

Desarrollo de Interfaces

Interfaces de Usuario

Swing I

Las interfaces gráficas de usuario (GUI) ofrecen al usuario ventanas, cuadros de diálogo, barras de herramientas, botones, listas desplegables y muchos otros elementos con los que ya estamos muy acostumbrados a tratar.

Las aplicaciones son conducidas por eventos y se desarrollan haciendo uso de las clases que para ello nos ofrece la API de Java.

La interfaz de usuario es la parte del programa que permite al usuario interactuar con él.

La API de Java proporciona una biblioteca de clases para el desarrollo de Interfaces gráficas de usuario (en realidad son dos).

Swing II

La biblioteca proporciona un conjunto de herramientas para la construcción de interfaces gráficas que tienen una apariencia y se comportan de forma semejante en todas las plataformas en las que se ejecuten.

La estructura básica de la biblioteca gira en torno a componentes y contenedores. Los contenedores contienen componentes y son componentes a su vez, de forma que los eventos pueden tratarse tanto en contenedores como en componentes.

La API está constituida por clases, interfaces y derivaciones: AWT y Swing,

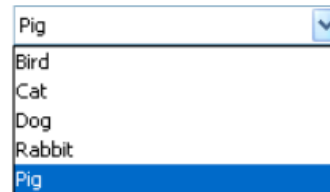
Componentes Swing I



[JButton](#)



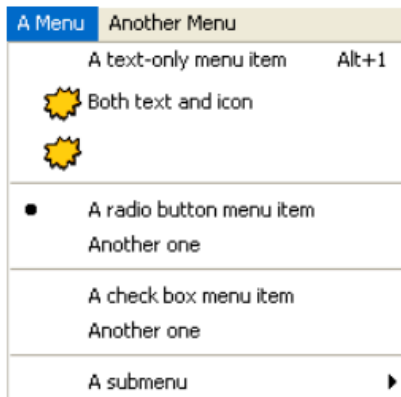
[JCheckBox](#)



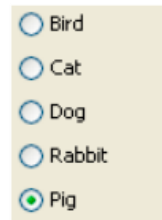
[JComboBox](#)



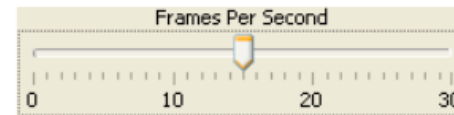
[JList](#)



[JMenu](#)



[JRadioButton](#)



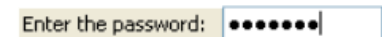
[JSlider](#)



[JSpinner](#)



[JTextField](#)



[JPasswordField](#)

Jerarquía de clases para las GUI I

Component: superclase de todas las clases de interfaz gráfica.

Container: para agrupar componentes.

JComponent: superclase de todos los componentes de Swing que se dibujan directamente en los lienzos (canvas).

Sus subclases son los elementos básicos de la GUI.

JFrame: ventana que no está contenida en otras ventanas.

JDialog: cuadro de diálogo.

JApplet: subclase de Applet para crear applets tipo Swing.

JPanel: contenedor invisible que mantiene componentes de interfaz y que se puede anidar, colocándose en otros paneles o en ventanas. También sirve de lienzo.

Graphics: clase abstracta que proporciona contextos gráficos donde dibujar cadenas de texto, líneas y otras formas sencillas.

Jerarquía de clases para las GUI II

Color: color de los componentes gráficos.

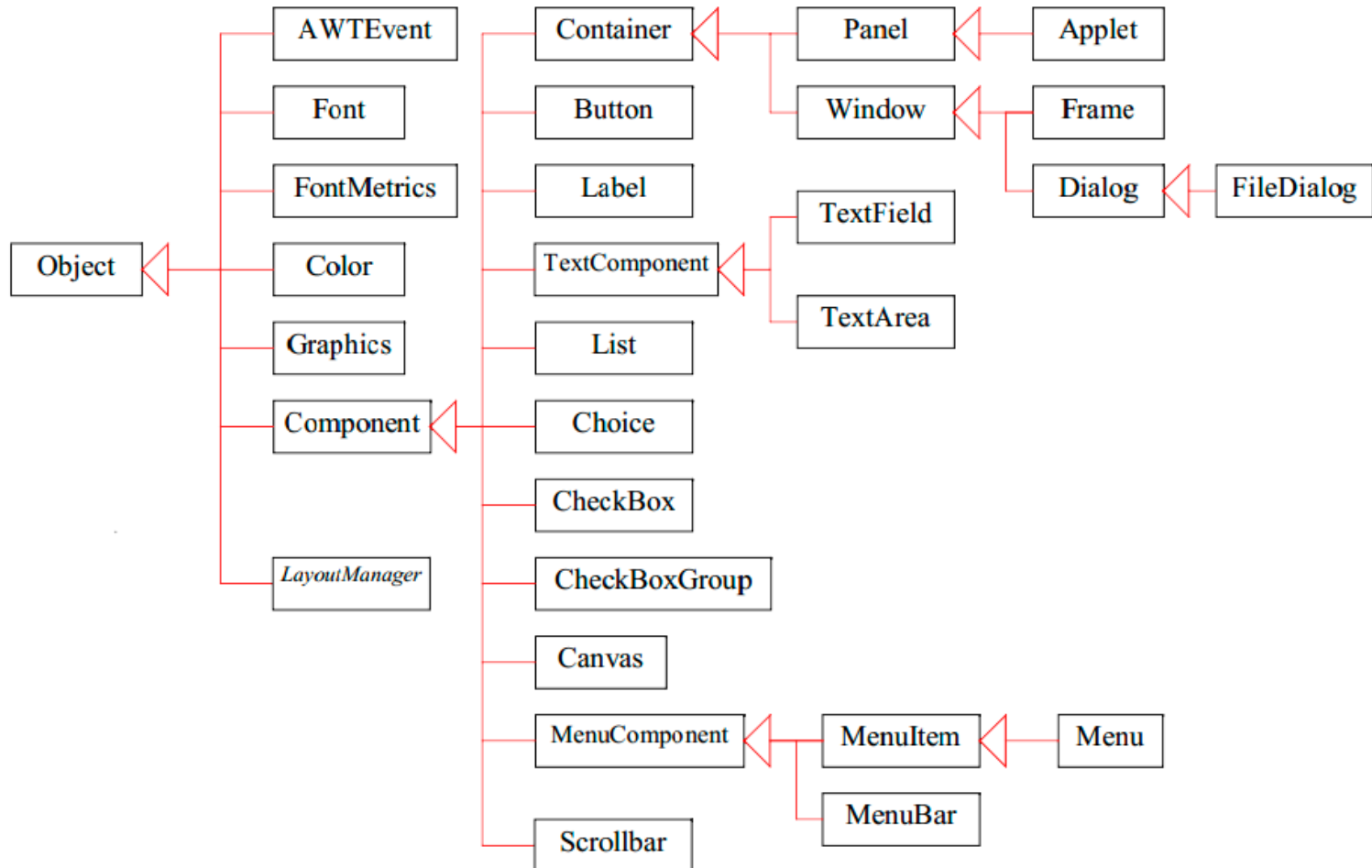
Font: aspecto de los caracteres.

