemplo podría ser el siguiente:

Se crea la clase Ventana como hija de JFrame

```
import javax.swing.*;

class Ventana extends JFrame {
   public Ventana(){
       this.setSize(600,300);
       this.setVisible(true);
       this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```

En el constructor:

- Se define su tamaño con `setSize(ancho, alto)`.
- Se usa `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)` para cerrar la aplicación al cerrar la ventana.
- Se hace visible con `setVisible(true)`.

En el main se instancia para que se cree la ventana:

```
public class Main {
   public static void main(String[] args) {
       Ventana ventana=new Ventana();
   }
}
```

Otros métodos importantes de JFrame:

- setLocation y setBounds (se heredan de la clase Component)

```
setLocation(200,200); //Indicar posición en pantalla
setBounds(200, 200, 800, 600); //posición y tamaño
```

- setTitle (se hereda de Frame)

```
setTitle("Título de la ventana");
```

- setResizable (pertenece a JFrame). Permite habilitar o deshabilitar el redimensionamiento de la ventana por el usuario

```
setResizable(false); //no se permite redimensionar
```

- setExtendedState (pertenece a la clase JFrame). Permite cambiar el estado de la ventana.

```
setExtendedState(Frame.MAXIMIZED_BOTH); //ventana maximizada
```

Otra clase muy útil es la clase `Toolkit` que es una clase de `java.awt` que proporciona herramientas para manejar recursos gráficos. Por tanto para utilizarla hay que importar la clase `java.awt`

```
import java.awt.*;
```

Para poder cargar imágenes y otros recursos se debe llamar a `getDefaultToolkit()` que devuelve una instancia del `Toolkit` del sistema. Algunos ejemplos de su utilizad podrían ser:

- Detectar la resolución del monitor donde se está ejecutando la aplicación y, por ejemplo, centrar nuestra ventana:

```
Toolkit mipantalla= Toolkit.getDefaultToolkit();
Dimension dimension = mipantalla.getScreenSize();
this.setSize(dimension.width/2, dimension.height/2);
this.setLocation(dimension.width/4, dimension.height/4);
```

- Cambiar el icono de la esquina superior izquierda de nuestra ventana

```
Toolkit mipantalla= Toolkit.getDefaultToolkit();
Image icono = mipantalla.getImage("cherry.png");
this.setIconImage(icono); //poner icono a la ventana
```

3. Paneles con Swing (JPanel)

Los `JPanel` son contenedores que se añaden a un `JFrame` y que permiten organizar componentes (etiquetas, botones, etc). Para utilizarlos, primero se añaden los componentes al JPanel y luego se añade el JPanel al JFrame. A continuación se muestra un ejemplo en el que primero se crea un `JPanel` donde se añaden los componentes (`JLabel`) y luego, el `JPanel` se agrega al `JFrame`:

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

class VentanaPrincipal extends JFrame {

   public VentanaPrincipal(){
     this.setTitle("Título de la ventana");
     this.setResizable(false);

     Toolkit mipantalla= Toolkit.getDefaultToolkit();
     Image icono = mipantalla.getImage("cherry.png");
     this.setIconImage(icono); //poner icono a la ventana

     Dimension dimension = mipantalla.getScreenSize();
     this.setSize(dimension.width/2, dimension.height/2);
     this.setLocation(dimension.width/4, dimension.height/4);

     JPanel panel = new JPanel(); // Crear un panel
     panel.setBackground(Color.RED);
     JLabel label = new JLabel("etiqueta 1"); // Crea etiqueta
     JLabel label2 = new JLabel("etiqueta 2");
     panel.add(label); // Agregar la etiqueta al panel
     panel.add(label2); // Agregar la etiqueta al panel

     this.add(panel); // Agregar el panel al JFrame
     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     this.setVisible(true); //hacer visible la ventana
   }
}
```

4. Componentes

4.1. JLabel

Se ha visto en el ejemplo anterior que sirven para mostrar texto en un JPanel.

4.2. JButton

Los botones son componentes con los que poder interactuar. Para ello se utiliza un `ActionListener`, que permite detectar eventos como clics, y para utilizarlos hay que importar las clases siguientes:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

A continuación se muestra un fragmento de código en el que se crea un `JButton` y un `JLabel` y se usa `addActionListener()` para detectar el clic y cambiar el texto del `JLabel`.

```
JPanel panel = new JPanel(); //Crear panel
JLabel label = new JLabel("Esperando ..."); //Crear etiqueta
JButton button = new JButton("Haz clic aquí");

// Evento del botón
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        label.setText("¡Botón presionado!");
    }
});
panel.add(label);
panel.add(button);
```

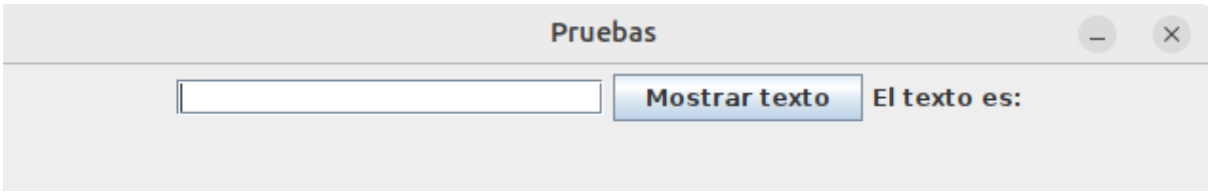
4.3. JTextField

Los **cuadros de texto (`JTextField`)** son útiles cuando queremos que el usuario introduzca datos. En el siguiente ejemplo se pide al usuario que introduzca un texto y que luego haga clic en un botón que cambia el contenido de una etiqueta.

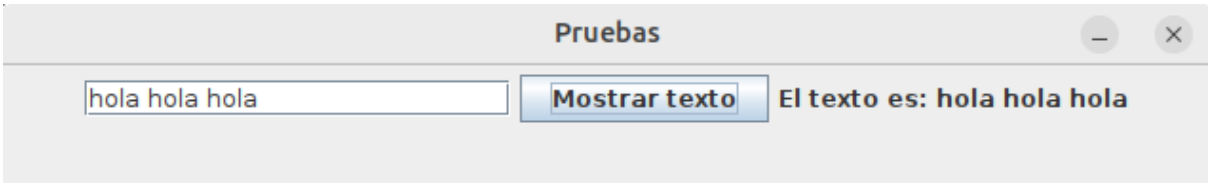
```
JTextField textField = new JTextField(20); // Campo de texto
JButton button = new JButton("Mostrar texto");
JLabel label = new JLabel("El texto es: ");

button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        label.setText("El texto es: " + textField.getText());
    }
});

panel.add(textField);
panel.add(button);
panel.add(label);
```

El método `getText()` obtiene el texto del textField y `getText()` lo modifica.

Antes de hacer clic en el botón:



Al introducir texto en el campo de texto y hacer clic en el botón, el resultado es:



4.4. JPasswordField

Funciona exactamente igual que *JTextField*, pero reemplaza el texto por un carácteres de ocultación (por ejemplo *). Es útil para pedir contraseñas al usuario.

4.5. JScrollPane

Permite agregar barras de desplazamiento a componentes que pueden contener más contenido del que se muestra en pantalla. Es especialmente útil para componentes como `JTextArea`, `JTable`, `JList` y `JPanel` cuando el contenido es más grande que la ventana. Se utiliza de la siguiente forma:

```
JScrollPane scrollPane = new JScrollPane(miComponente);
```

Por defecto se mostrarán barras de desplazamiento si el contenido excede el tamaño visible. Para controlar cuándo aparecen las barras se utiliza `ALWAYS`, `NEVER`, `AS_NEEDED` a continuación tienes unos ejemplos:

```
scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS); // Siempre visible
scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);  // Nunca visible
scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED); // Solo cuando sea necesario

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS); // Siempre visible
scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_NEVER);  // Nunca visible
scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED); // Solo cuando sea necesario
```

Por defecto, `AS_NEEDED` está activado, lo que significa que las barras aparecen solo si el contenido no cabe.

