GUIs CON JAVA SWING

Contenedores

Cualquier aplicación, con interfaz gráfica típica, comienza con la apertura de una ventana principal, que suele contener la barra de título, los botones de minimizar, maximizar/restaurar y cerrar, y unos bordes que delimitan su tamaño.

Esa ventana constituye un marco dentro del cual se van colocando el resto de los componentes que necesita el programador: menús, barras de herramientas, botones, casillas de verificación, cuadros de texto, etc...

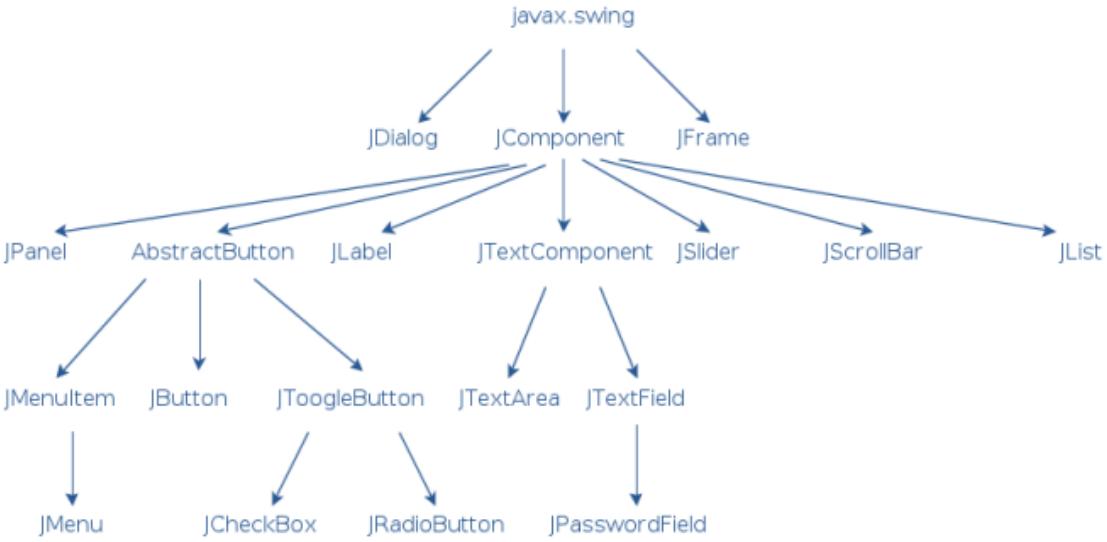
Esa ventana principal o marco sería el contenedor de alto nivel de la aplicación. Toda aplicación de interfaz gráfica de usuario Java tiene, al menos, un contenedor de alto nivel.

JFrame	Implementa una ventana. Las ventanas principales de una aplicación deberían de ser un JFrame.
JDialog	Implementa una ventana de tipo diálogo. Es te tipo de ventanas se utilizan como ventanas secundarias y generalmente son llamadas por ventanas padre del tipo JFrame.
JApplet	Un *applet* es una aplicación Java que se ejecuta dentro de un navegador web en la máquina del cliente.

La componente básica que requerimos cada vez que implementamos una interfaz visual con la libraría Swing es la clase JFrame. Esta clase encapsula una ventana clásica de cualquier sistema operativo con entorno gráfico (Windows, OS X, Linux etc.).

Esta clase se encuentra en el paquete javax.swing

Podemos hacer una aplicación mínima con la clase JFrame: lo más correcto es plantear una clase que herede de la clase JFrame y extienda sus responsabilidades agregando botones, etiquetas, editores de línea etc.



Creación de una ventana básica

1. Crear una clase que herede de JFrame.

```
public class Ventana extends JFrame {}
```

2. Darle un tamaño y una ubicación a la ventana usando los siguientes métodos (varias opciones):

a)
```
setSize(ancho, alto);
setLocation(x, y);
```
//establece el tamaño en pixeles.
//establece la ubicación en pantalla por
//medio de coordenadas.

b)
```
setBounds(x, y, ancho, alto);
```
//las dos cosas a la vez.

c)
```
setSize(ancho, alto);
setLocationRelativeTo(null);
```
// ventana en el centro de la
// pantalla.

Se puede llamar a estos métodos desde el constructor de la clase (más claro y organizado) o desde la función que está creando la ventana (por ejemplo el main()).