FontMetrics: clase abstracta para propiedades de las fuentes.

Categorías de clases:

- ✓ Contenedores:
 - ✓ JFrame, JApplet, JWindow, JDialog
- ✓ Componentes intermedios:
 - ✓ JPanel, JScrollPane
- ✓ Componentes:
 - ✓ JLabel, JButton, JTextField, JTextArea, ...
- ✓ Clases de soporte:
 - ✓ Graphics, Color, Font, ...

Jerarquía de clases para las GUI: Jcomponent II



Jerarquía de componentes I

Graphics (java.awt)

Component (funcionalidad básica de componentes gráficos)

Button, Canvas, CheckBox, Choice, Label, List, ScrollBar

TextComponent

TextField, TextArea

Container (permite agrupar, añadir y eliminar componentes)

(también definir diseño o disposición (layout managers))

ScrollPane

Panel

Applet (java.applet)

JApplet (javax.swing)

Window

Frame

JFrame (javax.swing)

Dialog

FileDialog

JDialog (javax.swing)

JWindow (javax.swing)

JComponent (javax.swing)

Contenedores I

Contenedores de alto nivel:

JFrame

Habitualmente la clase JFrame se emplea para crear la ventana principal de una aplicación en Swing.

JDialog

Ventanas de interacción con el usuario.

Contenedores intermedios:

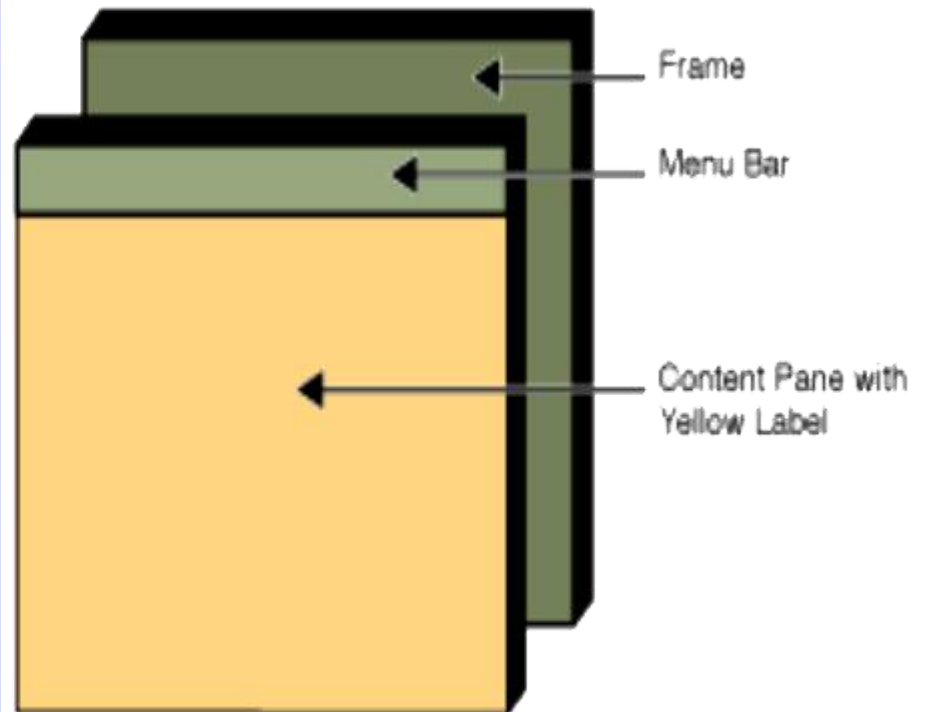
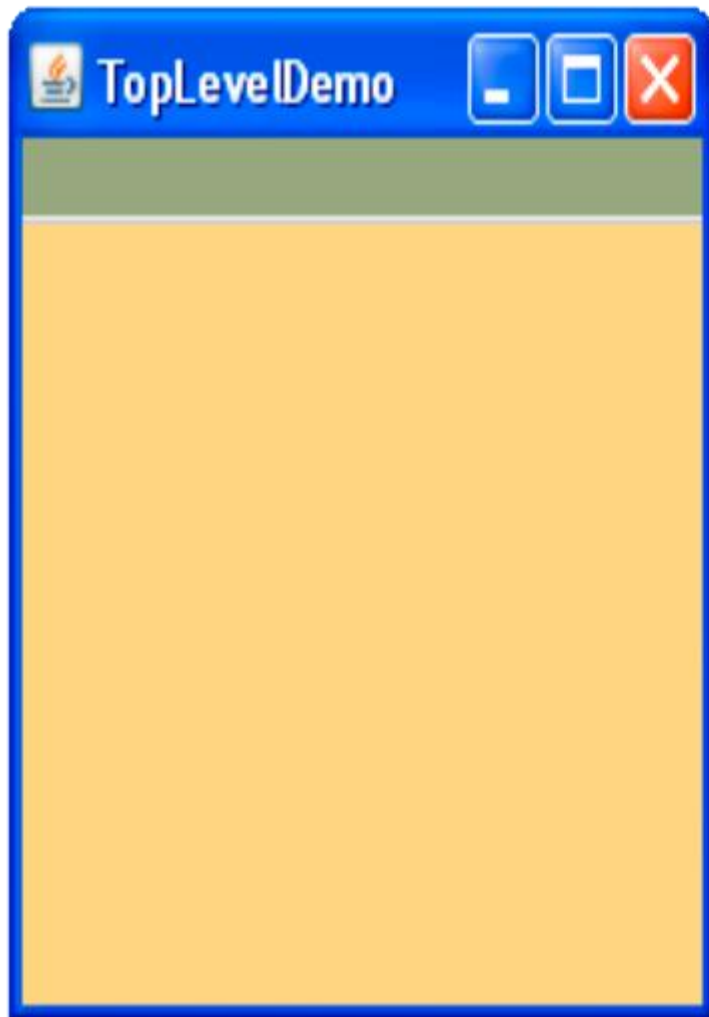
JPanel

Agrupar a otros componentes.