Iremos viendo su utilización a medida que veamos los componentes `JTextArea`, `JTable`, `JList` y `JPanel`

4.6. JTextArea

Permite la edición y visualización de múltiples líneas de texto. Puede configurarse como editable o de solo lectura.

Algunos de sus métodos son:

```
textArea.setEditable(false);  // Solo lectura
textArea.append("Texto adicional\n");  // Añade contenido
textArea.setText("Nuevo contenido");  // Reemplaza contenido
```

Se puede modificar su apariencia con estos métodos:

```
textArea.setFont(new Font("Arial", Font.BOLD, 14));  // Fuente
textArea.setForeground(Color.BLUE);  // Color texto
textArea.setBackground(Color.LIGHT_GRAY); // Color de fondo
```

Un ejemplo podría ser:

```
JTextArea txtDescrip = new JTextArea(8, 30);  // 8 filas, 30 columnas
txtDescrip.setLineWrap(true);  // Ajuste de línea automático
txtDescrip.setWrapStyleWord(true);  // Romper palabras completas
txtDescrip.setPreferredSize(new Dimension(300, 200)); // Ancho y alto fijos
```

4.7. JCheckBox

Permite a los usuarios seleccionar o deseleccionar una opción a través de una casilla de verificación. Varias casillas de verificación pueden estar seleccionadas al mismo tiempo. Para crear una casilla hay que indicar el texto que aparecerá junto a ella y si aparecerá seleccionada o no.

```
// No aparecerá seleccionada
JCheckBox checkBox1 = new JCheckBox("Opción 1");
// Aparecerá seleccionada
JCheckBox checkBox2 = new JCheckBox("Opción 2", true);
```

En este caso la primera casilla tendrá el texto *Opción 1* y no estará seleccionada y la segunda tendrá el texto *Opción 2* y aparecerá seleccionada. Para saber si una casilla está seleccionada o no, se utiliza el método ***isSelected()***

```
if (checkBox.isSelected()) {
   etiqueta.setText("Estado: Seleccionado");
} else {
   etiqueta.setText("Estado: No seleccionado");
}
```

Para detectar cambios en la selección se puede utilizar ActionListener

```
checkBox.addActionListener(e -> {
    if (checkBox.isSelected()) {
        System.out.println("La casilla se ha marcado");
    } else {
        System.out.println("La casilla se ha desmarcado");
    }
});
```

Se puede modificar su apariencia con estos métodos:

```
checkBox.setFont(new Font("Arial", Font.BOLD, 14));    // Fuente
checkBox.setForeground(Color.BLUE);  // Color texto
checkBox.setBackground(Color.LIGHT_GRAY);  // Color de fondo
checkBox.setSelected(true);  // Marca la casilla por defecto
```

Varias casillas de verificación pueden estar seleccionadas al mismo tiempo. Un ejemplo sencillo con múltiples JCheckBox podría ser:

```
JCheckBox opcion1 = new JCheckBox("Manzana");
JCheckBox opcion2 = new JCheckBox("Pera");
JCheckBox opcion3 = new JCheckBox("Kiwi");

JButton boton = new JButton("Confirmar selección");
JLabel resultado = new JLabel("Selecciona las frutas que te
gustan");

boton.addActionListener(e -> {
    String seleccion = "Seleccionaste: ";
    if (opcion1.isSelected()) seleccion += "Manzana ";
    if (opcion2.isSelected()) seleccion += "Pera ";
    if (opcion3.isSelected()) seleccion += "Kiwi ";
    resultado.setText(seleccion);
});
```

Si lo que se necesita es que solamente sea posible seleccionar una de las opciones, el componente que se debe utilizar es el `JRadioButton`

4.8. JRadioButton

Permite a los usuarios seleccionar solo una opción de un grupo de botones de radio. Para que solo un `JRadioButton` pueda estar seleccionado a la vez, todos los JRadioButton se deben agrupar en un `ButtonGroup` así, al seleccionar un botón, el otro se deselecciona automáticamente. A continuación se muestra un ejemplo:

```
JRadioButton radio1 = new JRadioButton("Opción 1");
JRadioButton radio2 = new JRadioButton("Opción 2");
ButtonGroup grupo = new ButtonGroup();
grupo.add(radio1);
grupo.add(radio2);
```

Para saber si una opción está seleccionada o no, se utiliza el método *isSelected()*

```
if (opcion1.isSelected()) {
   etiqueta.setText("Seleccionaste: Opción 1");
}
```

Para detectar cambios en la selección se puede utilizar ActionListener

```
opcion1.addActionListener(e -> System.out.println("Op 1"));
opcion2.addActionListener(e -> System.out.println("Op 2"));
opcion3.addActionListener(e -> System.out.println("Op 3"));
```

Se puede modificar su apariencia con estos métodos:

```
opcion1.setFont(new Font("Arial", Font.BOLD, 14));  // Fuente
opcion1.setForeground(Color.BLUE);  // Cambia el color del texto
opcion1.setBackground(Color.LIGHT_GRAY);  // Cambia el fondo
opcion1.setSelected(true);  // Lo marca por defecto
```

Diferencias entre `JRadioButton` y `JCheckBox`: Si necesitas que el usuario pueda seleccionar múltiples opciones, usa `JCheckBox`. Si solo debe elegir una, usa `JRadioButton` con `ButtonGroup`.

Componente	Selección múltiple	Grupo requerido
`JCheckBox`	Sí	No
`JRadioButton`	No	Sí

4.9. JComboBox

Permite a los usuarios seleccionar un elemento de una lista desplegable. Algunos métodos útiles son:

- **addItem(Object item)**: Agrega un elemento a la lista.
- **removeItem(Object item)**: Elimina un elemento de la lista.
- **setSelectedItem(Object item)**: Establece un elemento como seleccionado.
- **getSelectedItem()**: Obtiene el elemento seleccionado.

Para el ejemplo de la aplicación de frutería podríamos utilizarlo para indicar el tipo de producto, pudiendo escoger entre los valores “Fruta” o “Verdura”.

Para crear el JComboBox se necesita el siguiente código:

```
JComboBox cmbTipo = new JComboBox();
cmbTipo.addItem("Fruta");
cmbTipo.addItem("Verdura");
```

También se podría haber inicializado a partir de un array de la siguiente manera:

```
String[] opciones = {"Fruta", "Verdura"};
JComboBox<String> cmbTipo = new JComboBox<>(opciones);
```

Para detectar cuándo un usuario cambia la selección, se usa un `ActionListener`:

```
cmbTipo.addActionListener(e -> {
   String seleccionado = (String) cmbTipo.getSelectedItem();
   System.out.println("Seleccionaste: " + seleccionado);
});
```

Para cargar la información en el JComboBox a partir de la información obtenida del registro que se está modificando:

```
cmbTipo.setSelectedItem(tablaDatos.getValueAt(fila, 4).toString());
```

Para leer el valor antes de guardar la información en la base de datos (se guarda en el atributo *tipo* de la clase):

```
tipo=(String)cmbTipo.getSelectedItem();
```

4.10. JTable

Permite mostrar y manipular datos en forma de tabla (filas y columnas). Obtiene sus datos a partir de un modelo (TableModel) que también se encarga de administrar la información definiendo la organización de los datos en la tabla e indicando si las celdas son editables o no. **`DefaultTableModel`** es el más fácil de utilizar. Para poder trabajar con este componente hay que importar la librería:

```
import javax.swing.table.DefaultTableModel;
```

Algunos métodos útiles son:

- Para que las celdas de la tabla no sean editables:
  ```
  tabla.setDefaultEditor(Object.class, null);
  ```
- Para obtener la fila seleccionada: `tabla.getSelectedRow();`
- Para obtener el dato de la columna 0 (en este caso es el ID que es de tipo int) de la fila seleccionada:
  ```
  int fila =tablaDatos.getSelectedRow();
  if (fila != -1) {
     int id = (int)tablaDatos.getValueAt(fila, 0);
  }
  ```

Un ejemplo básico podría ser:

```
JPanel panel = new JPanel(new FlowLayout());
DefaultTableModel modelo = new DefaultTableModel();
modelo.addColumn("ID");
modelo.addColumn("Nombre");
modelo.addColumn("Edad");

// Añadir filas
modelo.addRow(new Object[]{1, "Pol", 25});
modelo.addRow(new Object[]{2, "Eli", 30});

JTable tabla = new JTable(modelo);
panel.add(tabla);
```

El resultado en pantalla:

1	Pol	25
2	Eli	30

Ejemplo en que se pasan las columnas y los datos al `TableModel` y se utiliza un `JScrollPane` (desplazamiento cuando hay muchas filas):

```
JPanel panel = new JPanel(new FlowLayout());
// Crear los datos para la tabla
String[] columnas = {"ID", "Nombre", "Edad"};
Object[][] datos = {
        {1, "Ana", 25},
        {2, "Juan", 30},
        {3, "Luis", 22}
};

// Crear el modelo de tabla
DefaultTableModel modelo = new DefaultTableModel(datos, columnas);
JTable tabla = new JTable(modelo);

// Agregar la tabla a un scroll
JScrollPane scrollPane = new JScrollPane(tabla);
panel.add(scrollPane);
```

El resultado en pantalla es el siguiente:

ID	Nombre	Edad
1	Ana	25
2	Juan	30
3	Luis	22

Ejemplo de JTable que muestra datos de un `ResultSet`:

```
Connection conn = null;
JPanel panel = new JPanel(new FlowLayout());

String sql = "SELECT * FROM productos";
try {
   conn = BD.connect();
   Statement stmt = conn.createStatement();
   ResultSet rs = stmt.executeQuery(sql);

     DefaultTableModel modelo = new DefaultTableModel();//modelo
vacío
```

```
    // Obtener metadatos de las columnas
    ResultSetMetaData metaData = rs.getMetaData();
    int columnCount = metaData.getColumnCount();

    // Añadir nombres de columnas al modelo
    for (int i = 1; i <= columnCount; i++) {
        modelo.addColumn(metaData.getColumnName(i));
    }

    int totalFilas = 0;
    while (rs.next()) {
        Object[] fila = new Object[columnCount];
        for (int i = 1; i <= columnCount; i++) {
            fila[i - 1] = rs.getObject(i);
        }
        modelo.addRow(fila);
        totalFilas++;
    }
    JTable tabla = new JTable(modelo);
    tabla.setDefaultEditor(Object.class, null); // no editable
    JScrollPane scrollPane = new JScrollPane(tabla); // con scroll

    if (totalFilas != 0){
       panel.add(scrollPane);
    }
    else {
        JLabel lbl = new JLabel("No hay datos");
        panel.add(lbl);
    }

} catch (SQLException e) {
    JLabel lbl = new JLabel("Se ha producido un error");
    panel.add(lbl);

}
```

El resultado en pantalla:

id	nombre	precio	procedencia
2	manzanas	1.75	ESP
3	kiwis	3.45	ESP

4.11. JList

Permite mostrar una lista de elementos para que el usuario los seleccione. Puede admitir una selección única o múltiple. Se puede crear a partir de un array y a partir de Modelo de Datos con **`DefaultListModel`** (para agregar o eliminar elementos dinámicamente):

Ejemplo con inicialización a partir de un array:

- Se crea un **array de Strings** con las opciones de la lista y se crea un **`JList<String>`** con esos valores.
- **`setSelectionMode(ListSelectionModel.SINGLE_SELECTION)`** se utiliza para permitir solo una selección.
- Se coloca la lista dentro de un **`JScrollPane`** para permitir el **scroll** si hay muchos elementos.
- Se utiliza un botón para **mostrar la selección actual** en una `JLabel`.

```
// Crear la lista con opciones
String[] opciones = { "Manzanas", "Peras", "Kiwis" };
JList<String> lista = new JList<>(opciones);
lista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);   //   Solo   una
opción seleccionable

// Agregar la lista dentro de un JScrollPane
JScrollPane scrollPane = new JScrollPane(lista);
scrollPane.setPreferredSize(new Dimension(120, 80)); // Ajustar tamaño

// Botón para obtener la selección
JButton boton = new JButton("Seleccionar");
JLabel etiqueta = new JLabel("Selecciona una fruta");

// Acción del botón
boton.addActionListener(e -> {
   String seleccion = lista.getSelectedValue();//Obtener seleccionado
   if (seleccion != null) {
       etiqueta.setText("Seleccionaste: " + seleccion);
   } else {
       etiqueta.setText("No has seleccionado nada");
   }
});
```

Para permitir que el usuario seleccione varias opciones, puedes usar:
`lista.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);`

Para obtener todos los elementos seleccionados sería:

```
List<String> seleccionados = lista.getSelectedValuesList();
for (String item : seleccionados) {
   System.out.println("Seleccionaste: " + item);
}
```

Ejemplo con inicialización a partir de un Modelo de Datos con `DefaultListModel` (modificar la lista en tiempo de ejecución):

```
DefaultListModel<String> modelo = new DefaultListModel<>();
modelo.addElement("Elemento 1");
modelo.addElement("Elemento 2");

JList<String> lista = new JList<>(modelo);
// Agregar un nuevo elemento
modelo.addElement("Elemento 3");
// Eliminar un elemento
modelo.removeElement("Elemento 1");
```

Se pueden detectar cambios en la selección usando un **ListSelectionListener**:

```
lista.addListSelectionListener(e -> {
   if (!e.getValueIsAdjusting()) { // Evita eventos duplicados
       System.out.println("Seleccionaste: " +
lista.getSelectedValue());
   }
});
```

Se puede modificar su apariencia con estos métodos:

```
lista.setFont(new Font("Arial", Font.BOLD, 14)); // Cambia la
fuente
lista.setForeground(Color.BLUE); // Cambia el color del texto
lista.setBackground(Color.LIGHT_GRAY); // Cambia el fondo
```

Y establecer **tamaños de celda personalizados**:

```
lista.setFixedCellHeight(25);
lista.setFixedCellWidth(100);
```

4.12. JMenuBar

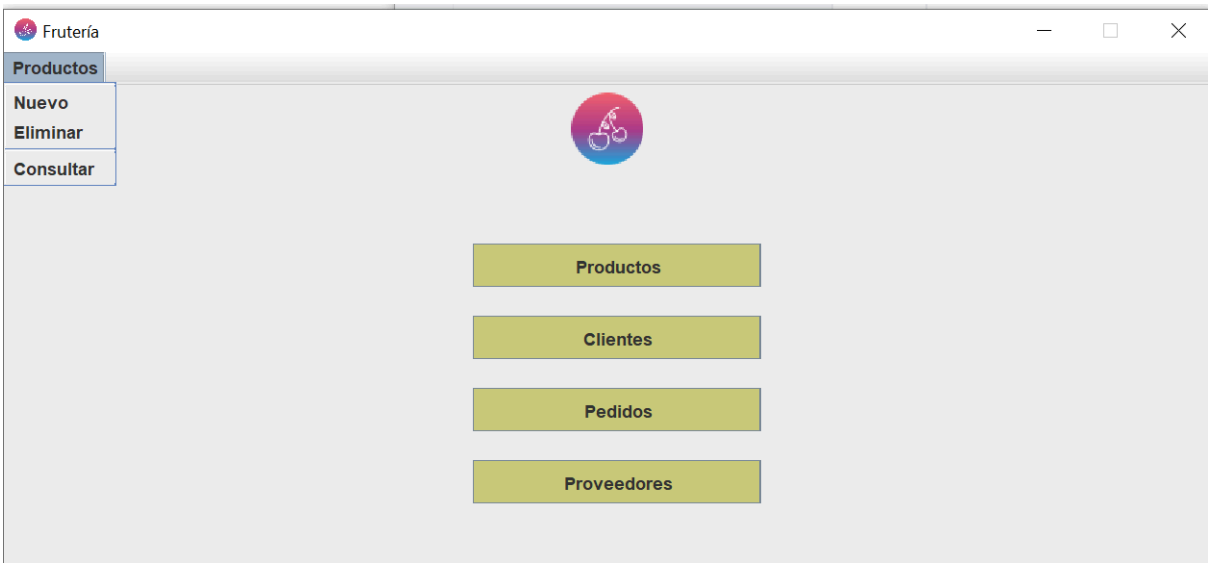
JMenuBar es un componente de Swing en Java que permite agregar menús a una ventana, como los típicos "Archivo", "Editar" y "Ayuda" en aplicaciones de escritorio. Un menú se compone de varios elementos:

1. **`JMenuBar`**: Es la barra de menús principal.
2. **`JMenu`**: Es un menú desplegable en la barra (`Archivo`, `Editar`, etc.).
3. **`JMenuItem`**: Son las opciones dentro de cada JMenu (`Abrir`, `Guardar`, `Salir`, etc.).
4. **`JCheckBoxMenuItem`**: Opción de menú con checkbox.
5. **`JRadioButtonMenuItem`**: Opción de menú con botones de radio.
6. **`JSeparator`**: Agrega una línea separadora entre elementos.

Un ejemplo podría ser:

```
// Crea la barra donde se agregarán los menús
JMenuBar menuBar = new JMenuBar();

JMenu menuProductos = new JMenu("Productos"); // Crea el menú

// Crea elementos del menú
JMenuItem nmProductosNuevo = new JMenuItem("Nuevo");
JMenuItem nmProductosConsultar = new JMenuItem("Consultar");

nmProductosNuevo.addActionListener(e -> nuevo()); // Agrega acción

// Agregar los elementos al menú
menuProductos.add(nmProductosNuevo);
menuProductos.add(nmProductosEliminar);
menuProductos.addSeparator(); // Línea separadora
menuProductos.add(nmProductosConsultar);

// Agregar el menú "Archivo" a la barra de menú
menuBar.add(menuProductos);

// Establecer la barra de menú en la ventana
this.setJMenuBar(menuBar);
```

El resultado es el siguiente:

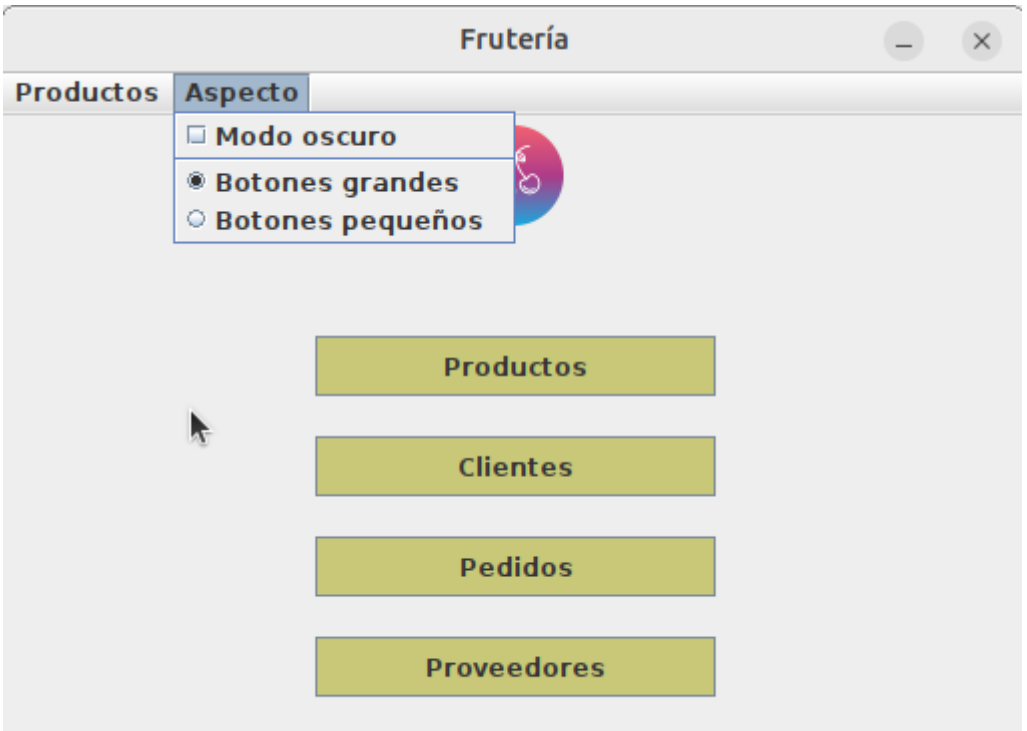


Ejemplo con JCheckBoxMenuItem y JRadioButtonMenuItem:

```
JMenuBar menuBar = new JMenuBar(); // Crear barra de menú
JMenu menuAspecto = new JMenu("Aspecto"); // Crear el menú

JCheckBoxMenuItem checkBox = new JCheckBoxMenuItem("Modo oscuro");

checkBox.addActionListener(e -> {
 if (checkBox.isSelected()){
      ...
 }
 });

JRadioButtonMenuItem radio1 = new JRadioButtonMenuItem("Botones
grandes");
radio1.setSelected(true);
radio1.addActionListener(e -> {
   if (radio1.isSelected()){
      ...
   }
});

JRadioButtonMenuItem radio2 = new JRadioButtonMenuItem("Botones
pequeños");
radio2.addActionListener(e -> {
```

```
    if (radio2.isSelected()){
            ...
    }

});

ButtonGroup grupo = new ButtonGroup(); // Grupo para los radio
grupo.add(radio1);
grupo.add(radio2);

// Añadir al menú los elementos
menuAspecto.add(checkBox);
menuAspecto.addSeparator();
menuAspecto.add(radio1);
menuAspecto.add(radio2);

menuBar.add(menuAspecto); // Añadir menú a la barra de menú
frame.setJMenuBar(menuBar); // Establecer barra menú en ventana
```

El resultado es el siguiente:



4.13. JToolBar

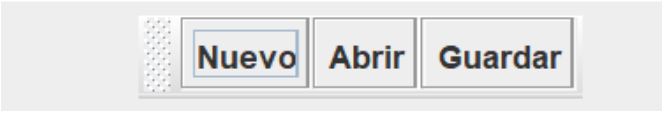
`JToolBar` es un componente de **Swing** en Java que proporciona una barra de herramientas. Se usa para **accesos rápidos a funciones** dentro de una aplicación, similar a las barras de herramientas en programas como Microsoft Word o Photoshop. Algunas de sus características son:

- Puede contener **botones, etiquetas, cajas de texto y otros componentes**.
- Se puede **mover y acoplar** en diferentes posiciones de la ventana.
- Puede estar **horizontal o vertical**.
- Se puede hacer **flotante o fija**.

Un ejemplo podría ser:

```
// Crear la barra de herramientas que contendrá los botones
JToolBar toolBar = new JToolBar();

// Crear botones
JButton btnNuevo = new JButton("Nuevo");
JButton btnAbrir = new JButton("Abrir");
JButton btnGuardar = new JButton("Guardar");
// Agregar botones a la barra
toolBar.add(btnNuevo);
toolBar.add(btnAbrir);
toolBar.add(btnGuardar);

// Acciones de los botones
btnNuevo.addActionListener(e -> nuevo());
btnAbrir.addActionListener(e -> abrir());
btnGuardar.addActionListener(e -> guardar());

// Agregar la barra de herramientas al panel
panel.add(toolBar);
```

El resultado es el siguiente:



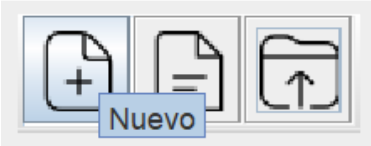
Algunos métodos útiles son:

- **toolBar.setFloatable(false)**: Para que no se pueda mover la barra de herramientas.
- **toolBar.setOrientation(JToolBar.VERTICAL)**: Para que la barra sea vertical.
- **toolBar.addSeparator()**: Añade un separador entre dos botones.

