Centro Asociado Palma de Mallorca

Introducción Práctica de Programación Java

Antonio Rivero Cuesta

Sesión VI

Interfaces Gráficas de Usuario

Introducción

Las GUI completan nuestras aplicaciones con una interfaz formada por ventanas, menús, botones y otros componentes gráficos.

Hacen que la aplicación tenga una apariencia más similar a las típicas aplicaciones que la mayoría de la gente usa hoy en día.

Una vez que sepamos cómo crear una GUI en Java, podremos desarrollar programas que tengan una mejor presentación visual.

Los principios que necesitamos comprender se pueden dividir en tres áreas:

- Qué clase de elementos podemos mostrar en una pantalla.
- Cómo podemos distribuir estos elementos.
- Cómo podemos reaccionar ante una entrada del usuario.

Discutiremos estas cuestiones mediante los términos:

- Componentes.
- Gestores de disposición.
- Manejo de eventos.

Componentes

Un componente es un objeto que tiene una representación gráfica que se puede visualizar en la pantalla y que se puede interactuar con el usuario.

Son las partes individuales a partir de las cuales se construye una GUI.

Son cosas tales como:

- Botones.
- Menús.
- Elementos de menú.
- Cajas de verificación.
- Deslizadores.
- Campos de texto.
- Etc.

La clase ***Component*** es la clase base de todos los componentes de una interfaz de usuario que no sea parte del menú.

Dicha clase hereda de la clase ***Object***.

Para colocar un componente es necesario que exista un contenedor.

Todos los componentes tienen al menos las funcionalidades heredadas de la clase ***Object*** y la clase ***Component***.

Container

Es un contenedor para los objetos GUI y también para los demás container.

Se utiliza junto a un Layout Manager y permite que abarquen más de un GUI.

Permite que aparezcan varios objetos en nuestras interfaces.

Gestores de Disposición

Los ***gestores de disposición, Layout Managers,*** participan de cuestiones relacionadas con la ubicación de los componentes en la pantalla.

Gestores de Disposición

- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout
- GroupLayout
- SpringLayout

Gestión de Eventos

La ***gestión de eventos*** se refiere a la técnica que usaremos para trabajar con las acciones del usuario.

Una vez que hemos creado nuestros componentes y que los posicionamos en la pantalla, también tenemos que estar seguros de que ocurra algo cuando el usuario presione un botón.

AWT y Swing

Java tiene dos bibliotecas para la construcción de interfaces gráficas de usuario.

La más antigua se denomina AWT.

La otra se denomina Swing.

Como existen clases equivalentes en AWT y en Swing, las versiones Swing han sido identificadas mediante el agregado de la letra **J** al comienzo del nombre de la clase.

Crear una Ventana

En Java, estas ventanas del más alto nivel se denominan *frames.*

En Swing, se representan mediante la clase de nombre JFrame.

Necesitamos varias de las clases de los siguientes paquetes:

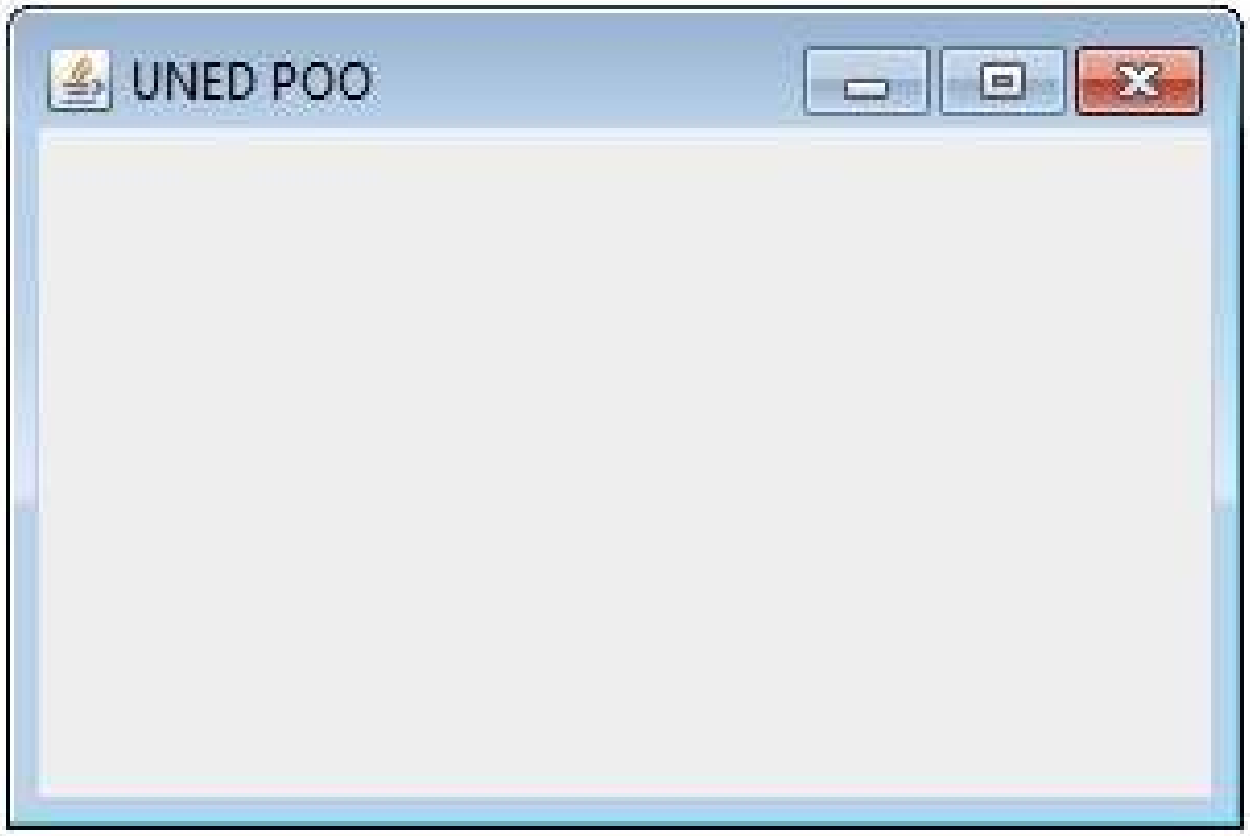
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

La clase debe de tener una variable de instancia de tipo JFrame.

Se usa para contener a la ventana que necesita el visor para mostrar las imágenes en la pantalla.

Voy a usar el ejemplo **`JFrameBasico.`**

JFrameBasico.java



```
import javax.swing.JFrame;
   public class JFrameBasico{
      public static void main(String[] args){
         JFrame f = new JFrame("UNED POO");
         f.setBounds(10,10,300,200);
         f.setVisible(true);
      }
   }
```

La línea:

```
JFrame f = new JFrame("UNED POO");
```

Crea una nueva ventana y la almacena en nuestra variable de instancia, para poder usarla más adelante.

Agregar Componentes Simples

Inmediatamente después la creación del **`JFrameBasico`**, la ventana no estará visible y su panel contenedor estará vacío.

Para que sea visible necesitamos llamar al método:

```
setVisible(true)
```

El método:

```
setBounds(10,10,300,200);
```

Posiciona la nueva ventana en el escritorio.