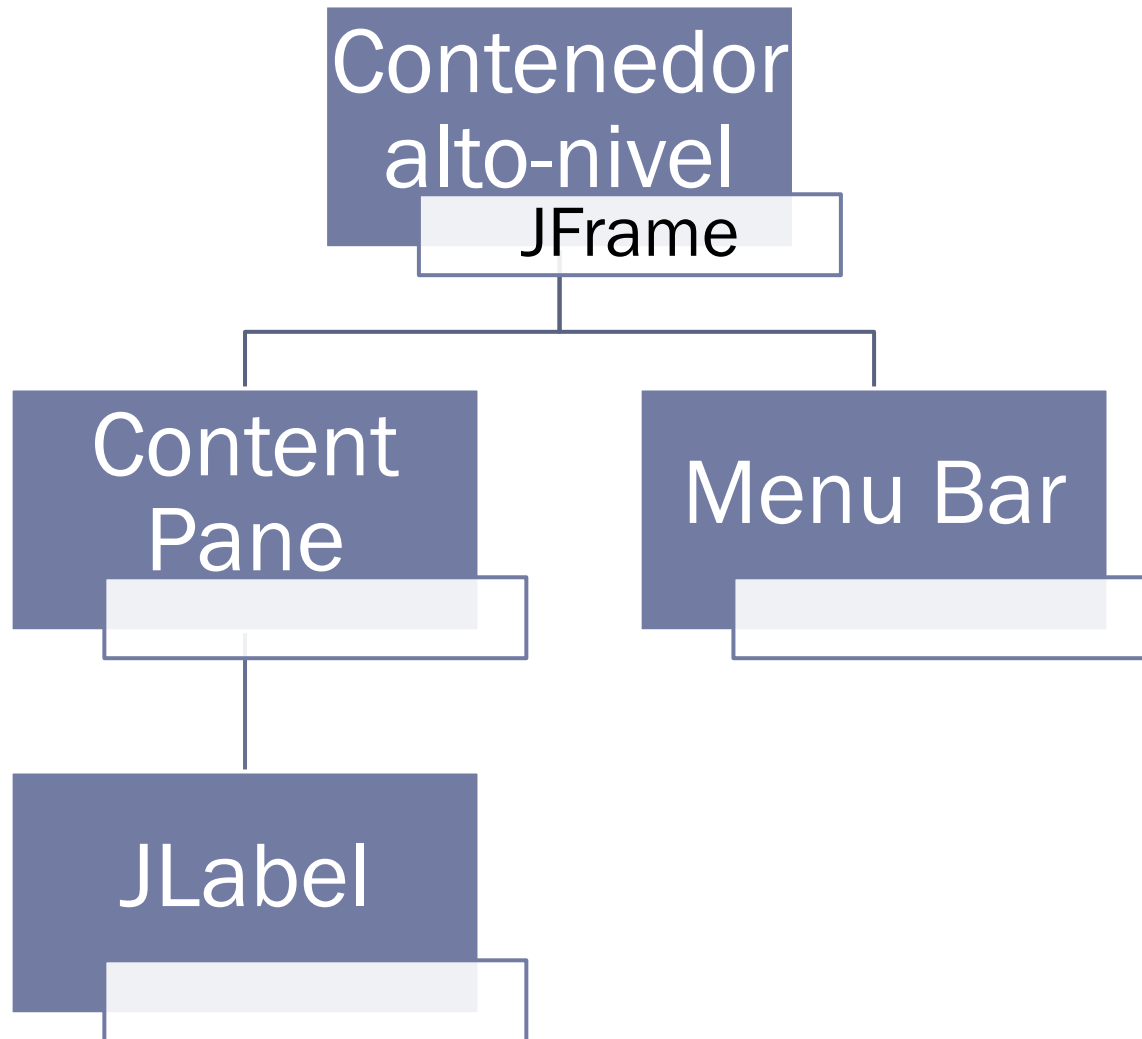
JScrollPane

Incluye barras de desplazamiento.

Contenedores II



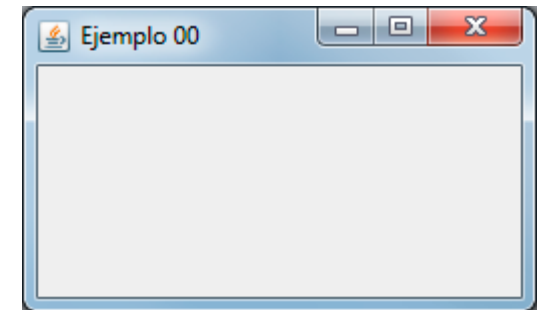
Jerarquía I



Jerarquía II

```
import javax.swing.*;

public class Gui00 extends JFrame {
    // Constantes y componentes (objetos)
    public Gui00() {
        super("Ejemplo 00");
        // Configurar Componentes ;
        // Configurar Manejadores Eventos ;
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    } // Terminar la aplicación al cerrar la ventana.
    public static void main(String args[]) {
        Gui00 aplicacion = new Gui00();
    }
}
```



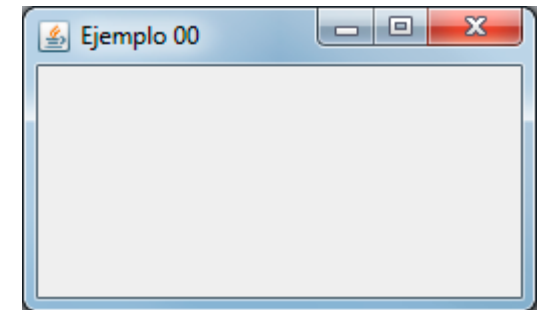
Jerarquía III

```
import javax.swing.*;

public class Gui00 {
    // Constantes y componentes (objetos)
    public Gui00() {
        JFrame frame = new JFrame("Ejemplo 00");
        // Configurar componentes
        // y añadirlos al panel del frame

        . . .
        frame.pack ();

        frame.setVisible (    true);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[]){
        Gui00 aplicacion = new Gui00();
    }
}
```