3. Decidir qué hacer cuando se cierra la ventana:

```
setDefaultCloseOperation(opcion_de_cierre);
```

4. Hacer visible la ventana (siempre la última sentencia):

```
setVisible(true);
```

Ejemplo

```
public class Ventana extends JFrame {

	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(300, 300);
		setLocationRelativeTo(null);
		setVisible(true);
	  }
}
```

```
public class Main {

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

Configurar la ventana

- no poder redimensionar

  `setResizable(false);`

- pantalla completa

  `setExtendedState(Frame.MAXIMIZED_BOTH);`

- establecer un título a la ventana

  `setTitle("titulo");`

- cambiar el icono

  `setIconImage(Toolkit.getDefaultToolkit().getImage("icono.jpg"));`

Componentes Swing

Los componentes son los elementos básicos de la programación con Swing. Todo lo que se ve en un GUI de Java es un componente. Los componentes se colocan en otros elementos llamados **contenedores** que sirven para agrupar componentes.

Pasos a seguir para agregar componentes a un contenedor:

1. Declarar los componentes necesarios como atributos de la clase,
2. desde el constructor crear los componentes,
3. ubicarlos llamando al método setBounds() (si no se usa un organizador),
4. añadirlos al contenedor llamando al método add().

Componente	Método	Significado
JFrame	add(componente)	Añade el componente al contenedor.
	setTitle(cadena)	Muestra un String en el título de la ventana.
	setLayout(organizador)	Establece el tipo de organizador de componentes.
	setResizable(bool)	Establece si la ventana podrá ser redimensionada.
	setJMenuBar(JMenuBar)	Establece una barra de menús.
	setDefaultCloseOperation();	Operación cuando se cierra la ventana: EXIT_ON_CLOSE DO_NOTHING_ON_CLOSE HIDE_ON_CLOSE DISPOSE_ON_CLOSE
	dispose()	Cerrar ventana.
JLabel	JLabel(cadena)	Constructor.
	setText(cadena)	Establece el texto de la etiqueta.
JButton	JButton(cadena)	Constructor.
	JButton(new ImageIcon("exit.PNG"))	Constructor con imagen.
	addActionListener()	Asocia la interface que capturará el evento de hacer clic en el botón.
JTextField	JTextField()	Constructor.
	getText()	Extrae el contenido del control.
	setFont(Font)	Establece las características del texto (tipo de letra, tamaño, …) a través de un objeto de la clase Font.
JTextArea	JTextArea()	Constructor.
	getText()	Extrae el contenido del control.
	append(cadena)	Agrega la cadena al texto del área de texto
JPasswordField	setEchoChar('char')	Indica el carácter de máscara.
	getPassword()	Recupera el password introducido.
JComboBox	JComboBox()	Constructor.
	addItem(cadena)	Añade elementos a la lista.
	getSelectedItem().toString()	Extrae el elemento seleccionado de la lista. El método toString es opcional.

	getSelectedIndex()	Devuelve el índice del elemento seleccionado (empezando por cero).
	addItemListener()	Asocia la interface que capturará el evento de cambio de elemento.
JCheckBox	JCheckBox(cadena)	Constructor.
	isSelected()	Devuelve el valor lógico correspondiente si la casilla está o no seleccionada.
	setSelected(boolean)	Selecciona la casilla.
	addChangeListener()	Asocia la interface que capturará el evento de cambio de estado en la casilla.
JRadioButton	JRadioButton(cadena)	Constructor.
	addChangeListener()	Asocia la interface que capturará el evento de cambio de estado en botón de radio.
	isSelected()	Devuelve el valor lógico correspondiente si el botón de radio está o no seleccionado.
	setSelected(boolean)	Selecciona el botón de radio.
ButtonGroup	ButtonGroup()	Constructor.
	add(JRadioButton)	Añade botones de radio a un grupo.
JMenuBar	JMenuBar()	Constructor.
	add(JMenu)	Añade un menú a una barra de menús.
JMenu	JMenu(cadena)	Constructor.
	add(JMenuItem)	Añade un elemento de menú a un menú.
	addSeparator()	Añade un separador (línea horizontal) a un menú.
JMenuItem	JMenuItem(cadena)	Constructor.
	JMenuItem(cadena, new ImageIcon("archivo.png"))	Constructor con texto e imagen.
	addActionListener(menuItem)	Asocia la interface que capturará el evento de hacer clic en un elemento de menú.

métodos de apariencia y posición

método	uso
void setVisible(boolean vis**)**	Muestra u oculta el componente según el valor del argumento sea **true** o **false**
Color getForeground()	Devuelve el color de frente en forma de objeto Color
void setForeGround(Color color**)**	Cambia el color frontal
Color getBackground()	Devuelve el color de fondo en forma de objeto *java.awt.Color*
void setBackground(Color color**)**	Cambia el color de fondo
Point getLocation()	Devuelve la posición del componente en forma de objeto Point
void setLocation(int x, **int** y**)**	Coloca el componente en la posición x, y
void setLocation(Point p**)**	Coloca el componente en la posición marcada por las coordenadas del punto P
Dimension getSize()	Devuelve el tamaño del componente en un objeto de tipo *java.awt.Dimension*.
void setSize(Dimension d**)**	
void setSize(int ancho, **int** alto**)**	Cambia las dimensiones del objeto en base a un objeto *Dimension* o indicando la anchura y la altura con dos enteros.
void setBounds(int x, **int** y, **int** ancho, **int** alto)	Determina la posición de la ventana (en la coordenada x, y) así como su tamaño con los parámetros *ancho* y *alto*
void setPreferredSize(Dimension d**)**	Cambia el tamaño preferido del componente. Este tamaño es el que el componente realmente quiere tener.
void setToolTipText(String texto**)**	Hace que el texto indicado aparezca cuando el usuario posa el cursor del ratón sobre el componente
String getToolTipText()	Obtiene el texto de ayuda del componente
Cursor getCursor()	Obtiene el cursor del componente en forma de objeto *java.awt.Cursor*
void setCursor(Cursor cursor**)**	Cambia el cursor del componente por el especificado en el parámetro.
void setFont(Font fuente**)**	Permite especificar el tipo de letra de la fuente del texto

activar y desactivar componentes

método	uso
void setEnabled(boolean activar**)**	Si el argumento es **true** se habilita el componente, si no, se deshabilita. Un componente deshabilitado es un método que actúa con el usuario. Por deshabilitar un componente, no se deshabilitan los hijos.

Colocar componentes en una ventana

a) Colocar los componentes de forma manual, sin organizador (layout):

1- Invalidar el organizador de controles por defecto (FlowLayout) para la ventana:

```
setLayout(null);
```

2- Crear el componente:

```
<Componente> miComponente = new Componente();
```

Los componentes son atributos de la clase contenedora, una ventana (JFrame) o un panel o lámina (JPanel). Se pueden declarar e instanciar en la misma sentencia o, una vez declarados como atributos, se crean en el constructor o en cualquier otro método.

3- Darle tamaño y posición. Esto es imprescindible si hemos anulado el organizador de componentes en el paso 1, por lo tanto, es el propio programador quién debe colocar los componentes en la ventana:

```
miComponente.setBounds(X, Y, ancho, alto);
```

4- Añadirlo al contenedor:

```
add(miComponente)
```

Ejemplo

```
public class Ventana extends JFrame {

        private JButton boton;

        public Ventana() {
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setSize(300, 300);
                setLocationRelativeTo(null);
            inicializarComponentes();
                setVisible(true);
          }

        private void inicializarComponentes() {
                boton = new JButton("boton");
        boton.setBounds(50, 50, 100, 40);
        add(boton);
    }
}
```

```
public class Main {

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

b) Colocar los componentes usando un organizador.

1- Establecer el organizador de controles que se desee para la ventana:

```
setLayout(new tipo_de_organizador(argumentos_necesarios));
```

2- Al añadir el componente, según el tipo de organizador elegido, habrá que especificar su ubicación:

```
add(componente,ubicación);
```

FLOWLAYOUT

1. Es la disposición por defecto (elementos centrados), no es necesaria establecerla:
```
setLayout(new FlowLayout());
```
2. Crear una disposición diferente a la de por defecto:
```
setLayout(new FlowLayout(FlowLayout.LEFT));
                                 .RIGHT));
```
3. Cambiar el espacio entre componentes:
```
setLayout(new FlowLayout(FlowLayout.CENTER,espacioH,espacioV));
```

BORDERLAYOUT

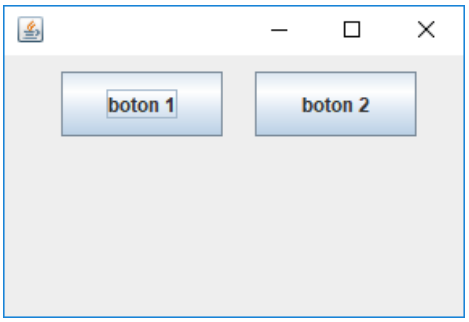
1. Establecer la disposición:
```
setLayout(new BorderLayout());
```
2. Añadir componentes en el área correspondiente:
```
add(componente, BorderLayout.NORTH);
                              .SOUTH);
                              .WEST);
                              .CENTER);
                              .EAST);
```

GRIDLAYOUT

1. Establecer la disposición:
```
setLayout(new GridLayout(filas,columnas));
```
2. Añadir componentes (se van colocando mientras se añaden, rellenando primero por filas).
```
add(componente);
```

Botones: JButton

Los componentes se pueden añadir directamente a la ventana (JFrame) o en una o varias lámina o paneles (JPanel) a modo de capas o cebollas. Si no se dice lo contrario, el organizador por defecto es FlowLayout.