Otro ejemplo utilizando imágenes en los botones:

```
// Barra de herramientas no movible
JToolBar toolBar = new JToolBar();
toolBar.setFloatable(false);

// Botones con imagen y tooltip con el nombre de la acción
ImageIcon imgNuevo = new ImageIcon("ico_nuevo.png");
ImageIcon iconoNuevo = new
ImageIcon(imgNuevo.getImage().getScaledInstance(32, 32,
Image.SCALE_SMOOTH));
JButton btnNuevo = new JButton(iconoNuevo);
btnNuevo.setToolTipText("Nuevo");

ImageIcon imgAbrir = new ImageIcon("ico_abrir.png");
ImageIcon iconoAbrir = new
ImageIcon(imgAbrir.getImage().getScaledInstance(32, 32,
Image.SCALE_SMOOTH));
JButton btnAbrir = new JButton(iconoAbrir);
btnAbrir.setToolTipText("Abrir");

ImageIcon imgGuardar = new ImageIcon("ico_guardar.png");
ImageIcon iconoGuardar = new
ImageIcon(imgGuardar.getImage().getScaledInstance(32, 32,
Image.SCALE_SMOOTH));
JButton btnGuardar = new JButton(iconoGuardar);
btnGuardar.setToolTipText("Guardar");

toolBar.add(btnNuevo);
toolBar.add(btnAbrir);
toolBar.add(btnGuardar);
```

El resultado es el siguiente:



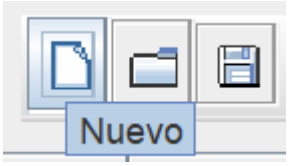
Los tamaños más comunes para íconos en barras de herramientas son:

- **16x16 px** (Muy pequeños, para menús compactos)
- **24x24 px** (Recomendado, buena visibilidad)
- **32x32 px** (Para interfaces más grandes)
- **48x48 px o más** (No recomendable, ocupa mucho espacio)

También se puede usar los iconos de Swing, por ejemplo:

```
btnNuevo.setIcon(UIManager.getIcon("FileView.fileIcon"));
btnAbrir.setIcon(UIManager.getIcon("FileView.directoryIcon"));
btnGuardar.setIcon(UIManager.getIcon("FileView.floppyDriveIcon"));
```

El resultado es el siguiente:



4.14. JCalendar

Permite al usuario seleccionar fechas de manera visual mediante un calendario. Es una clase que hereda de `JPanel`, lo que significa que es un panel gráfico que se puede agregar fácilmente a una interfaz gráfica en Java.

```
public class JCalendar extends JPanel
```

Es un componente de la biblioteca JCalendar pero al no estar incluida en Swing por defecto se debe agregar como una librería externa. Para ello sigue estos pasos:

- Ve a https://jarcasting.com/artifacts/com.toedter/jcalendar/1.4/
- Descarga el archivo `jcalendar-1.4.jar` y añádelo manualmente al classpath de tu proyecto. Indicando que lo quieres añadir como librería.
- `import com.toedter.calendar.JCalendar;`

Algunos métodos útiles son:

- **getDate()**: Devuelve la fecha seleccionada como Date.
- **setDate(Date date)**: Establece una fecha en el calendario.
- **getMonthChooser()**: Obtiene el selector de mes.
- **getYearChooser()**: Obtiene el selector de año.
- **getDayChooser()**: Obtiene el selector de día.

Se puede modificar su apariencia con estos métodos:

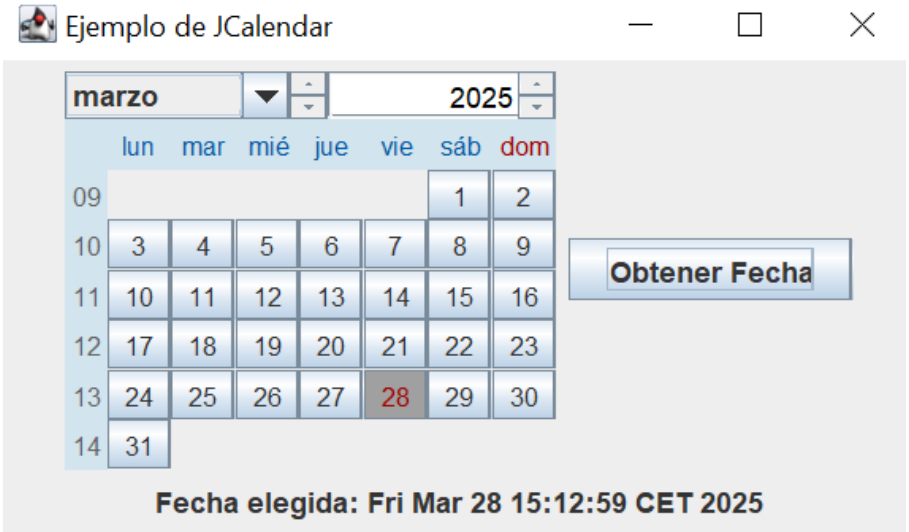
```
calendar.setForeground(Color.BLUE); // Cambia el color del texto
calendar.setBackground(Color.LIGHT_GRAY); // Cambia el fondo
calendar.setDate(new Date()); // Establece fecha actual por defecto
```

Ejemplo en el que al hacer clic en el botón obtener fecha cambia el texto de la etiqueta inferior:

```
import com.toedter.calendar.JCalendar;
import javax.swing.*;
import java.awt.*;
import java.util.Date;

public class JCalendarEjemplo {
   public static void main(String[] args) {
        // Crear ventana
        JFrame frame = new JFrame("Ejemplo de JCalendar");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        frame.setLayout(new FlowLayout());

        // Crear un JCalendar
        JCalendar calendar = new JCalendar();

        // Botón para obtener la fecha seleccionada
        JButton boton = new JButton("Obtener Fecha");
        JLabel etiqueta = new JLabel("Fecha seleccionada: ");

        // Acción del botón
        boton.addActionListener(e -> {
            Date fechaSeleccionada = calendar.getDate();
            etiqueta.setText("Fecha elegida: " +
fechaSeleccionada);
        });
```

```
        // Agregar componentes a la ventana
        frame.add(calendar);
        frame.add(boton);
        frame.add(etiqueta);
        frame.setVisible(true);
    }
}
```

El resultado es el siguiente:



5. Avisos y confirmaciones

5.1. JDialog

Es una ventana emergente para mostrar formularios o confirmaciones sin abrir una nueva ventana (`JFrame`) y también para preguntar al usuario. Puede ser **modal** (bloquea la ventana principal hasta que se cierre) o **no modal** (no bloquea). Por tanto siempre depende de un JFrame sobre el que aplica su efecto modal o no.