Añadir elementos

Modo 1:

1. Obtenemos el panel de contenido del frame:

```
Container panel = this.getContentPane();
```

2. Añadimos componentes a dicho panel:

```
panel.add(unComponente);
```

Modo 2:

A partir de 1.5 también se puede hacer directamente sobre el JFrame

```
add(unComponente);
```

1. Con herencia de JFrame utilizando Container

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

public class Gui01 extends JFrame {

    private Container panel;
    private JButton miboton;

    public Gui01() {
        super("Ejemplo 01 con botón");
        // Configurar componentes ;
        miboton = new JButton("Aceptar");
        panel = getContentPane();
        panel.add(miboton);

        setSize(200, 100);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        Gui01 aplicacion = new Gui01();
    }
}
```



2. Con herencia de JFrame sin Container

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

public class Gui01 extends JFrame {

    private JButton miboton;

    public Gui01() {
        super("Ejemplo 01 con botón");
        . . .
        miboton = new JButton("Aceptar");
        add(miboton);
        . . .
        setSize(200, 100);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        Gui01 aplicacion = new Gui01();
    }
}
```



3. Ejemplo sin herencia con Container

```
public class Gui01 {  
  
    private JButton miboton;  
    private Container panel;  
  
    public Gui01() {  
        JFrame frame = new JFrame(  
            "Ejemplo 01");  
        panel = frame.getContentPane();  
        . . .  
        miboton = new JButton("Aceptar");  
        panel.add(miboton);  
        //se añade al contentPane del frame  
        . . .  
        frame.pack();  
        frame.setVisible(true);  
        frame.setDefaultCloseOperation(JFrame.EXIT  
    }  
  
    public static void main(String args[]) {  
        Gui01 aplicacion = new Gui01();  
    }  
}
```



4. Ejemplo sin herencia sin Container

```
public class Gui01 {  
  
    private JButton miboton;  
  
    public Gui01() {  
        JFrame frame = new JFrame(  
            "Ejemplo 01");  
  
        . . .  
        miboton = new JButton("Aceptar");  
  
        frame.add (miboton); //se añade al frame  
        . . .  
        frame.pack ();  
  
        frame.setVisible (true);  
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);  
    }  
    public static void main(String args[]) {  
        Gui01 aplicacion = new Gui01();  
    }  
}
```



Administradores de disposición I

Los componentes se agregan al contenedor con el método `add()`.

```
JButton unBoton = new JButton("Texto del botón");  
panel.add(unBoton);
```

El efecto de `add()` depende del esquema de colocación o disposición (layout) del contenedor que se use.

Existen diversos esquemas de disposición: **FlowLayout**, **BorderLayout**, **GridLayout**, ...

Los objetos contenedores se apoyan en objetos **LayoutManager** (administradores de disposición).

Clases más usadas que implementa la interfaz **LayoutManager**:

FlowLayout: un componente tras otro de izquierda a derecha.

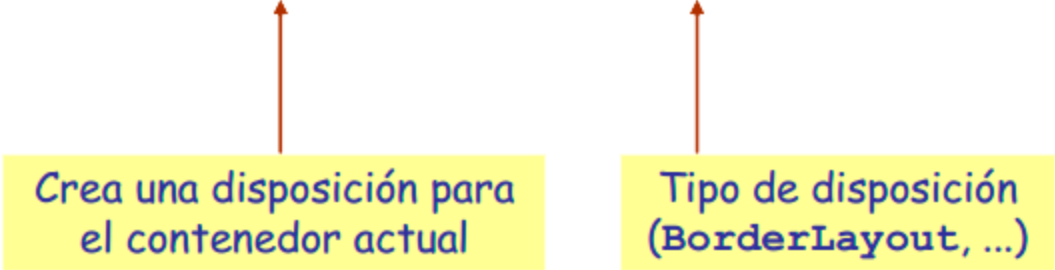
BorderLayout: 5 regiones en el contenedor (North, South, ...).

GridLayout: contenedor en filas y columnas.

Administradores de disposición II

Para organizar el contenedor se utiliza el método `setLayout()`:

```
public void setLayout(LayoutManager mgr)
```



Crea una disposición para el contenedor actual

Tipo de disposición (BorderLayout, ...)

```
setLayout(new BorderLayout());
```

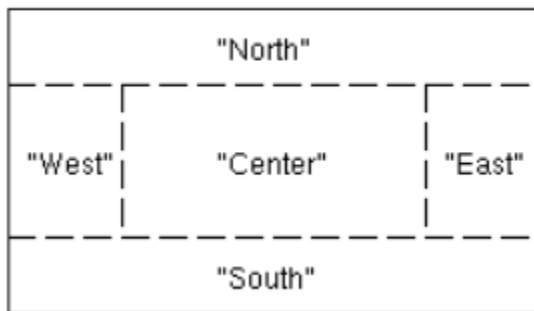
```
setLayout(new FlowLayout());
```

```
setLayout(new GridLayout(3, 4));
```

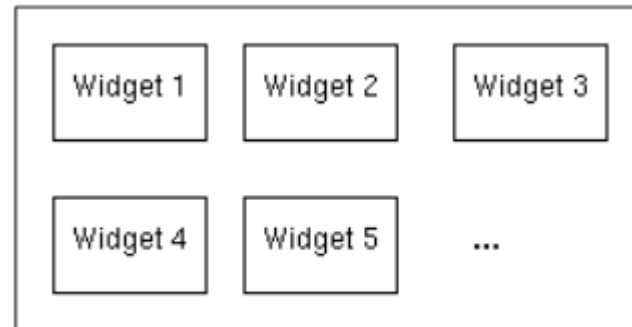
El layout manager elige la mejor posición y tamaño de cada componente de acuerdo al espacio disponible.

Administradores de disposición III

BorderLayout organiza el contenedor en 5 zonas:



FlowLayout coloca los componentes de izquierda a derecha y de arriba hacia abajo:



Para distribuciones más complejas podemos insertar paneles (JPanel) en los contenedores y obtener el tamaño de un componente con el método `getSize()`.