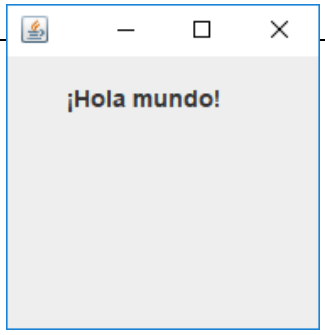

```
package vista;

import java.awt.Font;
import javax.swing.*;
import controlador.ManejadorEventos;

public class Ventana extends JFrame {

	private JButton boton1, boton2;

	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(300, 200);
		setLocationRelativeTo(null);
		setLayout(null);
		inicializarComponentes();
		setVisible(true);
	}

	private void inicializarComponentes() {
		boton1 = new JButton("boton 1");
		boton1.setBounds(35, 10, 100, 40);
		add(boton1);

		boton2 = new JButton("boton 2");
		boton2.setBounds(155, 10, 100, 40);
		add(boton2);

	}

	public void establecerManejador(ManejadorEventos manejador) {
		boton1.addActionListener(manejador);
		boton2.addActionListener(manejador);
	}

	public JButton getBoton1() {
		return boton1;
	}

	public JButton getBoton2() {
		return boton2;
	}

}
```

Gestión de eventos usando el **patrón MVC**
(se ven más adelante)

```java
package controlador;

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

import vista.Ventana;

public class ManejadorEventos implements ActionListener {

	private Ventana ventana;

	public ManejadorEventos(Ventana ventana) {
		this.ventana = ventana;
	}

	@Override
	public void actionPerformed(ActionEvent e) {

		if (e.getSource() == ventana.getBoton1()) {
			//acciones
		}

		if (e.getSource() == ventana.getBoton2()) {
			//acciones
		}
	}

}
```

```java
package controlador;

import vista.Ventana;

public class Main {

	public static void main(String[] args) {
		Ventana miVentana = new Ventana();
		ManejadorEventos manejador = new ManejadorEventos(miVentana);
		miVentana.establecerManejador(manejador);
	}
}
```

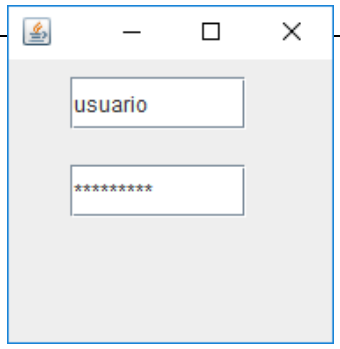
Etiquetas: JLabel




```
package vista;

import java.awt.Font;
import javax.swing.*;
import controlador.ManejadorEventos;

public class Ventana extends JFrame {

	private JLabel etiqueta;

	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(200, 200);
		setLocationRelativeTo(null);
		setLayout(null);
		inicializarComponentes();
		setVisible(true);
	}

	private void inicializarComponentes() {
		etiqueta = new JLabel("¡Hola mundo!");
		etiqueta.setFont(new Font("Arial", Font.BOLD, 14));
		etiqueta.setBounds(35, 10, 100, 30);
		add(etiqueta);

	}

}
```

```
package controlador;

import vista.Ventana;

public class Main {

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

Cajas de texto: JTextField y JPasswordField

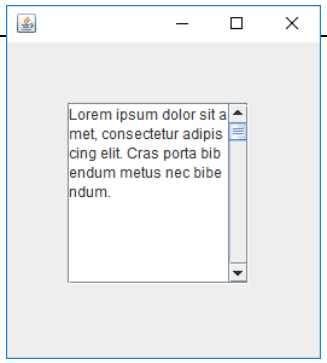



```
package vista;

import java.awt.Font;
import javax.swing.*;
import controlador.ManejadorEventos;

public class Ventana extends JFrame {

	JTextField cajaTexto;
	JPasswordField cajaContraseña;

	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(200, 200);
		setLocationRelativeTo(null);
		setLayout(null);
		inicializarComponentes();
		establecerManejador();
		setVisible(true);
	}

	private void inicializarComponentes() {
		cajaTexto = new JTextField(10);
		cajaTexto.setBounds(35, 10, 100, 30);
		add(cajaTexto);

		cajaContraseña = new JPasswordField(10);
		cajaContraseña.setBounds(35, 60, 100, 30);
		cajaContraseña.setEchoChar('*');
		add(cajaContraseña);
	}

	private void establecerManejador() {
		ManejadorEventos manejador = new ManejadorEventos();
		cajaTexto.getDocument().addDocumentListener(manejador);
		cajaContraseña.getDocument().addDocumentListener(manejador);
	}
}
```

```java
package controlador;

import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;

public class ManejadorEventos implements DocumentListener  {

	@Override
	public void insertUpdate(DocumentEvent e) {
		// TODO Auto-generated method stub

	}

	@Override
	public void removeUpdate(DocumentEvent e) {
		// TODO Auto-generated method stub

	}

	@Override
	public void changedUpdate(DocumentEvent e) {
		// TODO Auto-generated method stub

	}

}
```

```java
package controlador;

import vista.Ventana;

public class Main {

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

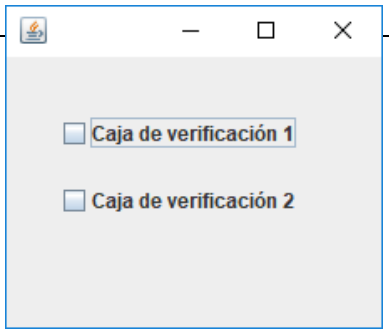
Áreas de texto: JTextArea




```
package vista;

import javax.swing.*;

public class Ventana extends JFrame {

	private JTextArea areaTexto;
	private JScrollPane panelScroll;


	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(275, 300);
		setLocationRelativeTo(null);
		setLayout(null);
		inicializarComponentes();
		setVisible(true);
	}

	private void inicializarComponentes() {
		areaTexto = new JTextArea(150, 150);
		areaTexto.setLineWrap(true);
		// si las líneas son demasiado
		// largas, salta a la siguiente
		// línea.
		areaTexto.setText("Lorem ipsum dolor sit amet, consectetur
	  adipiscing elit. Cras porta bibendum metus nec bibendum.");

		panelScroll = new JScrollPane(areaTexto);
		// se añade el área de texto al scroll
		panelScroll.setBounds(50, 50, 150, 150);
		add(panelScroll);
		 // se añade el scroll con el área de texto a la ventana
	}

}
```

```
package controlador;

import vista.Ventana;

public class Main {

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

Casillas de verificación: JCheckBox



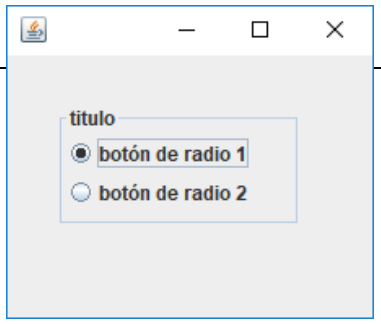

```
package vista;

import javax.swing.*;

import controlador.ManejadorEventos;

public class Ventana extends JFrame {

	private JCheckBox check1, check2;

	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(240, 200);
		setLocationRelativeTo(null);
		setLayout(null);
		inicializarComponentes();
		setVisible(true);
	}

	private void inicializarComponentes() {
		check1 = new JCheckBox("Caja de verificación 1");
		check1.setBounds(30, 30, 150, 30);
		add(check1);

		check2 = new JCheckBox("Caja de verificación 2");
		check2.setBounds(30, 70, 150, 30);
		add(check2);
	}

	public void establecerManejador(ManejadorEventos manejador) {
		check1.addChangeListener(manejador);
		check2.addChangeListener(manejador);
	}

	public JCheckBox getCheck1() {
		return check1;
	}

	public JCheckBox getCheck2() {
		return check2;
	}
}
```

```java
package controlador;