Ejemplo de `JDialog` modal con un botón:

```
JButton btnAbrirDialogo = new JButton("Abrir JDialog");
btnAbrirDialogo.addActionListener(new ActionListener() {
```

```
    @Override
    public void actionPerformed(ActionEvent e) {
        mostrarDialogo();
    }
});

private void mostrarDialogo() {
     /* Crear el JDialog con referencia a la ventana principal
(this), el tercer parámetro (true) indica que es modal */
    JDialog dialogo = new JDialog(this, "Diálogo Modal", true);
    dialogo.setSize(300, 150); // Tamaño del diálogo.
    dialogo.setLayout(new FlowLayout());

    JLabel etiqueta = new JLabel("Este es un JDialog modal.");
    JButton btnCerrar = new JButton("Cerrar");

    btnCerrar.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
             dialogo.dispose(); // cierra el JDialog
         }
    });

    dialogo.add(etiqueta);
    dialogo.add(btnCerrar);
    dialogo.setLocationRelativeTo(this); //Centra en ventana princip
    dialogo.setVisible(true);
}
```

El fragmento de código:

```
    btnCerrar.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
             dialogo.dispose(); // cierra el JDialog.
         }
    });
```

Utiliza una **clase anónima** que implementa `ActionListener` y se requiere escribir el método `actionPerformed()`. Se podría sustituir por:

```
    btnCerrar.addActionListener(e -> {dialogo.dispose();});
```

Este código es más corto ya que utiliza una expresión lambda, y en este caso funciona porque ActionListener es una interfaz funcional (tiene solo un método abstracto que es `actionPerformed`).

Para que el `JDialog` **no bloquee** la ventana principal, y ésta siga siendo interactiva aunque el `JDialog` esté abierto, hay que crear el `JDialog` como no modal:

```
JDialog dialogo = new JDialog(this, "Diálogo No Modal", false);
```

Se podría mejorar pasando dos String (título y mensaje) como parámetros al método `mostrarDialogo`. Así su código sería mucho más reutilizable.

```
private void mostrarDialogo(String titulo, String mensaje) {
    // Crear el JDialog con referencia a la ventana principal
    JDialog dialogo = new JDialog(this, titulo, true);
    dialogo.setSize(300, 150);
    dialogo.setLayout(new FlowLayout());

    JLabel etiqueta = new JLabel(mensaje);
    JButton btnCerrar = new JButton("Cerrar");

    btnCerrar.addActionListener(e -> { dialogo.dispose(); });

    dialogo.add(etiqueta);
    dialogo.add(btnCerrar);
        dialogo.setLocationRelativeTo(this);  //Centrar  en  ventana
princip
    dialogo.setVisible(true);
}
```

Ejemplo de JDialog con "Aceptar" y "Cancelar":

```
private void mostrarDialogo2() {
    // Crear el JDialog modal
    JDialog dialogo = new JDialog(this, "Confirmación", true);
    dialogo.setSize(300, 150);
    dialogo.setLayout(new FlowLayout());

    JLabel etiqueta = new JLabel("¿Deseas continuar?");
    JButton btnAceptar = new JButton("Aceptar");
    JButton btnCancelar = new JButton("Cancelar");
```

```
    // Acciones de los botones
    btnAceptar.addActionListener(e -> {
        mostrarDialogo("Diálogo", "Has aceptado");
        dialogo.dispose(); // Cierra el diálogo
    });

    btnCancelar.addActionListener(e -> {
        mostrarDialogo("Diálogo", "Has cancelado");
        dialogo.dispose(); // Cierra el diálogo
    });

    dialogo.add(etiqueta);
    dialogo.add(btnAceptar);
    dialogo.add(btnCancelar);

        dialogo.setLocationRelativeTo(this);  //Centra en ventana
principa
    dialogo.setVisible(true);
}
```

En este caso el diálogo 2 llama al diálogo visto anteriormente para mostrar un mensaje de la acción seleccionada por el usuario y luego cierra el mensaje, pero este planteamiento está hecho para realizar otro tipo de acciones, como seguir con confirmar el borrado de información, cambiar de ventana en un programa, etc.

5.2. JOptionPane

Para mostrar mensajes o pedir información al usuario de forma rápida y sencilla. El ejemplo anterior utilizando JOptionPane quedaría así:

```
// Acciones de los botones
btnAceptar.addActionListener(e -> {
    JOptionPane.showMessageDialog(dialogo, "Has aceptado.");
    dialogo.dispose();
});

btnCancelar.addActionListener(e -> {
    JOptionPane.showMessageDialog(dialogo, "Has cancelado.");
    dialogo.dispose(); // Cierra el diálogo
});
```

Ejemplo que utiliza el método showInputDialog() de JOptionPane para pedir al usuario información en un cuadro de diálogo utilizaremos:

```
private void mostrarDialogo3() {
    // Crear el JDialog modal
    JDialog dialogo = new JDialog(this, "Nombre", true);
    dialogo.setSize(300, 150);
    dialogo.setLayout(new FlowLayout());

    JLabel etiqueta = new JLabel("¿Quieres decir tu nombre?");
    JButton btnAceptar = new JButton("Sí");
    JButton btnCancelar = new JButton("No");

    // Acción del botón Aceptar
    btnAceptar.addActionListener(e -> {
        // Pedir el nombre con un InputDialog
        String nombre = JOptionPane.showInputDialog(dialogo, "Nombre?");
        // Verificar
        if (nombre != null && !nombre.trim().isEmpty()) {
            JOptionPane.showMessageDialog(dialogo, "Hola, " + nombre);
        } else {
            JOptionPane.showMessageDialog(dialogo, "Falta nombre");
        }
        dialogo.dispose();
    });

    // Acción del botón Cancelar
    btnCancelar.addActionListener(e -> {
        JOptionPane.showMessageDialog(dialogo, "Has cancelado.");
        dialogo.dispose();
    });

    dialogo.add(etiqueta);
    dialogo.add(btnAceptar);
    dialogo.add(btnCancelar);
    dialogo.setLocationRelativeTo(this); // Centra en ventana principal
    dialogo.setVisible(true);
}
```

6. Layout

Los **layouts** (`LayoutManager`) controlan cómo se organizan los componentes dentro de un contenedor (`JPanel`, `JFrame`, etc.). Swing dispone de varios Layouts para distribuir los componentes como mejor convenga en cada caso.

- FlowLayout: Componentes alineados en una fila (o varias si no caben). Utilizar para paneles pequeños y botones en línea.
- GridLayout: Distribuye componentes en una cuadrícula uniforme. Para tablas, teclados, tableros de juego.
- null: Posición fija, no adaptable. Utilizar solo cuando necesitas posiciones absolutas.
- BorderLayout: Distribuye en 5 zonas (Norte, Sur, Este, Oeste, Centro). Para dividir en zonas una ventana.
- BoxLayout: Organiza en una única fila o columna. Para formularios o listas verticales/horizontales.

A continuación se describe cada uno de estos Layouts:

6.1. FlowLayout (Diseño en Línea)

Es el que usa por defecto Swing (por lo tanto no es necesario indicarlo).

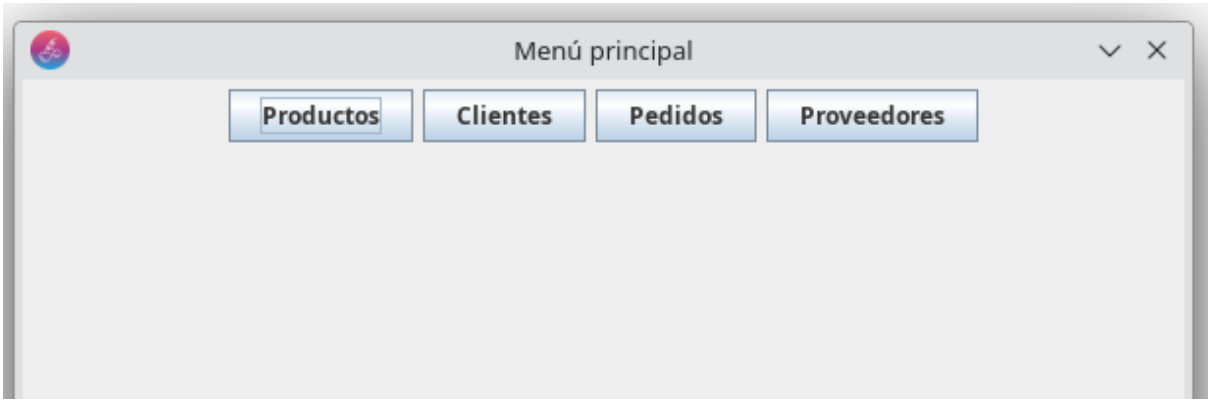
- **Coloca los componentes en una fila**, de izquierda a derecha.
- Cuando no hay más espacio, **baja a la siguiente línea**.
- Se puede alinear al **centro (por defecto), izquierda o derecha**.
- **Uso recomendado:** Para paneles pequeños o formularios con pocos elementos.