FlowLayout

```
public class Gui02 extends JFrame {

    public Gui02() {
        super("Ejemplo de Layout");
        // Configurar componentes ;
        // Configurar layout ;
        setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
        for (int i = 1; i <= 10; i++) {
            add(new JButton("Componente " + i));
        }
        setSize(200, 200); //pack();
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        Gui02 aplicacion = new Gui02();
    }
}
```

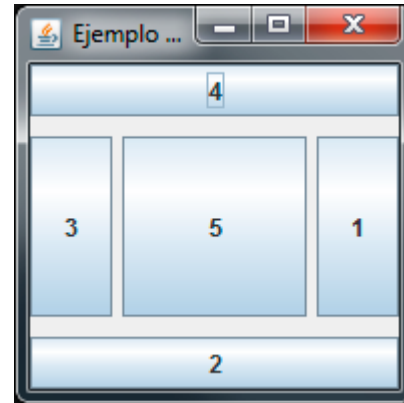


BorderLayout

```
public class Gui03 extends JFrame {

    public Gui03() {
        super("Ejemplo de Layout");
        // BorderLayout
        setLayout(new BorderLayout(5, 10));
        add(new JButton("1"), BorderLayout.EAST);
        add(new JButton("2"), BorderLayout.SOUTH);
        add(new JButton("3"), BorderLayout.WEST);
        add(new JButton("4"), BorderLayout.NORTH);
        add(new JButton("5"), BorderLayout.CENTER);
        setSize(200, 200); //pack();
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        Gui03 aplicacion = new Gui03();
    }
}
```



GridLayout

```
setLayout(new GridLayout(filas, columnas))
```

Crea una zona de filas x columnas componentes y éstos se van acomodando de izquierda a derecha y de arriba a abajo.

GridLayout tiene otro constructor que permite establecer la separación (en pixels) entre los componentes, que es cero con el primer constructor.

Así, por ejemplo:

```
new GridLayout(3, 4, 2, 2)
```

crea una organización de 3 filas y 4 columnas donde los componentes quedan a dos pixels de separación.

Ejemplo:

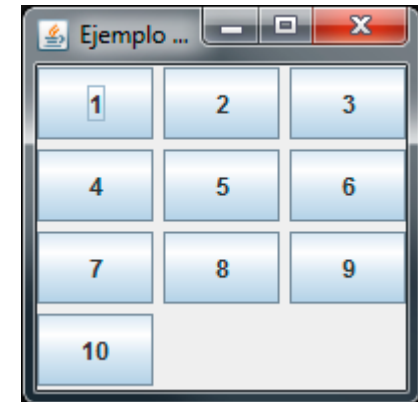
```
setLayout(new GridLayout(3, 4, 2, 2);  
for(int i = 0; i < 3 * 4; i++) {  
    add(new JButton(Integer.toString(i + 1)));  
}
```

GridLayout

```
public class Gui03b extends JFrame {

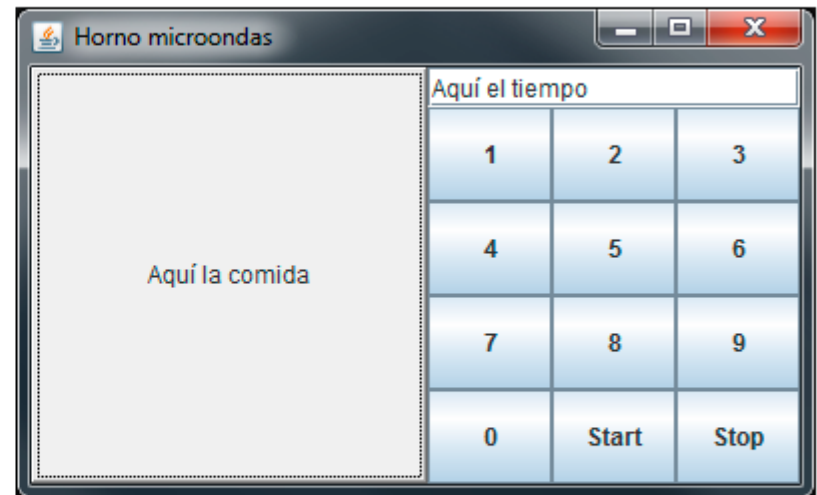
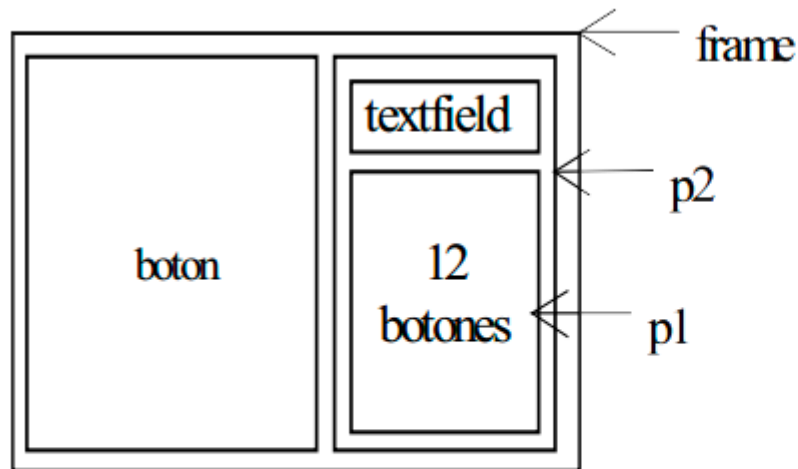
    public Gui03b() {
        super("Ejemplo de Layout");
        setLayout(new GridLayout(4, 3, 5, 5));
        for (int i = 1; i <= 10; i++) {
            add(new JButton(Integer.toString(i)));
        }
        setSize(200, 200);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        Gui03b aplicacion = new Gui03b();
    }
}
```



Paneles como contenedores I

Los paneles actúan como pequeños contenedores para agrupar componentes. Colocamos los componentes en paneles y los paneles en el frame o incluso en otros paneles.