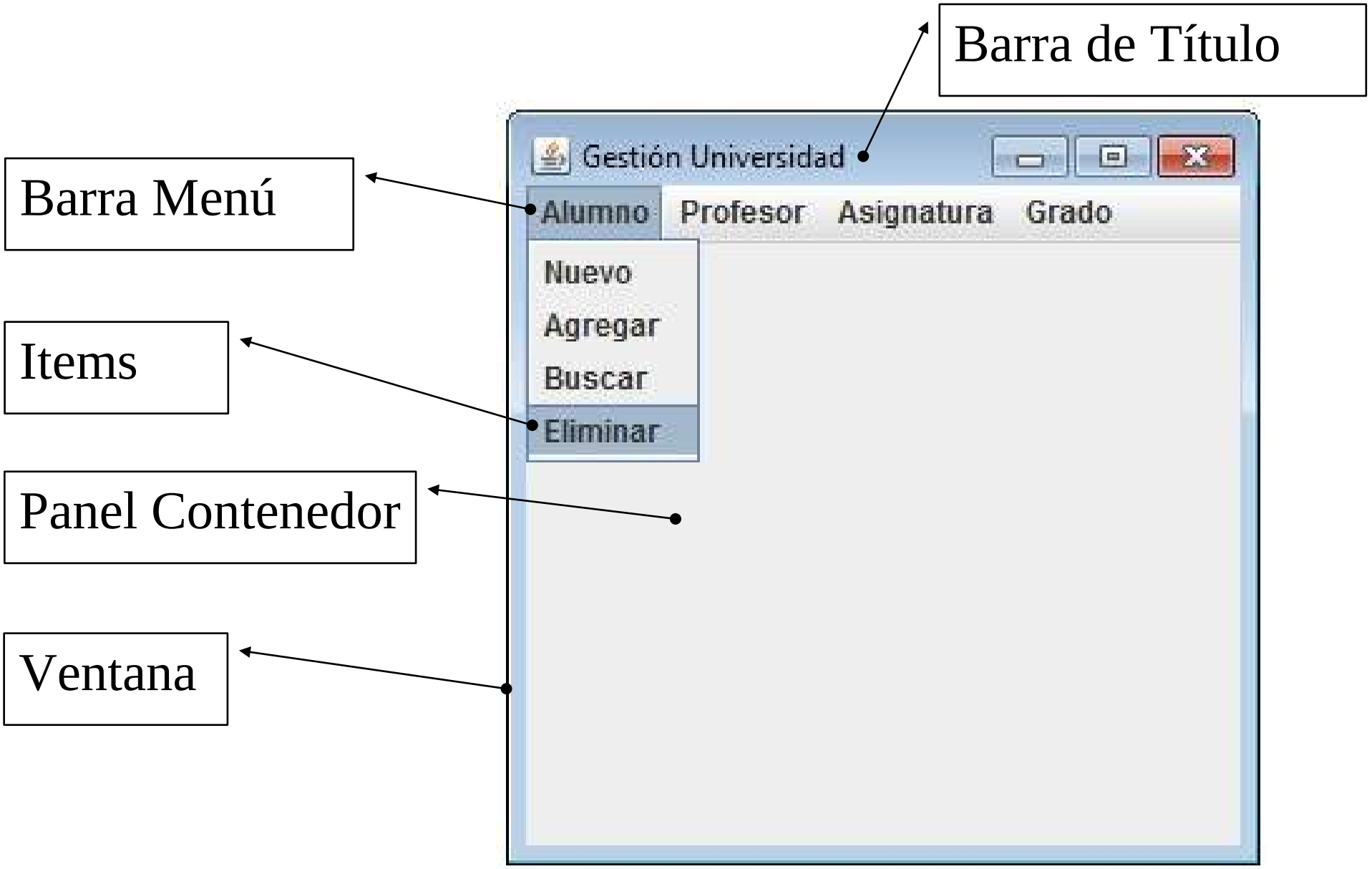
Agregar Menús

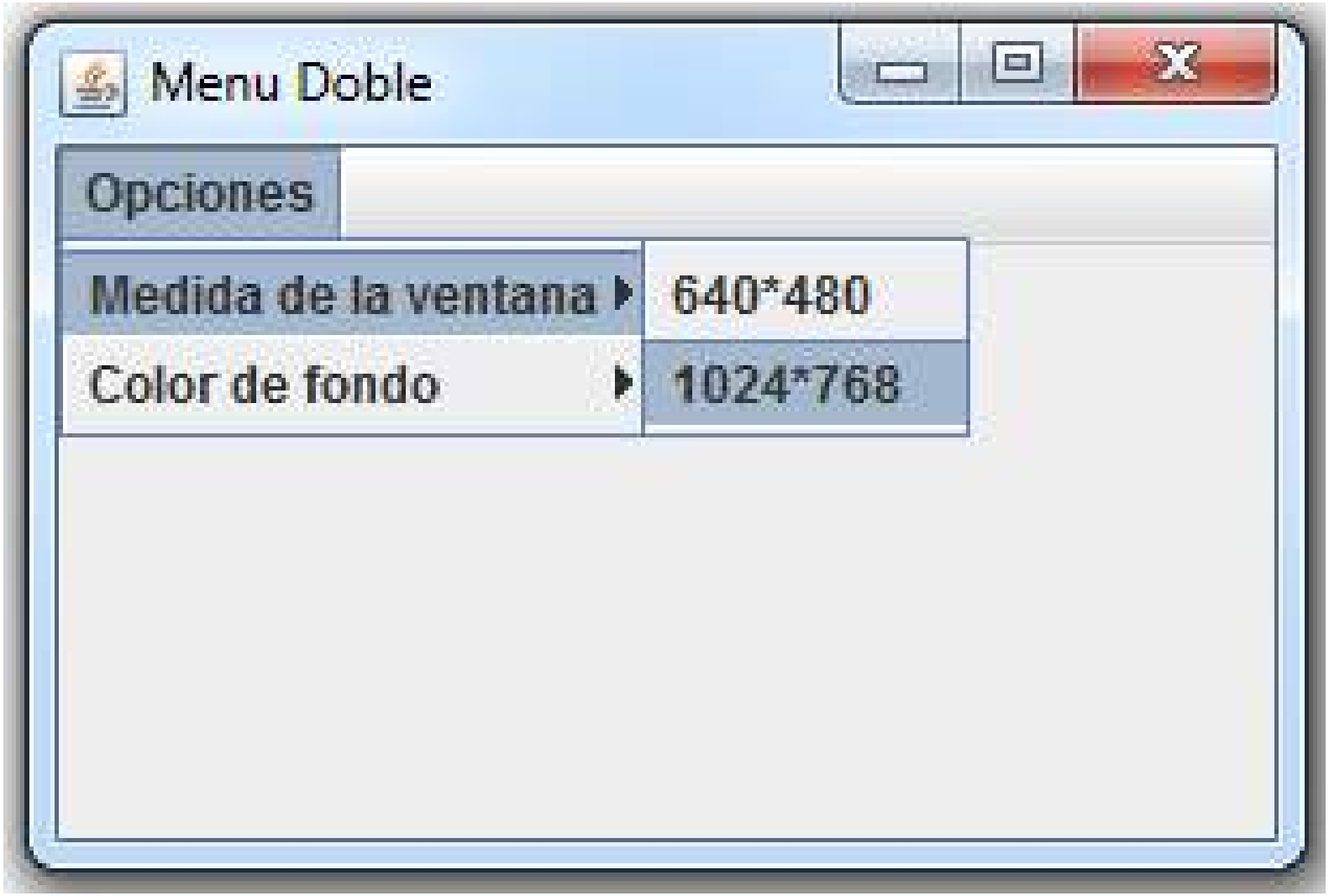
Las tres clases involucradas en esta tarea son:

- **`JMenuBar.`**
- **`JMenu.`**
- **`JMenuItem.`**

Una ventana consta de cuatro partes:

- La barra del título. **`JMenuBar.`**
- Una barra de menú opcional. **`JMenu.`**
- Un menú Item. **`JMenuItem.`**
- Un panel contenedor.


Barra de Título
Barra Menú
Items
Panel Contenedor
Ventana
Gestión Universidad
Alumno
Profesor
Asignatura
Grado
Nuevo
Agregar
Buscar
Eliminar


Menu Doble
Opciones
Medida de la ventana
640*480
Color de fondo
1024*768

JMenuBar

Un objeto de esta clase representa una barra de menú que se puede mostrar debajo de la barra de título en la parte superior de una ventana.

Cada ventana tiene un **`JMenuBar`** como máximo.

JMenu

Los objetos de esta clase representan un solo menú.

Por ejemplo, los menús comunes:

- Archivo.
- Edición.
- Ayuda.

Los menús frecuentemente están contenidos en una barra de menú.

También pueden aparecer en menús emergentes.

JMenuItem

Los objetos de esta clase representan un solo elemento de menú dentro de un menú.

Por ejemplo:

- Abrir.
- Guardar.

```
public class JMenuDoble extends JFrame
implements ActionListener{

private JMenuBar barraMenu;
private JMenu menu1;
private JMenu menu2;
private JMenu menu3;
private JMenuItem menuItem1;
private JMenuItem menuItem2;
private JMenuItem menuItem3;
private JMenuItem menuItem4;
```

```
public JMenuDoble() {

    setTitle("Menu Doble");
    setLayout(null);

    barraMenu = new JMenuBar();
    setJMenuBar(barraMenu);

...
```

```
    menu1 = new JMenu("Opciones");
    barraMenu.add(menu1);

    menu2 = new JMenu("Medida ventana");
    menu1.add(menu2);

    menu3 = new JMenu("Color de fondo");
    menu1.add(menu3);

...
```

```
menuItem1 = new JMenuItem("640*480");
menu2.add(menuItem1);
menuItem1.addActionListener(this);

menuItem2 = new JMenuItem("1024*768");
menu2.add(menuItem2);
menuItem2.addActionListener(this);

menuItem3=new JMenuItem("Rojo");
menu3.add(menuItem3);
menuItem3.addActionListener(this);

menuItem4 = new JMenuItem("Verde");
menu3.add(menuItem4);
menuItem4.addActionListener(this);
```

Hemos llevado a cabo la mitad de nuestra tarea.

Podemos crear y mostrar menús.

Todavía no ocurre nada cuando un usuario selecciona un menú.

Ahora tenemos que agregar código para reaccionar a las selecciones del menú.

Lo vemos en el manejo de eventos.