Por ejemplo, si creamos 4 botones en nuestro JPanes de la siguiente forma:

```
JButton btnProductos = new JButton("Productos");
btnProductos.addActionListener(e -> {
   Productos productos=new Productos();
});
panel.add(btnProductos);

JButton btnClientes = new JButton("Clientes");
```

```
btnClientes.addActionListener(e -> {
    System.out.println("\nClic en botón de clientes");
});
panel.add(btnClientes);

JButton btnPedidos = new JButton("Pedidos");
btnPedidos.addActionListener(e -> {
    System.out.println("\nClic en botón de pedidos");
});
panel.add(btnPedidos);

JButton btnProveedores = new JButton("Proveedores");
btnProveedores.addActionListener(e -> {
    System.out.println("\nClic en botón de proveedores");
});
panel.add(btnProveedores);
```

Se mostrarán así en pantalla:



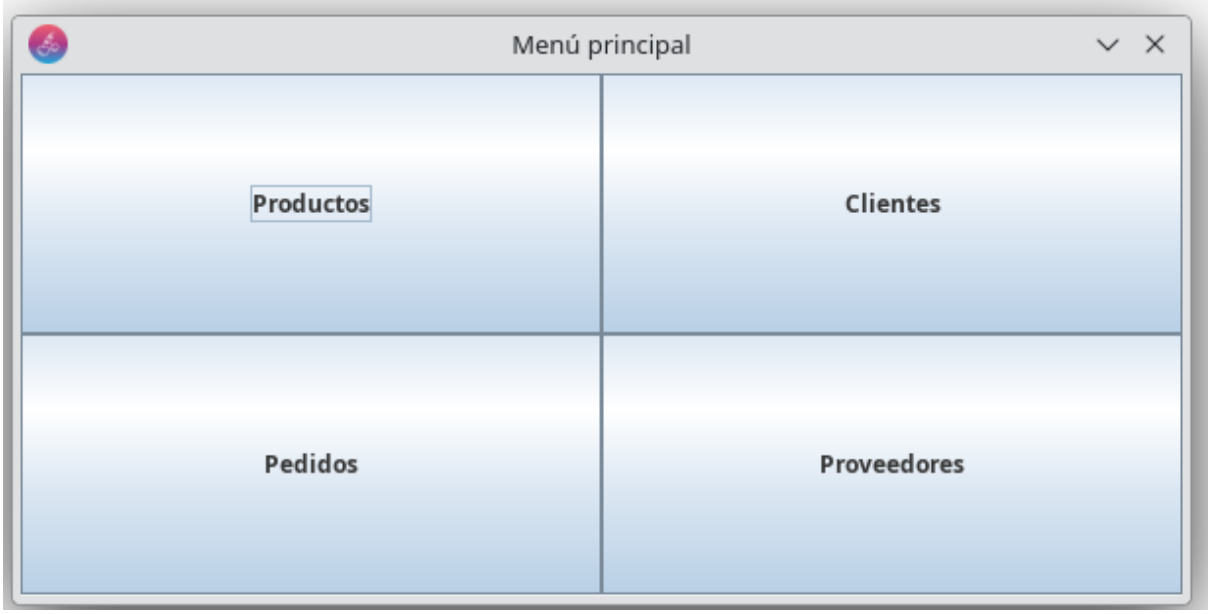
6.2. GridLayout (Diseño en Cuadrícula)

- **Organiza los componentes en una cuadrícula (filas y columnas).**
- Todos los componentes tienen **el mismo tamaño**.
- **Uso recomendado:** Para tableros, teclados, paneles organizados en filas y columnas.

Siguiendo con el ejemplo de los 4 botones anteriores, solamente cambiando el layout de la siguiente forma:

```
JPanel panel = new JPanel(); //Crear un panel
panel.setLayout(new GridLayout(2, 2));//2filas, 2columnas
```

Los botones quedarían distribuidos así:



6.3. null (Diseño Absoluto)

Este Layout no usa un gestor de diseño, por tanto:

- Hay que establecer manualmente tamaño y posición con `setBounds`.
- No se adapta automáticamente si la ventana cambia de tamaño. Por tanto, es útil cuando se necesita control total de la posición pero hay que evitarlo en interfaces dinámicas.

Siguiendo con nuestro ejemplo, podríamos posicionar nuestros 4 botones de la forma que quisiéramos, por ejemplo:

```
panel.setLayout(null);

btnProductos.setBounds(50, 50, 120, 30);//Coordenadas y tamaño
btnClientes.setBounds(60, 100, 120, 30);
btnPedidos.setBounds(70, 150, 120, 30);
btnProveedores.setBounds(80, 200, 120, 30);
```

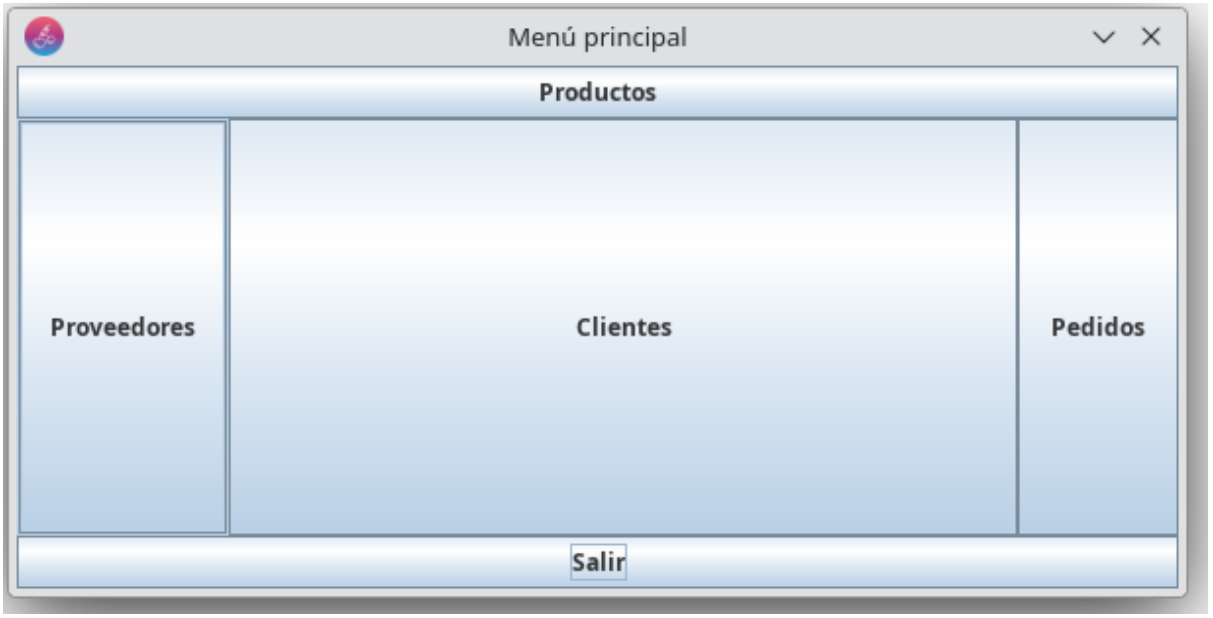
El resultado es el que se muestra en la captura de pantalla siguiente:

6.4. BorderLayout (Diseño en Zonas)

- Divide el contenedor en **5 zonas**: **NORTH (Arriba), SOUTH (Abajo), EAST (Derecha), WEST (Izquierda), CENTER (Centro).**
- **El centro ocupa todo el espacio disponible** si no se agregan componentes a los lados.
- Si no se especifica ninguna región, por defecto el componente se inserta en el centro del contenedor.
- **Uso recomendado:** Para estructuras principales de una interfaz gráfica (menús, barras de herramientas, áreas de trabajo).