Paneles como contenedores II

```
public class Gui04 extends JFrame {  
  
    public Gui04() {  
        setTitle("Horno microondas");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(new BorderLayout());  
        // Create panel p1 for the buttons and set GridLayout  
        JPanel p1 = new JPanel();  
        p1.setLayout(new GridLayout(4, 3));  
        // Add buttons to the panel  
        for (int i = 1; i <= 9; i++) {  
            p1.add(new JButton("" + i));  
        }  
        p1.add(new JButton("" + 0));  
        p1.add(new JButton("Start"));  
        p1.add(new JButton("Stop"));  
    }  
}
```

Paneles como contenedores III

```
// Create panel p2 to hold a text field and p1
JPanel p2 = new JPanel();
p2.setLayout(new BorderLayout());
p2.add(new JTextField("Aquí el tiempo"),
        BorderLayout.NORTH);
p2.add(p1, BorderLayout.CENTER);
// Add p2 and a button to the frame
add(p2, BorderLayout.EAST);
add(new Button("Aquí la comida"),
        BorderLayout.CENTER);
setSize(400, 250);
setVisible(true);
}

public static void main(String[] args) {
    Gui04 frame = new Gui04();
}
}
```

Dibujo de gráficos en paneles

JPanel se puede usar para dibujar.

Para dibujar en un panel se crea una clase derivada de JPanel y se redefine el método `paintComponent()` que le indica al panel como dibujar.

La clase ***Graphics*** es una clase abstracta para todos los contextos gráficos.

Una vez obtenido el contexto gráfico podemos llamar desde este objeto a las funciones gráficas definidas en la clase Graphics.

Graphics contiene información acerca de la zona que necesita ser redibujada: el objeto donde dibujar, un origen de traslación, el color actual, la fuente actual, etcétera.

Ejemplo de dibujo

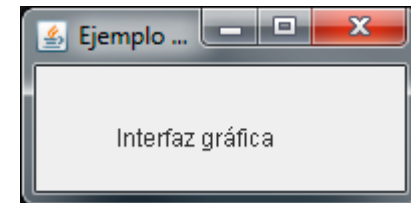
```
import javax.swing.*;
import java.awt.*;

class MiPanel extends JPanel {

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("Interfaz gráfica", 40, 40);
    }
}

public class Gui05 extends JFrame {

    public Gui05() {
        super("Ejemplo de dibujo");
        add(new MiPanel());
        setSize(200, 100);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```



Ejemplo de dibujo

Cuando utilizamos el método `paintComponent()` para dibujar en un contexto gráfico `g`, ese contexto es un ejemplar de una subclase concreta de la clase `Graphics` para la plataforma específica.

El método **`paintComponent()`** es llamado la primera vez y cada vez que es necesario redibujar el componente.

Al hacer **`super.paintComponent(g)`** nos aseguramos de que el área visualizada se limpia antes de volver a dibujar.

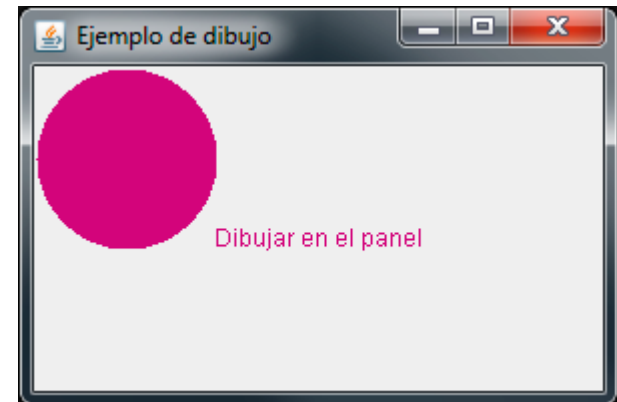
```
class MiPanel extends JPanel {  
    public void paintComponent(Graphics g) {  
        super.paintComponent(g);  
        g.drawString("Interfaz gráfica", 40, 40);  
    }  
}
```


Ejemplo de dibujo

```
class MiPanel2 extends JPanel {  
  
    public void paintComponent(Graphics g) {  
        Color c = new Color(180, 10, 120);  
        g.setColor(c);  
        g.drawString("Dibujar en el panel", 90, 90);  
        g.fillOval(1, 1, 90, 90);  
    }  
}
```

← Creación de un color RGB

```
public class Gui06 extends JFrame {  
  
    public Gui06() {  
        super("Ejemplo de dibujo");  
        add(new MiPanel2());  
        setSize(300, 200);  
        setVisible(true);  
        setDefaultCloseOperation(EXIT_ON_CLOSE);  
    }  
}
```



Algunos métodos de Graphics

```
Graphics g;  
...  
    g.setColor(Color.blue);  
    g.setBackground(Color.red);  
    g.drawLine(int x1, int y1, int x2, int y2);  
    g.drawRect(int x, int y, int ancho, int alto);  
    g.drawRoundedRect(int x, int y, int ancho, int alto,  
        int arcWidth, int arcHeight);  
    g.fillRect(int x, int y, int ancho, int alto);  
    g.fillRoundedRect(int x, int y, int ancho, int alto,  
        int arcWidth, int arcHeight);  
    g.drawOval(int x, int y, int ancho, int alto);  
    g.fillOval(int x, int y, int ancho, int alto);  
    g.drawArc(int x, int y, int ancho, int alto, int angl,  
        int ang2);  
    g.drawString(String cadena, int x, int y);  
    g.setFont(Font f);  
...  
|
```

Interacción con el usuario I

Al interactuar con la aplicación, el usuario:

- Acciona componentes (ActionEvent).
 - El usuario pulsa un botón.
 - El usuario termina de introducir un texto en un campo y presiona Intro.
 - El usuario selecciona un elemento de una lista pulsando el preferido (o de un menú).
 - Pulsa o suelta botones del ratón (MouseEvent).
- Minimiza, cierra o manipula una ventana (WindowEvent).
- Escribe con el teclado (KeyEvent).
- Descubre porciones de ventanas (PaintEvent)

Interacción con el usuario II

Cuando el usuario de un programa o applet mueve el ratón, presiona un pulsador o pulsa una tecla, genera un evento (actionEvent).

Los eventos son objetos de ciertas clases. Normalmente un objeto de alguna subclase de EventObject que indica:

- ✓ El elemento que accionó el usuario.
- ✓ La identificación del evento que indica la naturaleza del evento.
- ✓ La posición del ratón en el momento de la interacción.
- ✓ Teclas adicionales pulsadas por el usuario, como la tecla
- ✓ Control, la tecla de Cambio a mayúsculas, etcétera.

Acciones del usuario

Acción	Objeto origen	Tipo de evento
Pulsar un botón	JButton	ActionEvent
Cambio del texto	JTextComponent	TextEvent
Pulsar Intro en un campo de texto	JTextField	ActionEvent
Selección de un nuevo elemento	JComboBox	ItemEvent ActionEvent
Selección de elemento(s)	JList	ListSelection- Event
Pulsar una casilla de verificación	JCheckBox	ItemEvent ActionEvent
Pulsar un botón de radio	JRadioButton	ItemEvent ActionEvent
Selección de una opción de menú	JMenuItem	ActionEvent
Mover la barra de desplazamiento	JScrollBar	AdjustmentEvent
Abrir, cerrar, minimizar, maximizar o cerrar la ventana	JWindow	WindowEvent
...		

Modelo para responder a eventos I

Creamos un manejador de eventos en dos pasos:

- ▶ Definimos una clase específica que haga de oyente de eventos y que implemente el método `actionPerformed()`.