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import vista.Ventana;

public class ManejadorEventos implements ChangeListener {

	private Ventana ventana;

	public ManejadorEventos(Ventana ventana) {
		this.ventana = ventana;
	}

	@Override
	public void stateChanged(ChangeEvent e) {
		if (ventana.getCheck1().isSelected()) {
			//acciones
		}

		if (ventana.getCheck1().isSelected()) {
			//acciones
		}
	}
}
```

```java
package controlador;

import vista.Ventana;

public class Main {

	public static void main(String[] args) {
		Ventana miVentana = new Ventana();
		ManejadorEventos manejador = new ManejadorEventos(miVentana);
		miVentana.establecerManejador(manejador);
	}
}
```

Botones de radio: JRadioButton

Los botones de radio deben meterse en un mismo grupo para que se desactiven mutuamente.



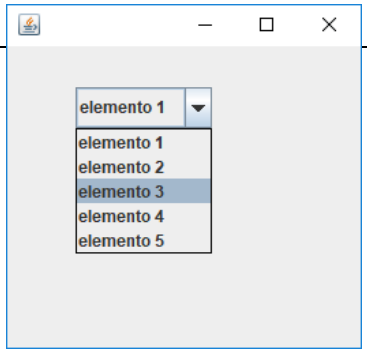

```
package vista;

import javax.swing.*;

import controlador.ManejadorEventos;

public class Ventana extends JFrame {

	private JRadioButton radio1, radio2;
	private ButtonGroup grupo;

	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(240, 200);
		setLocationRelativeTo(null);
		setLayout(null);
		inicializarComponentes();
		setVisible(true);
	}

	private void inicializarComponentes() {
		radio1 = new JRadioButton("botón de radio 1", true);
		radio2 = new JRadioButton("botón de radio 2");

		grupo = new ButtonGroup();
		grupo.add(radio1);
		grupo.add(radio2);

		JPanel panel = new JPanel();
		panel.setBounds(30, 30, 150, 75);

		// configuración de la ubicación de los componentes
		// en el contenedor (en vertical, es decir, siguiendo el eje Y)
		panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

		// añadir componentes al contenedor
		panel.add(radio1);
		panel.add(radio2);

		// establecimiento de un borde y un título para el contenedor
		panel.setBorder(BorderFactory.createTitledBorder("titulo"));

		// añadir el contenedor a la ventana
		add(panel);
	}

	public void establecerManejador(ManejadorEventos manejador) {
		radio1.addChangeListener(manejador);
		radio2.addChangeListener(manejador);
	}
```

```
	public JRadioButton getRadio1() {
		return radio1;
	}

	public JRadioButton getRadio2() {
		return radio2;
	}
}
```

```
package controlador;

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import vista.Ventana;

public class ManejadorEventos implements ChangeListener {

	private Ventana ventana;

	public ManejadorEventos(Ventana ventana) {
		this.ventana = ventana;
	}

	@Override
	public void stateChanged(ChangeEvent e) {
		if (ventana.getRadio1().isSelected()) {
			//acciones
		}

		if (ventana.getRadio2().isSelected()) {
			//acciones
		}

	}

}
```

```
package controlador;

import vista.Ventana;

public class Main {

	public static void main(String[] args) {
		Ventana miVentana = new Ventana();
		ManejadorEventos manejador = new ManejadorEventos(miVentana);
		miVentana.establecerManejador(manejador);
	}
}
```

Listas desplegables: JComboBox

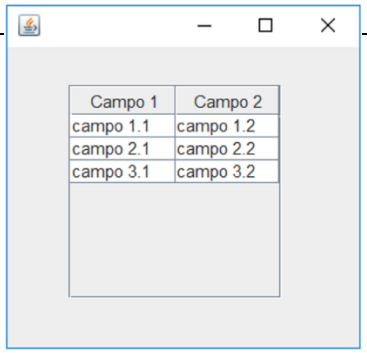



```
package vista;

import javax.swing.*;

import controlador.ManejadorEventos;

public class Ventana extends JFrame {

	private JComboBox combo;

	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(275, 260);
		setLocationRelativeTo(null);
		setLayout(null);
		inicializarComponentes();
		setVisible(true);
	}

	private void inicializarComponentes() {
		String[] valores = { "elemento 1", "elemento 2", "elemento 3",
	    "elemento 4", "elemento 5" };
		combo = new JComboBox(valores);
		combo.setBounds(50, 30, 100, 30);
		add(combo);
	}

	public void establecerManejador(ManejadorEventos manejador) {
		combo.addItemListener(manejador);
	}

	public JComboBox getCombo() {
		return combo;
	}
}
```

```java
package controlador;

import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import vista.Ventana;

public class ManejadorEventos implements ItemListener {

	private Ventana ventana;

	public ManejadorEventos(Ventana ventana) {
		this.ventana = ventana;
	}

	@Override
	public void itemStateChanged(ItemEvent e) {
		int indiceSeleccionado = ventana.getCombo().getSelectedIndex();

		switch (indiceSeleccionado) {
			case 0:

			case 1:

			...
		}
	}
}
```

```java
package controlador;

import vista.Ventana;

public class Main {

	public static void main(String[] args) {
		Ventana miVentana = new Ventana();
		ManejadorEventos manejador = new ManejadorEventos(miVentana);
		miVentana.establecerManejador(manejador);
	}
}
```

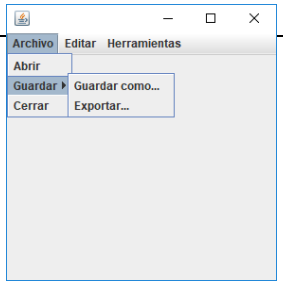
Tablas de datos




```
package vista;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
import controlador.ManejadorEventos;

public class Ventana extends JFrame {

	private JTable table;
	JScrollPane scrollPane;
	private DefaultTableModel tableModel;

	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(275, 260);
		setLocationRelativeTo(null);
		setLayout(null);
		inicializarComponentes();
		setVisible(true);
	}

	private void inicializarComponentes() {
		String[] columnNames = {"Campo 1", "Campo 2"};
		tableModel = new DefaultTableModel(columnNames, 0);
		table = new JTable(tableModel);

		scrollPane = new JScrollPane(table);
		scrollPane.setBounds(50,25,150,150);
		add(scrollPane);
	}

	public void establecerManejador(ManejadorEventos manejador) {
		table.addMouseListener(manejador);
	}

	public DefaultTableModel getTableModel() {
		return tableModel;
	}

	public JTable getTable() {
		return table;
	}

}
```

```java
package controlador;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import vista.Ventana;

public class ManejadorEventos implements MouseListener{

	private Ventana ventana;

	public ManejadorEventos(Ventana ventana) {
		this.ventana = ventana;
		rellenarTablaConDatos();
	}

	public void rellenarTablaConDatos() {
		ventana.getTableModel().addRow(new String[]{"campo 1.1","campo 1.2"});
		ventana.getTableModel().addRow(new String[]{"campo 2.1","campo 2.2"});
		ventana.getTableModel().addRow(new String[]{"campo 3.1","campo 3.2"});
	}

	@Override
	public void mouseClicked(MouseEvent e) {
		// Se invoca cuando se hace clic en el botón del mouse
		// (presionado y soltado) en un componente.
	}

	@Override
	public void mousePressed(MouseEvent e) {
		// Se invoca cuando se presiona un botón del mouse sobre un componente.