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == menuItem1) {
       setSize(640,480);
    }
    if (e.getSource() == menuItem2) {
       setSize(1024,768);
    }
    if (e.getSource() == menuItem3) {
  getContentPane().setBackground(Color.RED);
    }
    if (e.getSource() == menuItem4) {
getContentPane().setBackground(Color.GREEN);
    }
}
```

Gestión de Eventos

Java usa un modelo muy flexible para reaccionar ante las acciones que se producen en la GUI.

Un modelo de ***gestión de eventos*** mediante ***oyentes de eventos***.

El marco de trabajo Swing y algunos de sus componentes activan eventos cuando ocurre algo en que otros objetos pueden estar interesados.

Existen diferentes tipos de eventos provocados por diferentes tipos de acciones:

- Cuando se presiona un botón o se selecciona un elemento de un menú, el componente dispara un ActionEvent;
- Cuando se presiona un botón del ratón o se mueve el ratón, se dispara un MouseEvent;
- Cuando se cierra una ventana o se la transforma en icono, se genera un WindowEvent.

Un objeto se convierte en un oyente de eventos mediante la implementación de varias interfaces de oyentes que existen.

Hay dos estilos alterativos para la implementación de oyentes de eventos:

- Un único objeto oye los eventos provenientes de varias fuentes diferentes.
- A cada fuente de eventos diferente se le asigna su propio y único oyente.

Recepción Centralizada Eventos

Para lograr que nuestro objeto se convierta en el único oyente de todos los eventos que provienen del menú tenemos que hacer tres cosas:

l. Debemos declarar, en el encabezado de la clase, que se implementa la interfaz ActionListener.

2. Tenemos que implementar un método con la signatura:

```
public void actionPerformed (ActionEvent e)
```

Este es el único método que se define en la interfaz **`ActionListener`**.

3. Debemos invocar al método **addActionListener** del elemento del menú para registrar al objeto como un oyente.

```
menuItem4 = new JMenuItem("Verde");
menu3.add(menuItem4);
menuItem4.addActionListener(this);
```

Se crea un elemento del menú.

Se registra el objeto actual como un oyente de acción.

Pasando al método `addActionListener` el parámetro `this`.

ActionListener

Es una Interface del grupo de los Listeners.

Tiene un sólo método:

```
void actionPerformed(ActionEvent e).
```

Se usa para detectar y manejar eventos de acción.

`ActionEvent`, son los eventos que tienen lugar cuando se produce una acción sobre un elemento del programa.

Tenemos que importar:

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

Formas de realizarlo:

- La **clase principal** implementa ActionListener.
- **Clase Interna**. La clase principal no implementa ActionListener pero sí lo hace otra clase. 10.2.
- **Clase Anónima**. La clase principal no implementa ActionListener. 10.6.

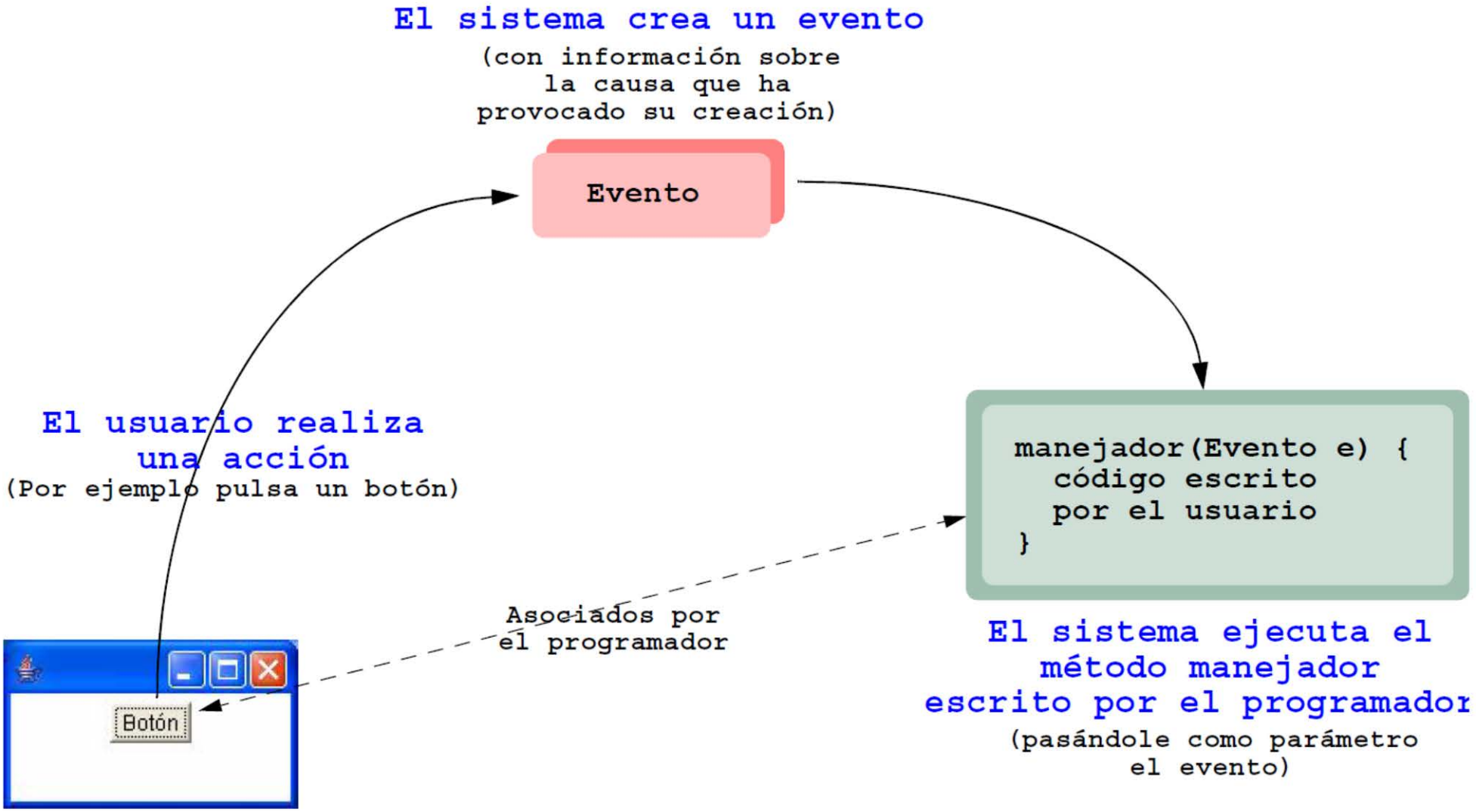
```
public class JMenuDoble extends JFrame implements
ActionListener{

public void actionPerformed(ActionEvent e) {
    if (e.getSource() == menuItem1) {
        setSize(640,480);
    }
    if (e.getSource() == menuItem2) {
        setSize(1024,768);
    }
    if (e.getSource() == menuItem3) {
   getContentPane().setBackground(Color.RED);
    }
    if (e.getSource() == menuItem4) {
   getContentPane().setBackground(Color.Blue);
    }
}
```

```
// Se crea una Clase Interna.
class OyenteAccion implements ActionListener{
   public void actionPerformed(ActionEvent e){
   JButton boton = (JButton)evento.getSource();
etiqueta.setText("Boton pulsado: " +
boton.getText());
   } // fin actionPerformed
} //FIN OyenteAccion
```

```
// Clase Anonima.
calcular.addActionListener(
   new ActionListener() {
      public void actionPerformed (ActionEvent e){
Double peso = Double.parseDouble(campoPeso.getText());
Double altura =
Double.parseDouble(campoAltura.getText());
Double imc = peso / (altura * altura);
String cadena = String.format("%6.2f", imc);
campoIMC.setText(cadena);
   }
   }
);// FIN addActionListener
```

Generación de Eventos



Ejemplos de eventos y sus escuchadores	
Acción que lanza un evento	Tipo de escuchador
El usuario hace un click, presiona Return en un área de texto o selecciona un menú	`ActionListener`
El usuario escoge un frame (ventana principal)	`WindowListener`
El usuario hace un click sobre una componente	`MouseListener`
El usuario pasa el mouse sobre una componente	`MouseMotionListener`
Una componente se hace visible	`ComponentListener`
Una componente adquiere el foco del teclado	`FocusListener`
Cambia la selección en una lista o tabla	`ListSelectionListener`

JFrame y Contenedores

Las ventanas de la librería “javax.swing” se engloban en la clase JFrame.

Lo único que tenemos que hacer es extender la clase principal de nuestro programa con JFrame quedando así:

```
public class Hola extends JFrame
```

¿Que son los Contenedores?

Son componentes que permiten almacenar, alojar o contener otros elementos gráficos.

Es el Tapiz donde vamos a pintar.

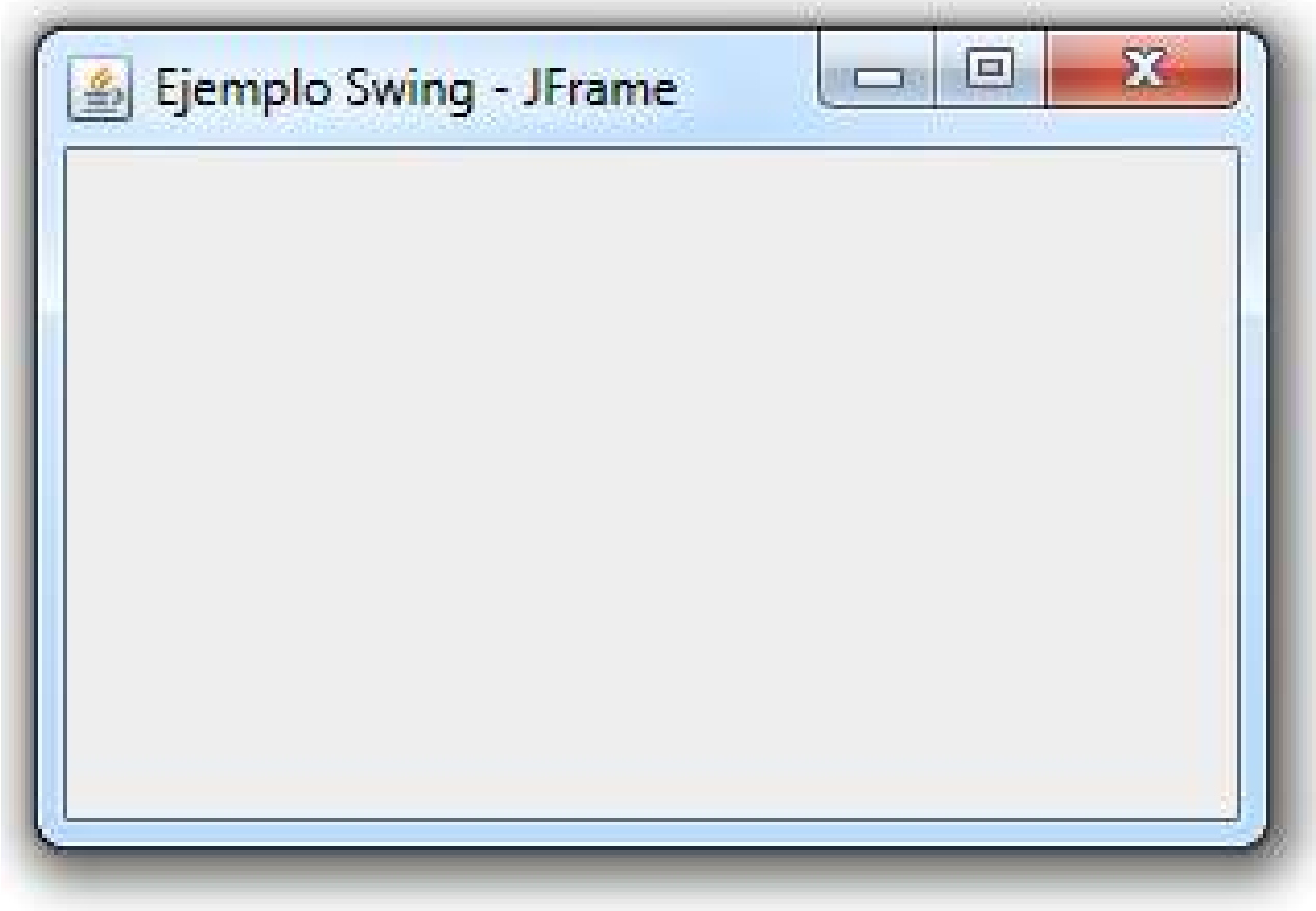
¿Cuáles Son?

Java Swing provee algunos contenedores útiles para diferentes casos.

Así cuando desarrollamos una Ventana podemos decidir:

- De qué manera presentar nuestros elementos.
- Como serán alojados.
- De qué forma serán presentados al usuario.

JFrame



JFrame

Este contenedor es uno de los principales y más usados.

Representa la ventana Principal de nuestra aplicación.

En el podemos alojar otros contenedores.

JPanel

Los JPanel en Java **son** objetos contenedores.

La finalidad de estos objetos es la agrupación de otros objetos tales como **botones, campos de texto, etiquetas, selectores, etc**.

Una gran ventaja de **usar JPanel en Java** es que podemos manejar la agrupación de una mejor forma.

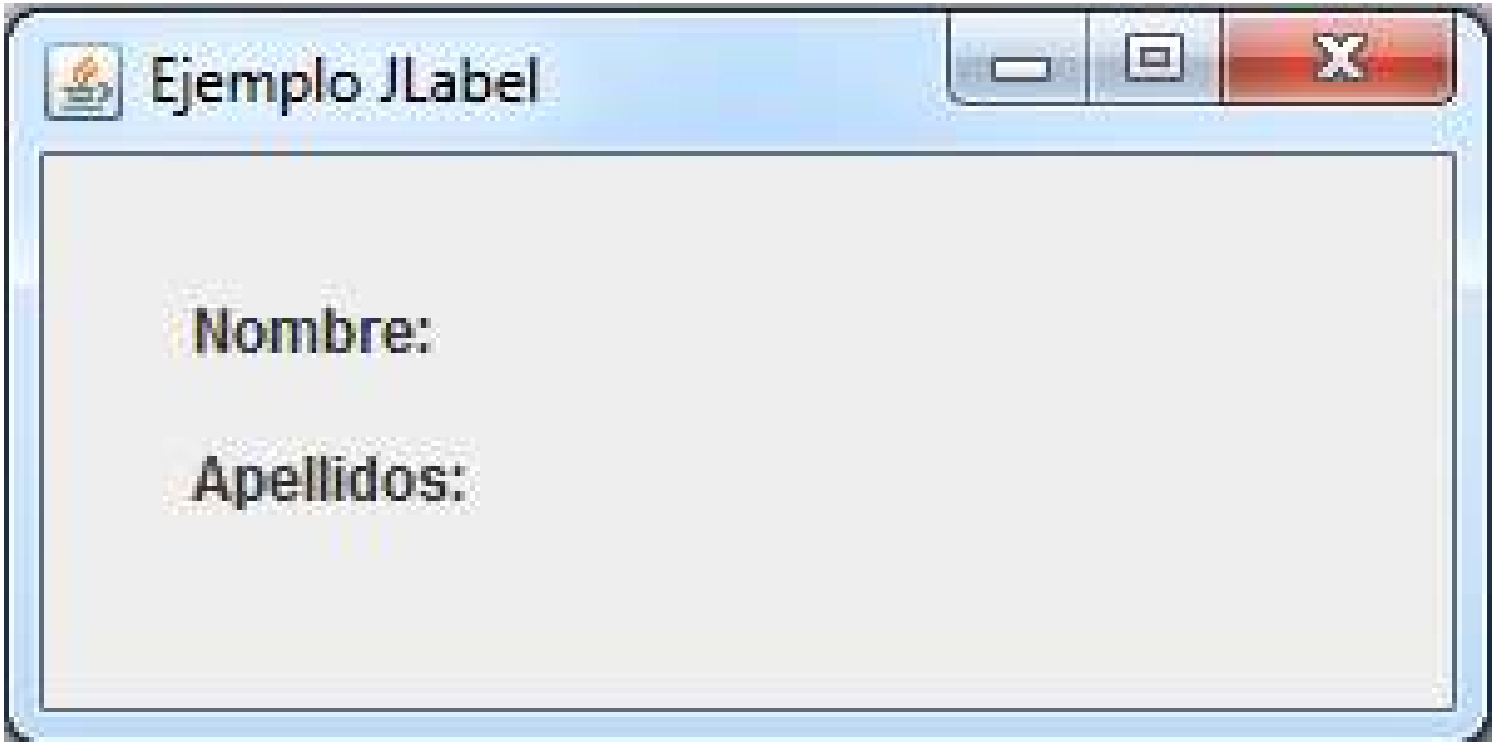
Supongamos que tenemos una serie de botones en un panel.

Deseamos desactivarlos todos a las vez.

En lugar de hacerlo individualmente con los botones, podemos desactivar el panel y con esto los botones.

JPanelEjemplo

JLabel

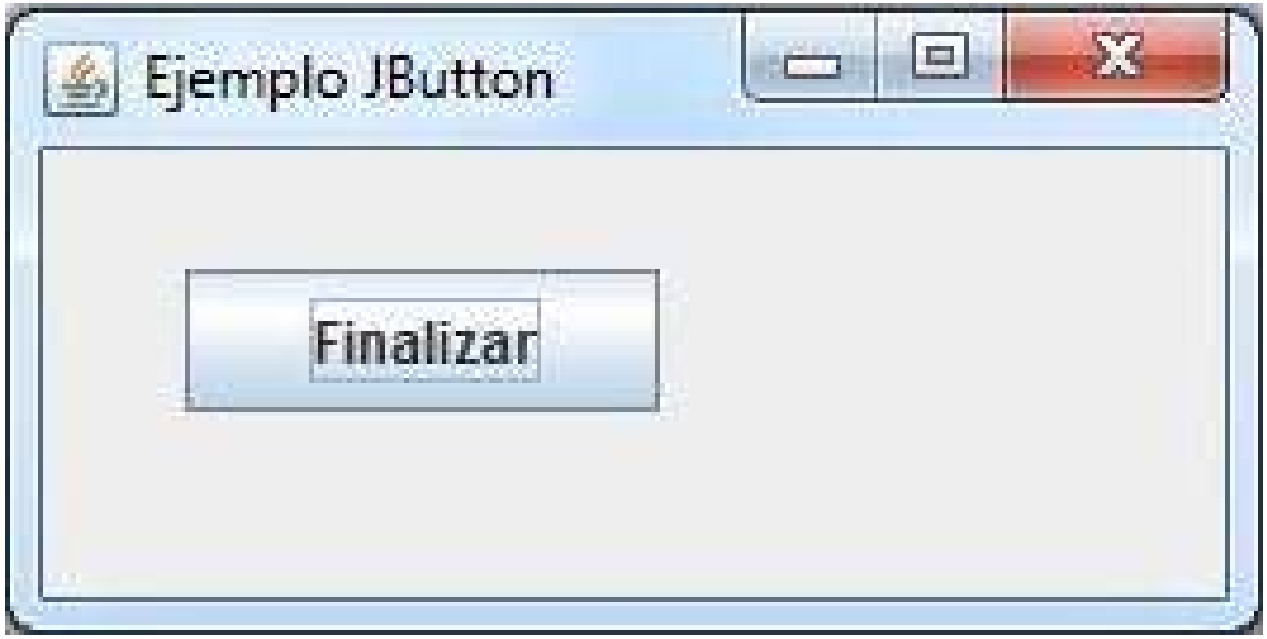


JLabel

La clase JLabel nos permite mostrar un texto.

Ponemos etiquetas.

JButton



JButton

Este control visual muestra un botón.

En este ejemplo veremos la captura de eventos con los controles visuales.

Uno de los eventos más comunes es cuando hacemos clic sobre un botón.

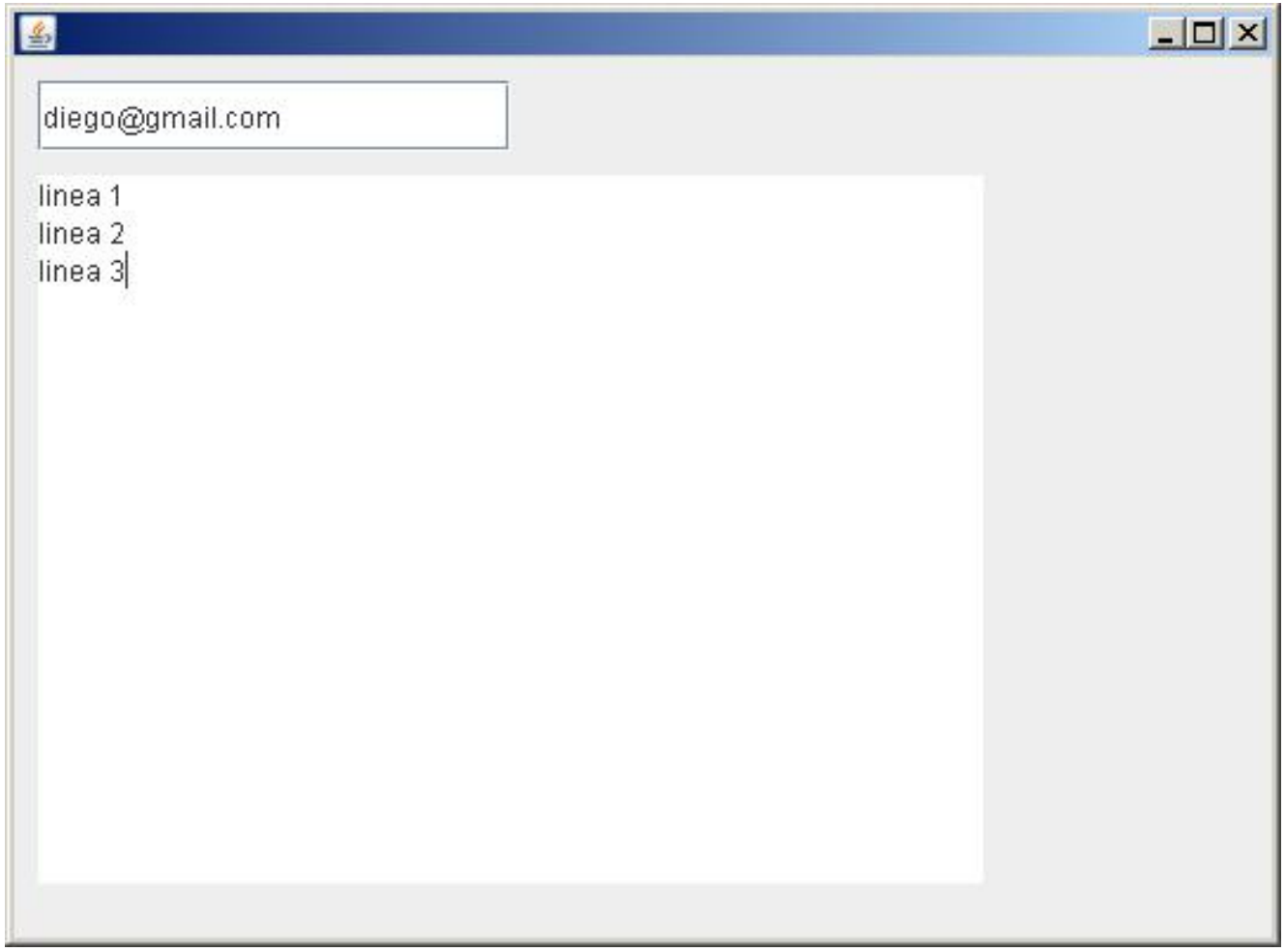
JTextField

JTextField

Así como podríamos decir que el control JLabel remplaza a la salida estándar System.out.print, el control JTextField cumple la función de la clase Scanner para el registro de datos.

El control JTextField permite al operador del programa registrar una cadena de caracteres por teclado.

JTextArea



JTextArea

El control de tipo JTextArea permite ingresar múltiples líneas, a diferencia del control de tipo JTextField.

JComboBox

JComboBox

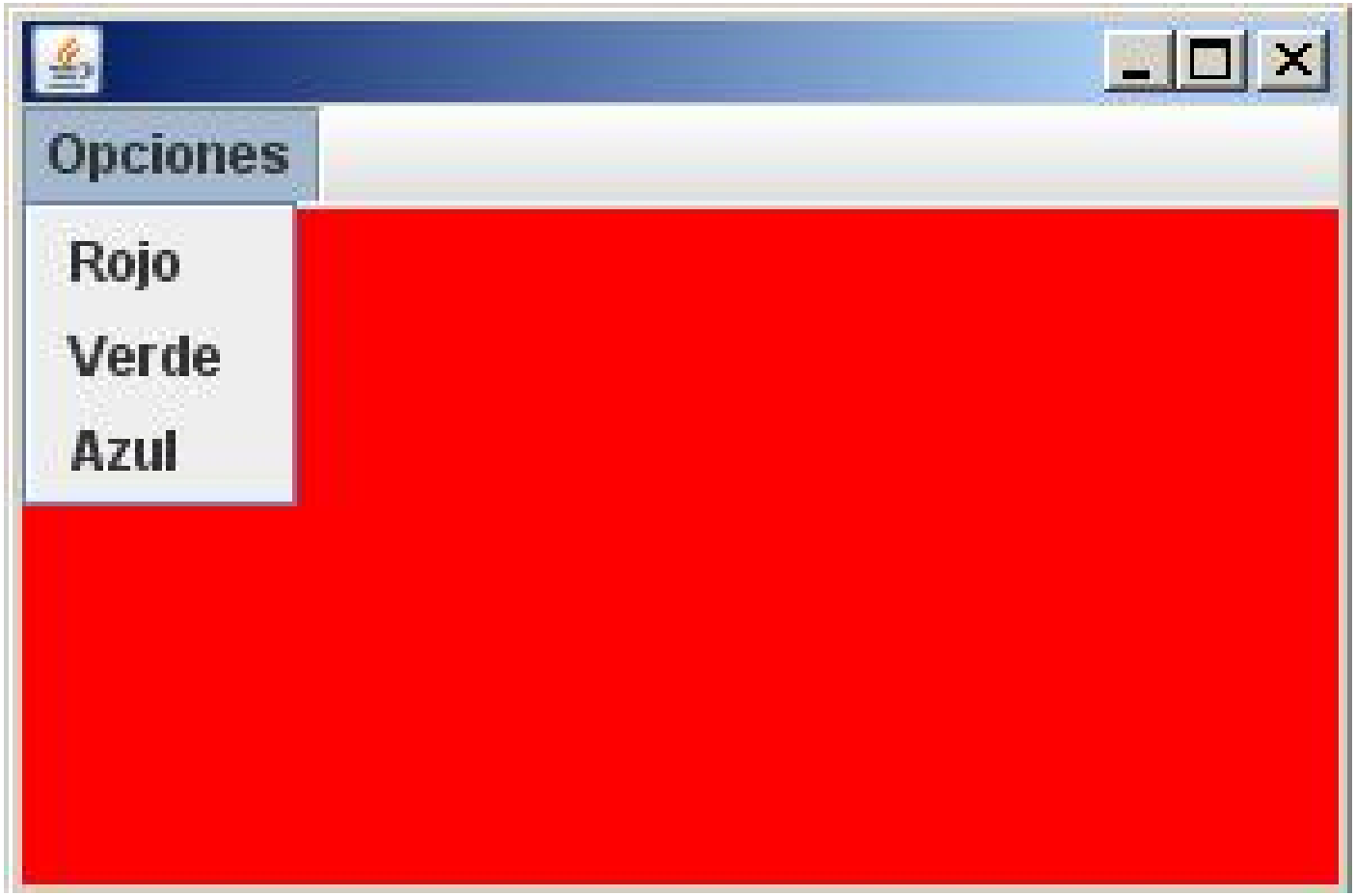
El control JComboBox permite seleccionar un String de una lista.

Para inicializar los String que contendrá el JComboBox debemos llamar al método addItem tantas veces como elementos queremos cargar.

Un evento muy útil con este control es cuando el operador selecciona un Item de la lista.