```
class MiOyente implements ActionListener {  
    public void actionPerformed() {  
        // Aquí se responde el evento  
    }  
}
```

- ▶ Registramos un ejemplar como oyente de componentes:

```
componente.addActionListener(ejemplar_de_MiOyente);
```

Modelo para responder a eventos II

```
class MiGui extends JFrame {  
    ...  
    public void MiGui() {  
        // Registro los componentes interesados  
        // en responder a eventos ...  
        componente.addActionListener(new MiOyente());  
    }  
    ...  
}
```

```
class MiOyente implements ActionListener {  
    public actionPerformed(ActionEvent e) {  
        ... // Aqui se responde el evento  
    }  
}
```

Se añade el oyente especificado (ejemplar de MiOyente) a la lista de oyentes para recibir eventos de acción (ActionEvent) desde ese componente.

Ejemplo: botón con pitido

```
public class Gui09 extends JFrame {

    JButton boton;

    public Gui09() {
        boton = new JButton("Pulsa!");
        add(boton);
        boton.addActionListener(new OyenteBoton());
        setSize(100, 100);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        Gui09 aplicacion = new Gui09();
    }
}

class OyenteBoton implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
    }
}
```

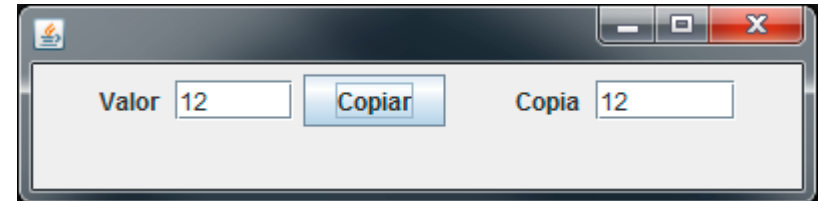


Ejemplo: reproducir un valor

```
public class Gui10 extends JFrame {

    JButton botonCopiar;
    JTextField campoValor, resultado;

    public Gui10() {
        setLayout(new FlowLayout());
        add(new JLabel("Valor "));
        campoValor = new JTextField(5);
        add(campoValor);
        botonCopiar = new JButton("Copiar");
        add(botonCopiar);
        botonCopiar.addActionListener(new OyenteBoton());
        add(new JLabel("Copia "));
        resultado = new JTextField(6);
        add(resultado);
        setSize(400, 100);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```



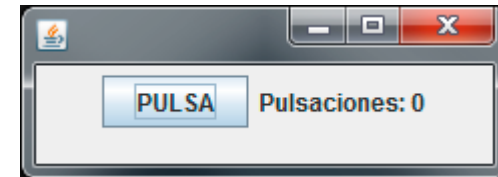
Ejemplo: reproducir un valor

```
1 public static void main(String args[]) {  
2     Gui10 ventana = new Gui10();  
3 }  
  
4 class OyenteBoton implements ActionListener {  
5  
6     public void actionPerformed(ActionEvent e) {  
7         String valor = campoValor.getText();  
8         resultado.setText(valor);  
9     }  
10 }  
11 }
```

Ejemplo: contador de pulsaciones

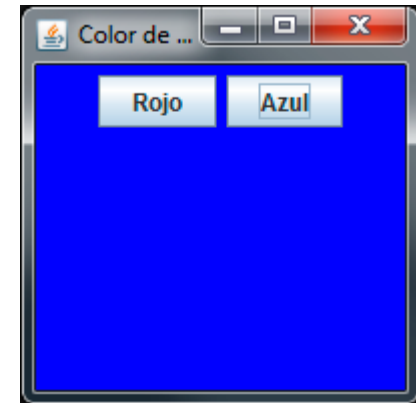
```
public class Gu11 extends JFrame {
    private final JButton boton1;
    public final JLabel label1;

    public Gu11() {
        boton1 = new JButton("PULSA");
        label1 = new JLabel("Pulsaciones: 0");
        add(boton1);
        add(label1);
        setLayout(new FlowLayout());
        boton1.addActionListener(new OyenteBotonPulsaciones());
        .....
    }
    public static void main(String args[]) {
        Gu11 ventana = new Gu11();
    }
    class OyenteBotonPulsaciones implements ActionListener {
        private int contador;
        public void actionPerformed(ActionEvent e) {
            contador++;
            label1.setText("Pulsaciones: " + contador);
        }
    }
}
```



Ejemplo: cambio de color

```
public class Gui12 extends JFrame {  
  
    JButton rojo = new JButton("Rojo");  
    JButton azul = new JButton("Azul");  
    Container p;  
  
    public Gui12() {  
        super("Color de fondo");  
        p = this.getContentPane();  
        setLayout(new FlowLayout());  
        add(rojo);  
        add(azul);  
        rojo.addActionListener(new OyenteRojo());  
        azul.addActionListener(new OyenteAzul());  
        setSize(200, 200);  
        setVisible(true);  
        setDefaultCloseOperation(EXIT_ON_CLOSE);  
    }  
}
```



Ejemplo: cambio de color

```
public static void main(String args[]) {  
    Gui12 ventana = new Gui12();  
}  
  
class OyenteRojo implements ActionListener {  
    public void actionPerformed(ActionEvent evento) {  
        p.setBackground(Color.red);  
    }  
}  
  
class OyenteAzul implements ActionListener {  
    public void actionPerformed(ActionEvent evento) {  
        p.setBackground(Color.blue);  
    }  
}  
}
```

ActionEvent

El objeto ActionEvent ofrece un método `getActionCommand()` que devuelve un objeto con la información sobre el origen del evento (el botón en nuestro caso).