	}

	@Override
	public void mouseReleased(MouseEvent e) {
		// Se invoca cuando se suelta un botón del mouse en un componente.

	}

	@Override
	public void mouseEntered(MouseEvent e) {
		// Se invoca cuando el ratón entra en un componente.

	}

	@Override
	public void mouseExited(MouseEvent e) {
		// Se invoca cuando el ratón sale de un componente.

	}
}
```

Menús: JMenuBar, JMenu, JMenuItem




```
package vista;

import javax.swing.*;
import controlador.ManejadorEventos;

public class Ventana extends JFrame {

	private JMenuBar barra;
	private JMenu menuArchivo, menuEditar, menuHerramientas, guardar;
	private JMenuItem abrir, cerrar, guardarComo, exportar;

	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(300, 300);
		setLocationRelativeTo(null);
		setLayout(null);
		inicializarComponentes();
		setVisible(true);
	}

	private void inicializarComponentes() {
		barra = new JMenuBar();
		barra.setBounds(0, 0, 300, 30);
		add(barra);

		menuArchivo = new JMenu("Archivo");
		menuEditar = new JMenu("Editar");
		menuHerramientas = new JMenu("Herramientas");

		barra.add(menuArchivo);
		barra.add(menuEditar);
		barra.add(menuHerramientas);

		abrir = new JMenuItem("Abrir");

		guardar = new JMenu("Guardar");

		guardarComo = new JMenuItem("Guardar como...");
		exportar = new JMenuItem("Exportar...");

		guardar.add(guardarComo);
		guardar.add(exportar);

		cerrar = new JMenuItem("Cerrar");

		menuArchivo.add(abrir);
		menuArchivo.add(guardar);
		menuArchivo.add(cerrar);
	}


	public void establecerManejador(ManejadorEventos manejador) {
		abrir.addActionListener(manejador);
		guardarComo.addActionListener(manejador);
```

```
                exportar.addActionListener(manejador);
                cerrar.addActionListener(manejador);
        }

        public JMenuItem getAbrir() {
                return abrir;
        }

        public JMenuItem getCerrar() {
                return cerrar;
        }

        ...
}
```

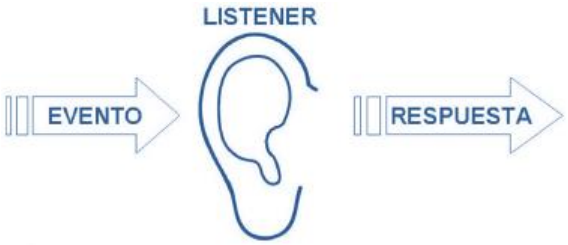
```
package controlador;

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

import vista.Ventana;

public class ManejadorEventos implements ActionListener {

        private Ventana ventana;

        public ManejadorEventos(Ventana ventana) {
                this.ventana = ventana;
        }

        @Override
        public void actionPerformed(ActionEvent e) {
                if (e.getSource() == ventana.getAbrir()) {
                        //acciones
                }

                if (e.getSource() == ventana.getCerrar()) {
                        System.exit(0);
                }
        }
}
```

```
package controlador;

import vista.Ventana;

public class Main {

        public static void main(String[] args) {
                Ventana miVentana = new Ventana();
                ManejadorEventos manejador = new ManejadorEventos(miVentana);
                miVentana.establecerManejador(manejador);
        }
}
```

Eventos y Controladores de eventos

En términos de Java, un evento es un objeto que es lanzado por un objeto y enviado aotro objeto llamado escuchador (listener). Un evento se lanza (o se dispara)cuando ocurre una determinada situación (un clic de ratón, una pulsación de tecla,...).

La programación de eventos es una de las bases de Java y permite mecanismos dediseño de programas orientados a las acciones del usuario. Es decir, son las acciones delusuario las que desencadenan mensajes entre los objetos (el flujo del código delprograma se desvía en función del evento producido, alterando la ejecución normal).



Existen una serie de manejadores de eventos que deberán asociarse al componente para que éste ejecute la respuesta necesaria. Estos listeners son diferentes dependiendo de los eventos a los que van a dar respuesta.

Imprescindible: identificar los tres objetos implicados:

- El objeto fuente. Que es el objeto que lanza los eventos. Dependiendo del tipo de objeto que sea, puede lanzar unos métodos u otros. El hecho de que dispare esos eventos no significa que el programa tenga que, necesariamente, realizar una acción. Sólo se ejecuta una acción si hay un objeto escuchando. Por ejemplo un botón.

- El objeto de evento. Se trata del objeto que es enviado desde el objeto fuente al escuchador. Es el suceso que se produce. Por ejemplo hacer clic (sobre el botón).

- El objeto escuchador u oyente (listener). Se trata del objeto que recibe el evento producido. Es el objeto que captura el evento y ejecuta el código correspondiente. Para ello debe implementar una interfaz relacionada con el tipo de evento que captura. Esa interfaz obligará a implementar uno o más métodos cuyo código es el que se ejecuta cuando se dispare el evento. Por ejemplo, la ventana (algo va a cambiar en la ventana o en algún componente que contiene esa ventana), también se puede crear una clase interna que se ocupe solo de esta parte.

Eventos de botón / campo de texto / menú

Interface	Método		Métodos del evento "e"	
ActionListener	**actionPerformed(ActionEvent e)**	Se ha hecho clic sobre el componente.	Object getSource()	Obtiene el objeto fuente que ha lanzado el evento.

Eventos de combo

Interface	Método		Métodos del evento "e"	
ItemListener	**itemStateChanged(ItemEvent e)**	Se ha seleccionado un elemento de la lista.	Object getSource()	Obtiene el objeto fuente que ha lanzado el evento.

Eventos de casilla de verificación / radio

Interface	Método		Métodos del evento "e"	
ChangeListener	**stateChanged(ChangeEvent e)**	Ha cambiado el estado (seleccionado/deseleccionado).	Object getSource()	Obtiene el objeto fuente que ha lanzado el evento.

Eventos de teclado

Interface	Método		Métodos del evento "e"	
KeyListener	keyPressed(KeyEvent e)	Se ha pulsado una tecla.	char getKeyChar()	Devuelve el carácter asociado con la tecla pulsada
	keyReleased(KeyEvent e)	Se ha liberado una tecla.	int getKeyCode	Devuelve el valor entero que representa la tecla pulsada
	keyTyped(KeyEvent e)	Se ha pulsado y liberado una tecla.	String getKeyText()	Devuelve un texto que representa el código de la tecla
			Object getSource()	Objeto que produjo el evento

Eventos de ratón

Interface	Método	
MouseListener	mousePressed(MouseEvent e)	Se ha pulsado un botón del ratón en un componente.
	mouseReleased(MouseEvent e)	Se ha liberado un botón del ratón en un componente.
	mouseClickerd(MouseEvent e)	Se ha pulsado y liberado un botón del ratón en un componente.
	mouseEntered(KeyEvent e)	Se ha entrado (con puntero del ratón) en un componente.
	mouseExited(KeyEvent e)	Se ha salido (con puntero del ratón) en un componente.
MouseMotionListener	mouseMoved(MouseEvent e)	Se mueve el puntero del ratón sobre un componente.

Programación del manejo de eventos

1. Implementar el método de la interfaz que se usa como listener.
2. crear el componente.
3. Se añade el listener adecuado al componente y el listener escuchará la acción sobre el componente.

a) MANEJAR EVENTOS DE CLIC (botón, menú, casilla de verificación, radio...)

1. Crear la clase oyente que implemente la interface **ActionListener**:

```
public class Manejador implements ActionListener {

}
```

2. Crear el objeto fuente (componente) y configurarlo para que lance eventos:

```
JButton boton = new JButton("aceptar");
boton.addActionListener(new Manejador());
```

3. Implementar el método de la interface en la clase oyente:

```
@Override
public void actionPerformed(ActionEvent e){
  //qué hacer cuando se ha hecho clic en el botón
}
```

Ejemplo

```
public class Ventana extends JFrame {

	private JButton boton;

	public Ventana() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(300, 300);
		setLocationRelativeTo(null);
		setLayout(null);
		inicializarComponentes();
		setVisible(true);
	}

	private void inicializarComponentes() {
		boton = new JButton("Cerrar");
	boton.setBounds(50, 50, 100, 40);
	boton.addActionListener(new ManejadorEventos());
	add(boton);
	}
}
```

```
public class ManejadorEventos implements ActionListener {

	@Override
	public void actionPerformed(ActionEvent e) {
System.exit(0);
	}
}
```

b) MANEJAR EVENTOS DE TECLADO.

Crear el objeto oyente del teclado (una clase que implemente la interface KeyListener y sus metodos) y activar la escucha a través de:

addKeyListener(objeto_que_implementa_la_interface):

```
public class Ventana extends JFrame {

        public Ventana() {
            addKeyListener(new EventoTeclado());
   }
}
```

```
Public class EventoTeclado implements KeyListener {

            @Override
            public void keyTyped(KeyEvent e) {
                System.out.println(e.getKeyChar());
            }

            @Override
              public void keyPressed(KeyEvent e) {
            }

            @Override
             public void keyReleased(KeyEvent e) {
            }
}
```

c) MANEJAR EVENTOS DE RATON.

Crear el objeto oyente del ratón (una clase que implemente la interface MouseListener y sus metodos) y activar la escucha a través de:

addMouseListener(objeto_que_implementa_la_interface):

```
public class Ventana extends JFrame {

  public Ventana() {
      addMouseListener(new EventoRaton());
  }

}
```

```
public  class EventoRaton implements MouseListener {

    @Override
    public void mouseClicked(MouseEvent me) {
       System.out.println("has hecho clic");
    }

    @Override
    public void mousePressed(MouseEvent me) {
       System.out.println("has presionado");
    }

    @Override
    public void mouseReleased(MouseEvent me) {
       System.out.println("has levantado");
    }

    @Override
    public void mouseEntered(MouseEvent me) {
    }

    @Override
    public void mouseExited(MouseEvent me) {
    }

}
```

d) MANEJAR EVENTOS DE CAJA DE TEXTO.

1. Clase que implemente la interface DocumentListener.

```
public class Manejador implements DocumentListener{
```

2. Escribir el código para sus métodos:

```
    @Override
    public void insertUpdate(DocumentEvent e) {
    }

    @Override
    public void removeUpdate(DocumentEvent e) {
    }

    @Override
    public void changedUpdate(DocumentEvent e) {

    }
```

3. Poner la caja de texto a la escucha:

```
public Ventana(){
   cajaTexto = new JTextField();
   cajaTexto.getDocument().addDocumentListener(new Manejador());
   add(cajaTexto);
}
```

Mensajes hacia el usuario: clase JOptionPane

Una de las labores típicas en la creación de aplicaciones gráficas del tipo que sea, es la de comunicarse con el usuario a través de mensajes en forma de cuadro de diálogo.

Algunos cuadros son extremadamente utilizados por su sencillez (textos de aviso, error, confirmación, entrada sencilla de datos, etc.).

La clase JOptionPane deriva de JComponent y es la encargada de crear este tipo de cuadros. Aunque posee constructores, normalmente se utilizan mucho más una serie de métodos estáticos que permiten crear de forma más sencilla objetos JOptionPane.

Cuadros de información

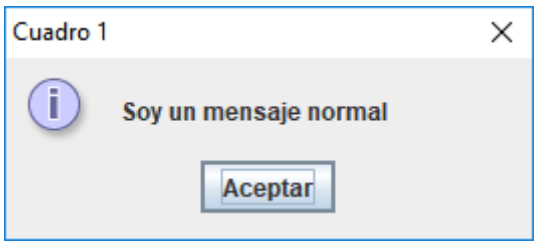
Son cuadros de diálogo que sirven para informar al usuario de un determinado hecho.Se construyen utilizando los siguientes métodos estáticos:

métodos	uso
static void **showMessageDialog(** Component padre, Object mensaje**)**	Muestra un cuadro de diálogo en el contenedor *padre* indicado con un determinado mensaje
static void **showMessageDialog(** Component padre, Object mensaje, String título, int tipo**)**	Muestra un cuadro de diálogo en el contenedor *padre* indicado con un determinado mensaje, título y tipo.
static void **showMessageDialog(** Component padre, Object mensaje, String título, int tipo, Icon i**)**	Igual que el anterior pero se permite indicar un icono para acompañar el mensaje

El tipo puede ser una de estasconstantes:

- JOptionPane.INFORMATION_MESSAGE.
- JOptionPane.ERROR_MESSAGE.
- JOptionPane.WARNING_MESSAGE.
- JOptionPane.QUESTION_MESSAGE.
- JOptionPane.PLAIN_MESSAGE.