Para capturar la selección de un item debemos implementar la interface ItemListener que contiene un método llamada itemStateChanged.

JMenuBar, JMenu, JMenuItem



JMenuBar, JMenu, JMenuItem

Lo usamos para crear un menú de opciones y la captura de eventos de los mismos.

Cuando necesitamos implementar un menú horizontal en la parte superior de un JFrame requerimos de un objeto de la clase JMenuBar, uno o más objetos de la clase JMenu y por último objetos de la clase JMenuItem.

Para la captura de eventos debemos implementar la interface ActionListener y asociarlo a los controles de tipo JMenuItem, el mismo se dispara al presionar con el mouse el JMenuItem.

JCheckBox

JCheckBox

El control JCheckBox permite implementar un cuadro de selección.

Básicamente un botón de dos estados.

JRadioButton

JRadioButton

Otro control visual muy común es el JRadioButton que normalmente se muestran un conjunto de JRadioButton y permiten la selección de solo uno de ellos.

Se los debe agrupar para que actúen en conjunto, es decir cuando se selecciona uno automáticamente se deben deseleccionar los otros.

Bordes

Se pueden usar bordes para agrupar componentes o sólo para agregar espacio entre ellos.

Cada componente Swing puede tener un borde.

Algunos gestores de disposición también aceptan parámetros en el constructor que definen sus espacios.

Luego, el gestor de disposición se encarga de crear el espacio requerido entre los componentes.

Los bordes más usados son:

- BevelBorder.
- CompoundBorder.
- EmptyBorder.
- EtchedBorder.
- TitledBorder.

Podemos hacer tres cosas para mejorar el aspecto de nuestra GUI:

- Agregar espacio alrededor de la parte exterior de la ventana.
- Agregar espacio entre los componentes de la ventana.
- Agregar una línea alrededor de la imagen.

Gestores de Disposición

La forma en que los componentes se distribuyen por la ventana viene dada por un ***distribuidor*** (*Layout*).

Los gestores se adaptan a las dimensiones de la ventana, de manera que si ésta se redimensiona, el distribuidor redistribuye los componentes de acuerdo a unas reglas.

Cualquier ***Container*** tiene operaciones para:

- Indicar cuál es su distribuidor. **setLayout().**
- Para añadir componentes. **add().**

Los tipos de *Layout* que hay son los siguientes:

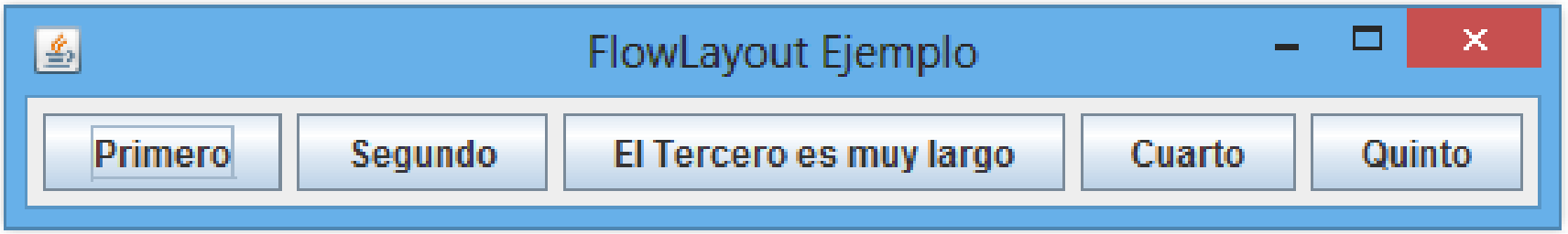
- FlowLayout.
- BorderLayout.
- GridLayout.