Hemos añadido un nuevo botón llamado Salir en nuestro ejemplo y lo hemos programado de la siguiente forma:

```
panel.setLayout(new BorderLayout());

panel.add(btnProductos, BorderLayout.NORTH);
panel.add(btnClientes, BorderLayout.CENTER);
panel.add(btnPedidos, BorderLayout.EAST);
panel.add(btnProveedores, BorderLayout.WEST);
panel.add(btnSalir, BorderLayout.SOUTH);
```

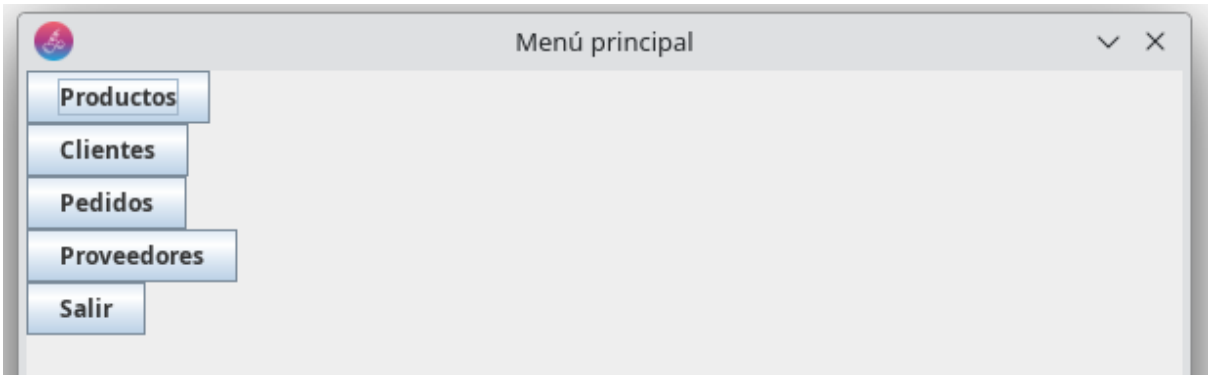


6.5. BoxLayout (Diseño en Línea o Columna)

- Permite organizar componentes en una sola fila o columna ya que se agrupan horizontal o verticalmente dentro del contenedor.
- Los componentes no se dividen en varias líneas como en `FlowLayout`
- Uso recomendado: Formularios o paneles con alineación específica.

Para distribuir nuestros botones de forma vertical, hay que indicarlo así en el código:

```
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
```



Para distribuir nuestros botones de forma horizontal, el código es:

```
panel.setLayout(new BoxLayout(panel, BoxLayout.X_AXIS));
```

6.6. Combinación de Layouts

Los layouts se pueden combinar para obtener la distribución más conveniente para la aplicación, en el siguiente ejemplo se crear una **ventana con tres secciones**:

1. **Encabezado** con el título en la parte superior (`FlowLayout`).
2. **Panel principal** con una cuadrícula con botones (`GridLayout`).
3. **Pie de página** en la parte inferior con un botón para cerrar (`FlowLayout`).

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

class Ventana extends JFrame {

   public Ventana(){
        this.setTitle("Título de la ventana");
        this.setResizable(false);
        this.setSize(600,300); //tamaño de la ventana
        this.setLocation(200,200); //ubicación en la pantalla

        setLayout(new BorderLayout()); // Layout principal
        this.add(panelSuperior(), BorderLayout.NORTH);
        this.add(panelCentral(), BorderLayout.CENTER);
        this.add(panelInferior(), BorderLayout.SOUTH);

        this.setVisible(true); //hacer visible la ventana
   }

   private JPanel panelSuperior() {
        // Panel superior (FlowLayout)
        JPanel panel = new JPanel(new FlowLayout());
        JLabel titulo = new JLabel("Ejemplo de varios Layouts");
        panel.add(titulo);
        return panel;
   }

   private JPanel panelCentral() {
        // Panel central (GridLayout)
        JPanel panel = new JPanel(new GridLayout(2, 2, 10, 10));
        panel.add(new JButton("Botón 1"));
        panel.add(new JButton("Botón 2"));
        panel.add(new JButton("Botón 3"));
        panel.add(new JButton("Botón 4"));
        return panel;
   }
```

```
    private JPanel panelInferior() {
        //  Panel inferior (FlowLayout)
        JPanel panel = new JPanel(new FlowLayout());
        JButton btnCerrar = new JButton("Cerrar");
        btnCerrar.addActionListener(e -> System.exit(0));
        panel.add(btnCerrar);
        return panel;
    }

}
```

7. CardLayout

Permite apilar múltiples paneles y **mostrar solo uno a la vez**, como si fueran cartas de una baraja. Se usa para formularios, asistentes, o pestañas personalizadas ya que es útil cuando necesitas cambiar entre diferentes vistas dentro de la misma ventana. Permite organizar varios paneles en una sola área, solo **uno** de los paneles es visible a la vez. Algunos de sus métodos son:

- **next(panel)**: Muestra el siguiente panel en la lista.
- **previous(panel)**: Muestra el panel anterior.
- **first(panel)**: Muestra el primer panel.
- **last(panel)**: Muestra el último panel.
- **show(panel, "nombre")**: Muestra el panel indicado en nombre.

Ejemplo con tres paneles y botones para navegar entre ellos:

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

public class EjemploCardLayout {
    public static void main(String[] args) {
        // Crear ventana
        JFrame frame = new JFrame("Ejemplo de CardLayout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        // Crear el contenedor con CardLayout
        JPanel contenedor = new JPanel(new CardLayout());

        // Crear los paneles
        JPanel panel1 = new JPanel();
        panel1.setBackground(Color.RED);
        panel1.add(new JLabel("Panel Rojo"));
        JPanel panel2 = new JPanel();
```

```
        panel2.setBackground(Color.GREEN);
        panel2.add(new JLabel("Panel Verde"));
        JPanel panel3 = new JPanel();
        panel3.setBackground(Color.BLUE);
        panel3.add(new JLabel("Panel Azul"));

        // Agregar los paneles al contenedor con nombres
        contenedor.add(panel1, "rojo");
        contenedor.add(panel2, "verde");
        contenedor.add(panel3, "azul");

        // Crear botones de navegación
        JButton btnSiguiente = new JButton("Siguiente");
        JButton btnAnterior = new JButton("Anterior");
        JButton btnRojo = new JButton("Rojo");
        JButton btnVerde = new JButton("Verde");
        JButton btnAzul = new JButton("Azul");

        // Obtener el layout para controlarlo
        CardLayout cardLayout = (CardLayout) contenedor.getLayout();

        // Acciones botones
        btnSiguiente.addActionListener(e -> cardLayout.next(contenedor));
        btnAnterior.addActionListener(e -> cardLayout.previous(contenedor));
        btnRojo.addActionListener(e -> cardLayout.show(contenedor, "rojo"));
        btnVerde.addActionListener(e -> cardLayout.show(contenedor, "verde"));
        btnAzul.addActionListener(e -> cardLayout.show(contenedor, "azul"));

        // Panel para los botones
        JPanel panelBotones = new JPanel();
        panelBotones.add(btnAnterior);
        panelBotones.add(btnSiguiente);
        panelBotones.add(btnRojo);
        panelBotones.add(btnVerde);
        panelBotones.add(btnAzul);

        // Agregar los componentes a la ventana
        frame.setLayout(new BorderLayout());
        frame.add(contenedor, BorderLayout.CENTER);
        frame.add(panelBotones, BorderLayout.SOUTH);
        frame.setVisible(true);      // Mostrar la ventana
    }
}
```