```
JOptionPane.showMessageDialog(null,"Soy un mensaje normal","Cuadro 1",
JOptionPane.INFORMATION_MESSAGE);
```



Cuadros de confirmación

La diferencia con los anteriores reside en que en estos hay que capturar la respuesta delusuario para comprobar si acepta o declina el mensaje. Los métodos estáticos decreación son:

métodos	uso
static int **showConfirmDialog(** Component padre, Object mensaje**)**	Muestra un cuadro de confirmación en el componente *padre* con el mensaje indicado y botones de **Sí, No** y **Cancelar**
static int **showConfirmDialog(** Component padre, Object mensaje, **String** título, int opciones**)**	Muestra cuadro de confirmación con el título y mensaje reseñados y las opciones indicadas (las opciones se describen al final)
static int **showConfirmDialog(** Component padre, Object mensaje, String título, int opciones, int tipo**)**	Como el anterior pero indicando el tipo de cuadro (los posibles valores son los indicados en la página anterior) y un icono
static int **showConfirmDialog(** Component padre, Object mensaje, String título, int opciones, int tipo, Icon icono**)**	Como el anterior pero indicando un icono.

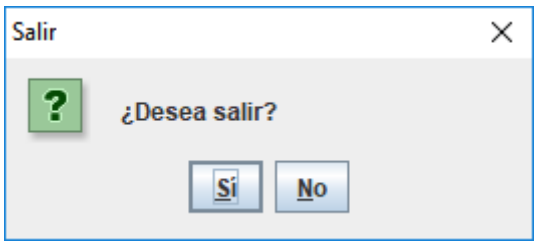
Estas constantes facilitan el uso del parámetro ***opciones*** que sirve para modificarla funcionalidad del cuadro. Son:

- JOptionPane.OK_CANCEL_OPTION. Cuadro con los botones OK y Cancelar.
- JOptionPane.YES_NO_OPTION. Cuadro con botones Sí y No .
- JOptionPane.YES_NO_CANCEL_OPTION. Cuadro con botones Sí, No y Cancelar.

Obsérvese como el tipo de retorno es un número entero; este número representa el botón del cuadro sobre el que el usuario hizo clic. Este valor se puede representar por medio de estas constantes de JOptionPane:

- JOptionPane.NO_OPTION. El usuario no pulsó ningún botón en el cuadro.
- JOptionPane.CLOSE_OPTION. El usuario cerró sin elegir nada.
- JOptionPane.OK_OPTION. El usuario pulsó OK.
- JOptionPane.YES_OPTION. El usuario pulsó el botón Sí.
- JOptionPane.CANCEL_OPTION. El usuario pulsó el botón Cancelar.

```
int res = JOptionPane.showConfirmDialog(null, "¿Desea salir?",
                                        "Salir", JOptionPane.YES_NO_OPTION,
                                         JOptionPane.QUESTION_MESSAGE);
if (res == JOptionPane.YES_OPTION)      System.exit(0);
```



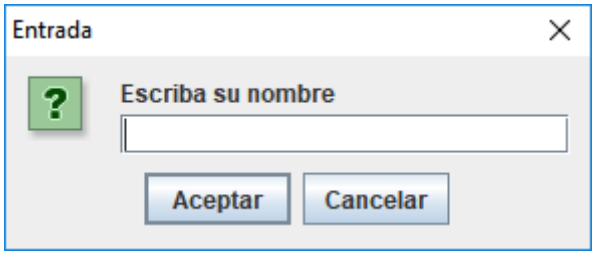
Cuadros de diálogo para rellenar entradas

Son cuadros que permiten que el usuario, desde un mensaje de sistema, rellene una determinada variable.

métodos	uso
static String **showInputDialog(**Object mensaje**)**	Muestra un cuadro de entrada con el mensaje indicado
static String **showInputDialog(** Component padre, Object mensaje**)**	Muestra un cuadro de entrada en el componente *padre* con el mensaje indicado
static **String showInputDialog(** Component padre, Object mensaje, String título, int tipo**)**	Muestra cuadro de entrada con el título y mensaje reseñados y el tipo que se indica
static Object **showInputDialog(** Component padre, Object mensaje, String título, int tipo, Icono icono, Object[] selección, Object selecciónInicial**)**	Indica además un icono, selecciones posibles y la selección inicial. El valor devuelto es un objeto **Object**.

Todos los métodos devuelven un String en el que se almacena la respuesta del usuario.En caso de que el usuario cancele el cuadro, devuelve null en la cadena a examinar.

```
String res = JOptionPane.showInputDialog("Escriba su nombre");
if (res == null)
        JOptionPane.showMessageDialog(null,"No escribió", "Cuadro 1",
                                                JOptionPane.WARNING_MESSAGE);
else
        nombre=res;
```



Organización de los controles en un contenedor

Para organizar los controles en un objeto que implemente la interfaz LayoutManager (por ejemplo, los paneles) es necesario establecer en ella un Layout Manager o administrador de diseño.

Un administrador de diseño se encarga de disponer la presentación de los componentes en un dispositivo de presentación concreto.

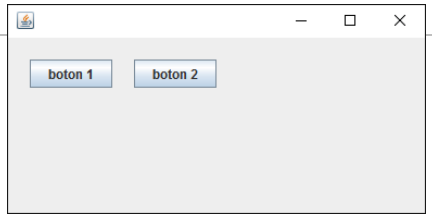
En algunos entornos los componentes se colocan con coordenadas absolutas. En Java se desaconseja esa práctica porque en la mayoría de casos es imposible prever el tamaño de un componente. Por tanto, en su lugar, se usan los layout managers que permiten colocar y maquetar de forma independiente de las coordenadas. Debemos hacer un buen diseño de la interfaz gráfica, y así tenemos que elegir el mejor gestor de distribución para cada uno de los contenedores o paneles de nuestra ventana. Esto podemos conseguirlo con el método setLayout(), a que se le pasa como argumento un objeto del tipo Layout que se quiere establecer. Los Layout Manager se pueden establecer al crear el objeto (constructor).

Tipo de Layout Manager	Definición
FlowLayout	Coloca los componentes en el contenedor de izquierda a derecha. Es el Layout Manager por defecto en los paneles.
BorderLayout	Divide el contenedor en cinco partes: norte, sur, este, oeste y centro.
CardLayout	Permite colocar grupos de componentes diferentes en momentos diferentes de la ejecución del programa.
GridLayout	Coloca los componentes en filas y columnas.
GridBagLayout	Coloca los componentes en filas y columnas, pero un componente puede ocupar más de una columna.
BoxLayout	Coloca los componentes en una fila o columna ajustándose.

Más Administradores de Diseño.

FlowLayout

Éste es el organizador por defecto. No hace falta establecerlo, pero si se quiere, se puede configurar con otras opciones. Por defecto, alinea los componentes de forma centrada.

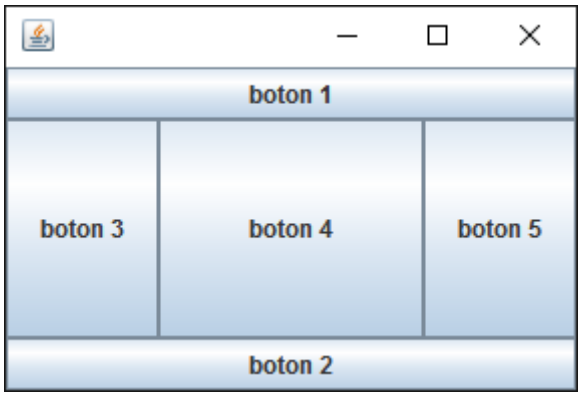



```
package graficos;

import javax.swing.*;

public class Marco extends JFrame {

   public Marco() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(250, 200);
      setLocationRelativeTo(null);
      add(new Lamina());
      setVisible(true);
   }
}
```

```
package graficos;