- BoxLayout.

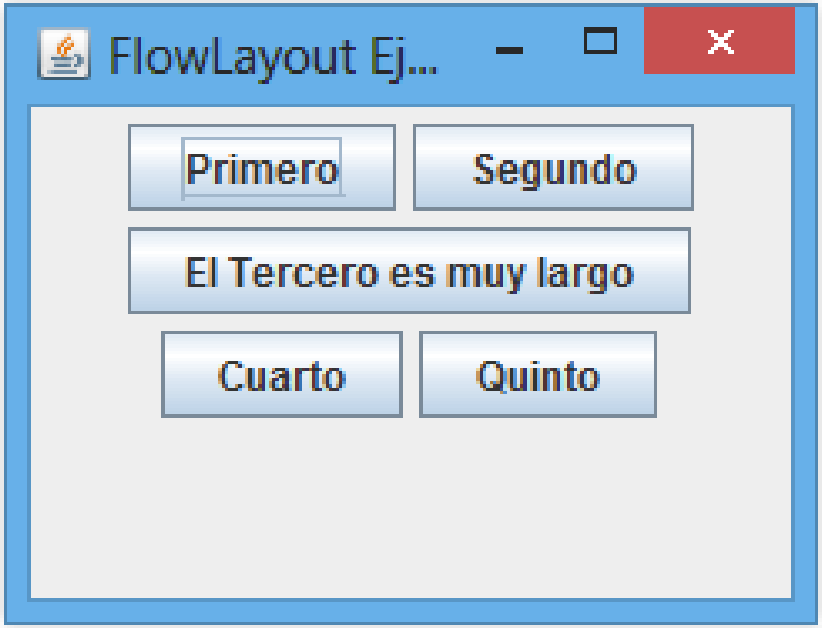
FlowLayout

Es el administrador de disposición más simple que proporciona Java y es el que se proporciona por defecto en los paneles **`JPanel`**.

Este administrador va colocando los componentes de izquierda a derecha, y de arriba abajo

FlowLayout


FlowLayout Ejemplo
Primero
Segundo
El Tercero es muy largo
Cuarto
Quinto

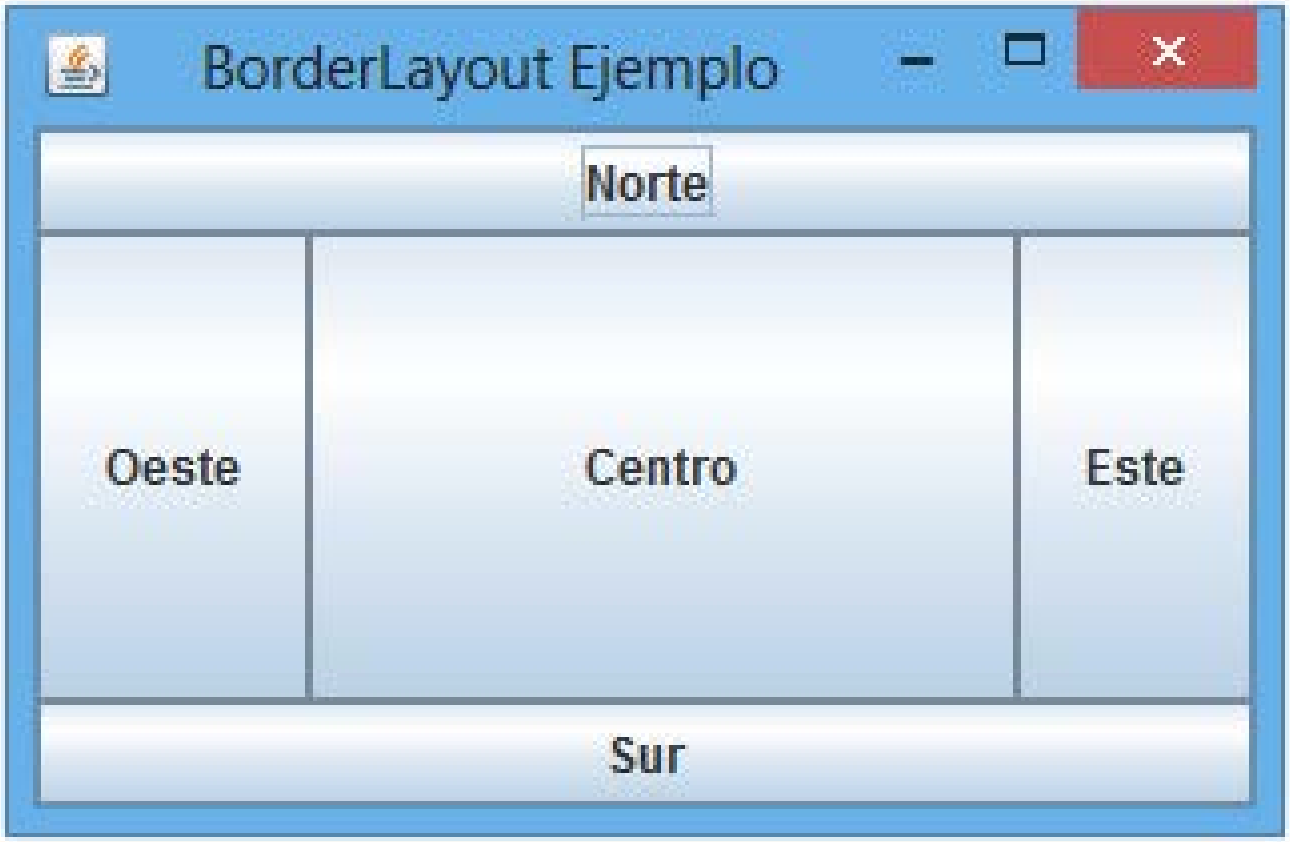

FlowLayout Ej...
Primero
Segundo
El Tercero es muy largo
Cuarto
Quinto

BorderLayout

Este administrador de disposición divide el contenedor en cinco zonas:

- Una central.
- Norte.
- Sur.
- Este.
- Oeste.

BorderLayout



GridLayout

Este administrador de disposición va colocando los componentes de izquierda a derecha, y de arriba abajo, según una matriz de celdas cuyo tamaño se especifica en el constructor.

GridLayout


GridLayout Ejemplo
Primero
Segundo
El Tercero es muy largo
Cuarto
Quinto
Sexto

BoxLayout

El **BoxLayout** permite tanto orientación horizontal como vertical.

El constructor del **BoxLayout** es más complejo que el del **FlowLayout**.

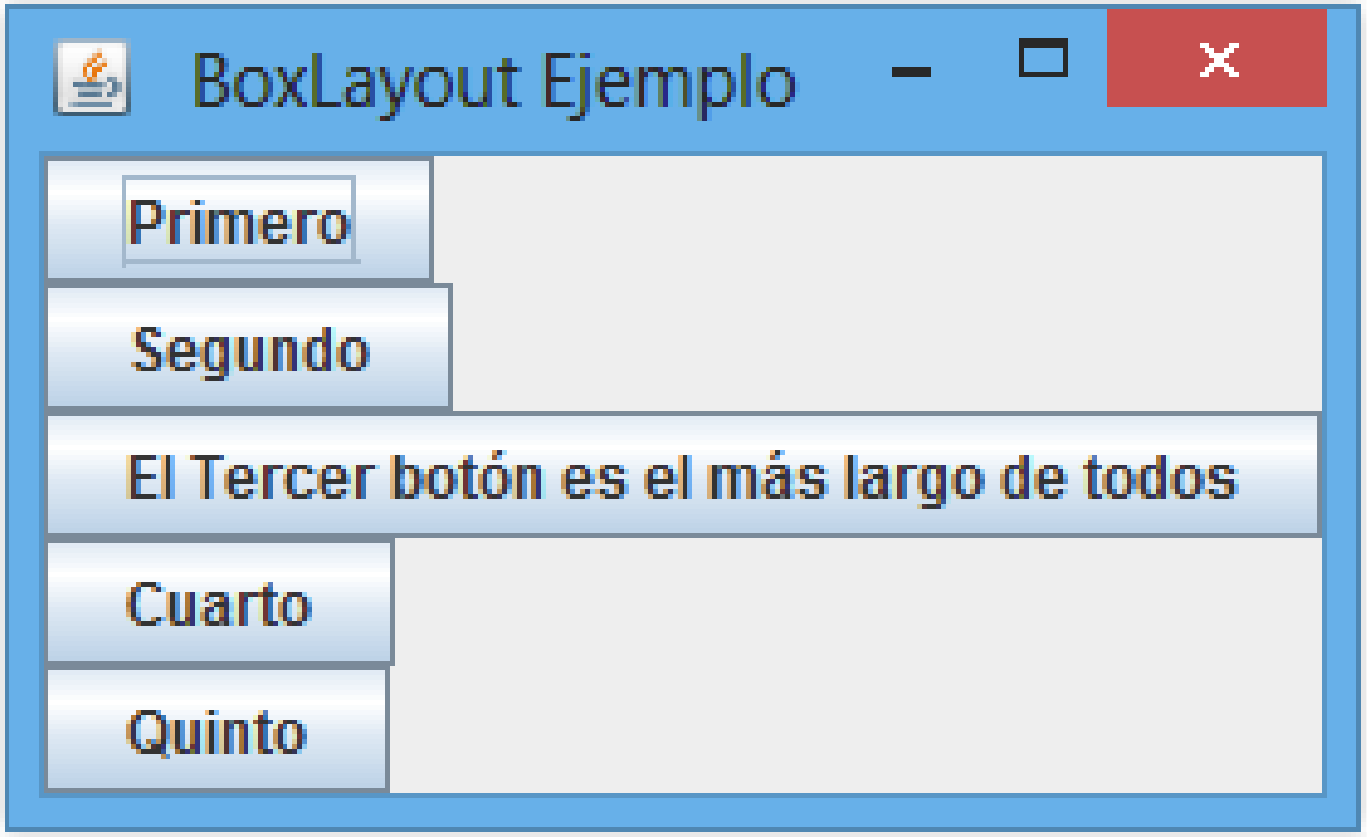
Podemos distribuir los componentes arriba, abajo, izquierda o derecha.

Debemos pasarle el contenedor al que lo estamos añadiendo, es decir, el parámetro panelIzquierdo.

También debemos pasarle si queremos:

- Orientación vertical **BoxLayout.Y_AXIX.**
- Orientación horizontal **BoxLayout.X_AXIS**.

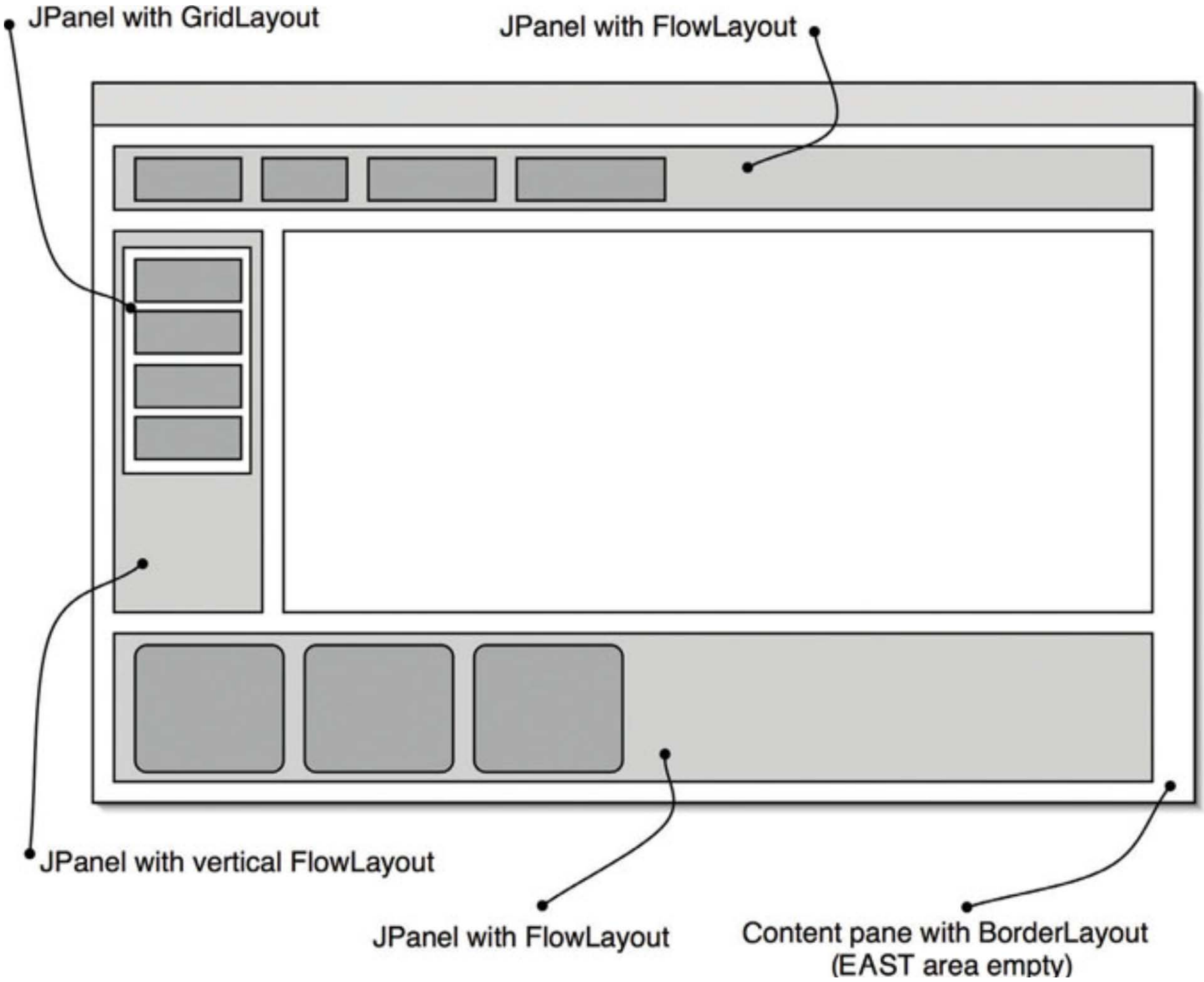
BoxLayout


BoxLayout Ejemplo
Primero
Segundo
El Tercer botón es el más largo de todos
Cuarto
Quinto

Contenedores Anidados

Para conseguir diseños sofisticados lo mejor es anidar contenedores:

- Se puede utilizar JPanel como contenedor básico.
- Cada contenedor tendrá su layout manager específico.


JPanel with GridLayout
JPanel with FlowLayout
JPanel with vertical FlowLayout
JPanel with FlowLayout
Content pane with BorderLayout
(EAST area empty)

La Clase JOptionPane

Un cuadro de diálogo es una ventana que nos permite mostrar mensajes.

Tipos de mensajes:

- De error.
- De advertencia.
- De información.
- Para pedir el ingreso de un valor.
- Permite solicitar al usuario su intervención para decidir si se realizará o no una acción.

JOptionPane es una clase de la biblioteca Swing.

Contiene las librerías de interfaz gráfica de usuario.

Para poder usar sus métodos es necesario importarla:

```
import javax.swing.JOptionPane;
```

JOptionPane tiene básicamente cuatro métodos, que definen la manera y la funcionalidad con la que se mostrará un cuadro de diálogo:

- **`showMessageDialog`.**
- **`showConfirmDialog`.**
- **`showInputDialog`.**
- **`showOptionDialog`.**

Métodos de JOptionPane

Método
int showMessageDialog(Component, Object) int showMessageDialog(Component, Object, String, int) int showMessageDialog(Component, Object, String, int, Icon)
int showConfirmDialog(Component, Object) int showConfirmDialog(Component, Object, String, int) int showConfirmDialog(Component, Object, String, int, int) int showConfirmDialog(Component, Object, String, int, int, Icon)
String showInputDialog(Object) String showInputDialog(Object, Object) String showInputDialog(Component, Object) String showInputDialog(Component, Object, Object) String showInputDialog(Component, Object, String, int) String showInputDialog(Component, Object, String, int, Icon, Object[], Object)
int showOptionDialog(Component, Object, String, int, int, Icon, Object[], Object)

Métodos de JOptionPane

Método	Propósito
showMessageDialog	Muestra un diálogo modal con un botón.
showConfirmDialog	Muestra un diálogo modal que elaborar.
showInputDialog	Muestra un diálogo de entrada.
showOptionDialog	Muestra un diálogo.

showMessageDialog

Muestra un cuadro de diálogo al usuario.

Normalmente de carácter informativo.

Parámetros:

(Component, Object)

(Component, Object, String, int)

(Component, Object, String, int, Icon)

Component componentePadre: el componente al que pertenece.

Object mensaje: es el objeto que corresponde al mensaje de texto que se muestra.

String título: texto que será el título del cuadro de diálogo.

int tipoDeMensaje: definido por una constante de la clase JOptionPane.

Tipos de Mensaje

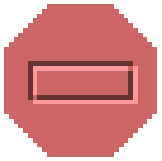
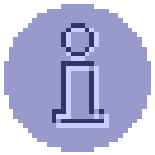
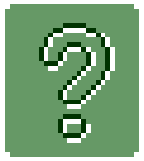
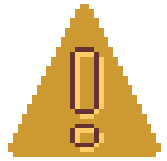
De error (ERROR_MESSAGE)

De tipo informativo (INFORMATION_MESSAGE)

Mensaje plano (PLAIN_MESSAGE)

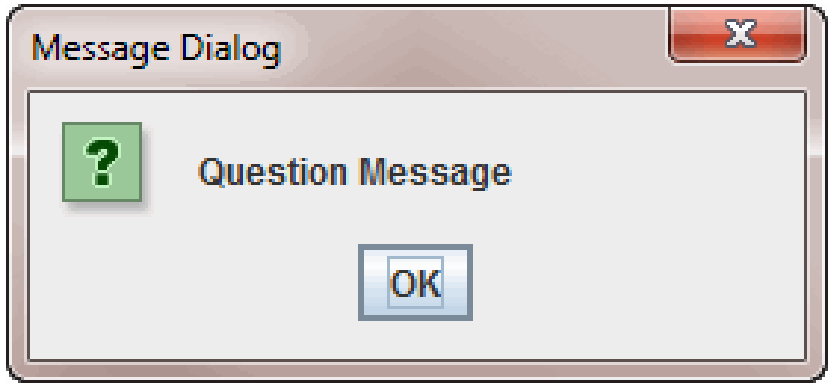
Mensaje interrogativo (QUESTION_MESSAGE)

De advertencia (WARNING_MESSAGE)

Iconos proporcionados por JOptionPane Aspecto y Comportamiento Java			
			
error	information	question	warning


Message Dialog
Error Message
OK


Message Dialog
Information Message
OK


Message Dialog
Question Message
OK


Message Dialog
Warning Message
OK

Icono: la imagen que acompañará al mensaje, si no se especifica (es decir, se pasa **null**) se establecerá uno por defecto de acuerdo al **tipoDeMensaje.**

showConfirmDialog

Sirve para realizar una pregunta al usuario con las posibilidades básicas de respuesta de sí, no o cancelar.

Parámetros:

```
showConfirmDialog(Component,Object)
showConfirmDialog(Component,Object,String,int)
showConfirmDialog(Component,Object,String,int,int)
showConfirmDialog(Component,Object,String,int,int,Icon)
```

showInputDialog

Sirve para mostrar una ventana que permita ingresar datos.

Parámetros:

```
showInputDialog(Object)
showInputDialog(Object, Object)
showInputDialog(Component, Object)
showInputDialog(Component, Object, Object)
showInputDialog(Component, Object, String, int)
showInputDialog(Component, Object, String, int, Icon,
                Object[], Object)
```

Cuando lo que se va a ingresar es un número, se debe realizar una conversión para poder utilizarlo como tal.

```
int numero =
Integer.parseInt(JOptionPane.showInputDialog(
this,
"Ingrese un número:",
"showInputDialog",
JOptionPane.INFORMATION_MESSAGE));
```

Para que no arroje errores en la conversión, podríamos solucionar tan sólo encerrándolo en un **`try-catch.`**

showOptionDialog

Con este método podemos obtener un cuadro de diálogo ajustado a nuestra necesidad.

Permite crear una ventana personalizada de cualquiera de los tipos anteriores.

Parámetros:

(Component, Object, String, int, int, Icon, Object[], Object)

componentePadre: el objeto que indica de qué componente es hijo.

objetoMensaje: un String que corresponde al texto a mostrarse como mensaje.
Titulo = String que se establecerá como titulo de la ventana.

TipoDeOpcion: es un entero, representado por unas constantes que definen qué opciones tendrá el cuadro de diálogo.

Tipo de Opción

DEFAULT_OPTION

YES_NO_OPTION

YES_NO_CANCEL_OPTION

OK_CANCEL_OPTION

tipoDeMensaje: es un entero, definido por una constante de la clase JOptionPane.

Tipos de Mensaje

De error (ERROR_MESSAGE)

De tipo informativo (INFORMATION_MESSAGE)

Mensaje plano (PLAIN_MESSAGE)

Mensaje interrogativo (QUESTION_MESSAGE)

De advertencia (WARNING_MESSAGE)

Icono: la imagen que acompañará al mensaje, si no se especifica (es decir, se pasa **null**) se establecerá uno por defecto de acuerdo al **tipoDeMensaje.**

Opciones: un array tipo **Object** que indica las opciones posibles, normalmente debe ser coherente con el **tipoDeOpcion** elegido.

ValorInicial: es la opción predeterminada, deberá ser una de las opciones introducidas en el array de opciones. Puede ser null.