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

public class Lamina extends JPanel {

   JButton boton1 = new JButton("boton 1");
   JButton boton2 = new JButton("boton 2");

   public Lamina() {
      setLayout(new FlowLayout(FlowLayout.LEFT,20,20));
      add(boton1);
      add(boton2);
   }
}
```

La ubicación por defecto es FlowLayout.CENTER y se puede cambiar a .LEFT o .RIGHT
También admite otros dos parámetros para indicar el espacio horizontal y vertical que habrá entre los componentes.

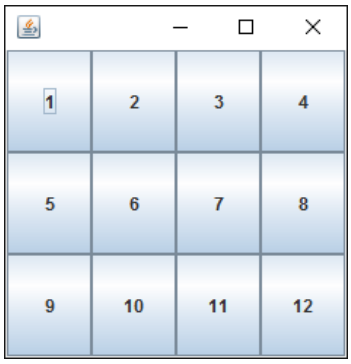
BorderLayout




```
package graficos;

import javax.swing.*;

public class Marco extends JFrame {

   public Marco() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(300, 200);
      setLocationRelativeTo(null);
      add(new Lamina());
      setVisible(true);
   }
}
```

```
package graficos;

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

public class Lamina extends JPanel {

   JButton boton1 = new JButton("boton 1");
   JButton boton2 = new JButton("boton 2");
   JButton boton3 = new JButton("boton 3");
   JButton boton4 = new JButton("boton 4");
   JButton boton5 = new JButton("boton 5");

   public Lamina() {
      setLayout(new BorderLayout());
      add(boton1,BorderLayout.NORTH);
      add(boton2,BorderLayout.SOUTH);
      add(boton3,BorderLayout.WEST);
      add(boton4,BorderLayout.CENTER);
      add(boton5,BorderLayout.EAST);
   }
}
```

GridLayout

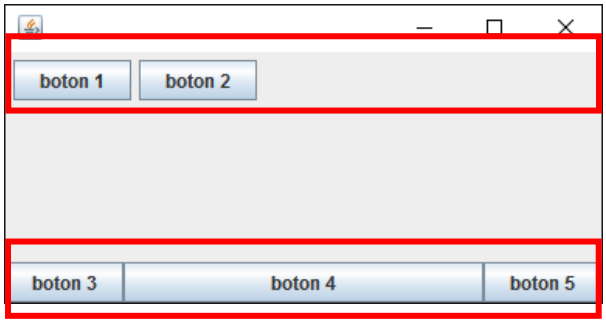



```
package graficos;

import javax.swing.*;

public class Marco extends JFrame {

   public Marco() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(250, 250);
      setLocationRelativeTo(this);
      add(new Lamina());
      setVisible(true);
   }

}
```

```
package graficos;

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

public class Lamina extends JPanel {

   public Lamina() {
      setLayout(new GridLayout(3, 4)); //establecer filas y columnas
      add(new JButton("1"));
      add(new JButton("2"));
      add(new JButton("3"));
      add(new JButton("4"));
      add(new JButton("5"));
      add(new JButton("6"));
      add(new JButton("7"));
      add(new JButton("8"));
      add(new JButton("9"));
      add(new JButton("10"));
      add(new JButton("11"));
      add(new JButton("12"));
   }
}
```

Varias disposiciones a la vez

NORTH

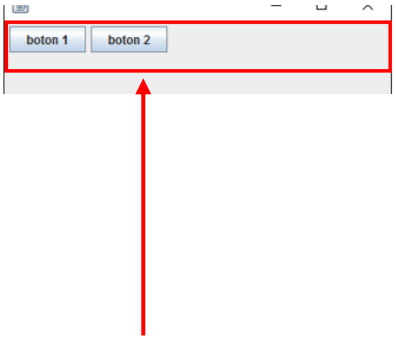


SOUTH

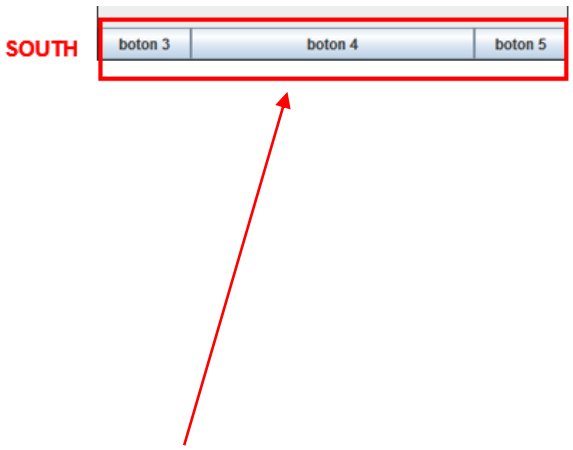
```
package graficos;

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

public class Marco extends JFrame {

   public Marco() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(400, 200);
      setLocationRelativeTo(this);
      add(new Lamina1(), BorderLayout.NORTH);
      add(new Lamina2(), BorderLayout.SOUTH);
      setVisible(true);
   }

}
```

```
package graficos;

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

public class Lamina1 extends JPanel {

   JButton boton1 = new JButton("boton 1");
   JButton boton2 = new JButton("boton 2");

   public Lamina1() {
      setLayout(new FlowLayout(FlowLayout.LEFT));
      add(boton1);
      add(boton2);
   }

}
```

NORTH



```
package graficos;

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

public class Lamina2 extends JPanel {

   JButton boton3 = new JButton("boton 3");
   JButton boton4 = new JButton("boton 4");
   JButton boton5 = new JButton("boton 5");

   public Lamina2() {
      setLayout(new BorderLayout());
      add(boton3, BorderLayout.WEST);
      add(boton4, BorderLayout.CENTER);
      add(boton5, BorderLayout.EAST);
   }

}
```



Para jugar con varias disposiciones tendremos que crear varias capas (láminas o paneles) y colocar cada una de ellas en la zona que corresponda.

Dentro de cada lámina se puede establecer la disposición oportuna y colocar los componentes en ellas.

Clase Color y Font

Los colores primarios son el rojo, el verde y el azul. Java utiliza un modelo de color denominado RGB, que significa que cualquier color se puede describir dando las cantidades de rojo (Red), verde (Green), y azul (Blue). Estas cantidades son números enteros comprendidos entre 0 y 255:

Nombre	Red (rojo)	Green (verde)	Blue (azul)
white	255	255	255
gray	128	128	128
black	0	0	0
red	255	0	0
pink	255	175	175
orange	255	200	0
yellow	255	255	0
green	0	255	0
magenta	255	0	255
cyan	0	255	255
blue	0	0	255

Los colores predefinidos son los siguientes:

Color.white	*Color.black*	*Color.yellow*
Color.orange	*Color.red*	*Color.green*
Color.gray	*Color.pink*	*Color.blue*

Para crear un objeto de la clase Color, se pasan tres números a su constructor que indican la cantidad de rojo, verde y azul:

```
new Color(r, g, b);
```

Para crear una fuente de texto u objeto de la clase Font llamamos a su constructor, y le pasamos el nombre de la fuente de texto, el estilo y el tamaño:

```
new Font("nombre fuente", estilo, tamaño);
```

Los **estilos** vienen datos por constantes (miembros estáticos de la clase Font):

- Font.BOLD, establece el estilo negrita
- Font.ITALIC, el estilo cursiva
- Font.PLAIN, el estilo normal.

Se pueden combinar las constantes Font.BOLD+Font.ITALIC para establecer el estilo negrita y cursiva a la vez.


unir
LA UNIVERSIDAD
EN INTERNET