Con un botón, devuelve la etiqueta del botón.

Para identificar botones individuales:

```
public void actionPerformed(ActionEvent e) {  
    String s = (String)e.getActionCommand();  
    if(s.equals("Aceptar")) {  
        // Tratamiento del botón Aceptar  
    }  
    ...  
}
```

ActionEvent

El método getSource() indica el objeto en el que se ha originado el evento:

```
public void actionPerformed(ActionEvent e) {  
    if(e.getSource() == lista) {  
        campo1.setText("En la lista.");  
    }  
    else if(e.getSource() == texto) {  
        campo2.setText("En el campo de texto.");  
    }  
    else if(e.getSource() == boton) {  
        campo3.setText("En el botón.");  
    }  
}
```

Un conversor Euros-Pesetas

```
public class Gui13 extends JFrame {  
  
    private JTextField cantidad;  
    private JButton boton1 , boton2;  
  
    public Gui13() {  
        super("Conversor Euros-Pesetas");  
        boton1 = new JButton("A euros");  
        boton2 = new JButton("A pesetas");  
        cantidad = new JTextField(10);  
        JLabel eti2 = new JLabel(new ImageIcon("C:\\logs\\indiprologo.png"));  
        add(eti2);  
        add(cantidad);  
        add(boton1);  
        add(boton2);  
        setLayout(new FlowLayout());  
        boton1.addActionListener(new OyenteBoton());  
        boton2.addActionListener(new OyenteBoton());  
        setSize(300, 250);  
        setVisible(true);  
        setDefaultCloseOperation(EXIT_ON_CLOSE);  
    }  
}
```



Un conversor Euros-Pesetas

```
public static void main(String args[]) {  
    Gui13 ventana = new Gui13();  
}  
  
class OyenteBoton implements ActionListener {  
  
    public void actionPerformed(ActionEvent ae) {  
        Float f = new Float(cantidad.getText());  
        float valor = f.floatValue();  
        String s = (String) ae.getActionCommand();  
        if (s.equals("A euros")) {  
            valor = (float) (valor / 166.321);  
        } else if (s.equals("A pesetas")) {  
            valor = (float) (valor * 166.321);  
        }  
        cantidad.setText(Float.toString(valor));  
    }  
}  
}
```

La clase OyenteBoton es interna y tiene acceso a los elementos de la clase Contenedora

Interfaces para procesar eventos I

Para facilitar la tarea del programador se han creado una serie de interfaces que deben implementarse cuando se quieren procesar algunos de estos eventos. Algunas interfaces y sus métodos son:

ActionListener Escucha eventos de tipo ActionEvent	actionPerformed (ActionEvent)
KeyListener Escucha eventos de teclado	keyPressed (KeyEvent) keyReleased (KeyEvent) keyTyped (KeyEvent)
MouseListener Escucha eventos de acción del ratón (botones)	mouseClicked (MouseEvent) mouseEntered (MouseEvent) mouseExited (MouseEvent) mousePressed (MouseEvent) mouseReleased (MouseEvent)
MouseMotionListener Escucha eventos de movimiento del ratón	mouseDragged (MouseEvent) mouseMoved (MouseEvent)

Interfaces para procesar eventos II

- ▶ **JButton**: Botón aislado. Puede pulsarse, pero su estado no cambia
- ▶ **JToggleButton** : Botón seleccionable. Cuando se pulsa el botón, su estado pasa a seleccionado, hasta que se pulsa de nuevo (entonces se deselecciona)
 - ▶ `isSelected()` permite chequear su estado
- ▶ **JCheckBox** : Especialización de JToggleButton que implementa una casilla de verificación. Botón con estado interno, que cambia de apariencia de forma adecuada según si está o no está seleccionado
- ▶ **JRadioButton**: Especialización de JToggleButton que tiene sentido dentro de un mismo grupo de botones (ButtonGroup) que controla que sóloamente uno de ellos está seleccionado

Componentes : JButton

► Constructores:

```
JButton(String text)
```

```
JButton(String text, Icon icon)
```

```
JButton(Icon icon)
```

► Respuesta a botones:

- Implementar la interfaz ActionListener
- Implementar el método actionPerformed(ActionEvent e)

```
public void actionPerformed(ActionEvent e) {  
    // Obtenemos la etiqueta  
    String actionCommand = e.getActionCommand();  
    if(e.getSource() instanceof JButton)  
        if("Aceptar".equals(actionCommand))  
            System.out.println("Pulsó Aceptar");  
}
```

Ejemplos

```
boton1 = new JButton("A Euros  ");  
boton1.setIcon(new ImageIcon("flag1.gif"));
```



```
boton2 = new JButton(new ImageIcon("flag2.gif"));
```



```
boton3 = new JButton("Botón",new ImageIcon("flag8.gif"));
```



Componentes : JLabel

Para texto, una imagen o ambos:

```
JLabel(String text,  
        int horizontalAlignment)  
JLabel(String text)  
JLabel(Icon icon)  
JLabel(Icon icon,  
        int horizontalAlignment)
```

```
eti1 = new JLabel("Etiqueta de texto...");  
eti2 = new JLabel(new ImageIcon("flag8.gif"));
```

Componentes : JTextField

Campos de texto para introducir caracteres:

```
JTextField(int columns)
```

```
JTextField(String text)
```

```
JTextField(String text, int columns)
```

```
JTextField text1 = new JTextField("hola", 10);
```

Poner texto: `text1.setText("Adios");`

Obtener texto: `String str = text1.getText();`

Agregar al Panel: `p1.add(text1);`

Componentes : JComboBox

Listas de elementos para seleccionar un solo valor:

Creación: **JComboBox ch1 = new JComboBox();**

Agregar opciones: **ch1.addItem(Object elemento);**

Registrar Evento: **ch1.addItemListener(objeto);**

Obtener selección: **val = ch1.getSelectedIndex();**
ch1.getItem()

Implementar la interfaz **ItemListener**

Implementar el método **itemStateChanged(ItemEvent e)**

```
ch1.addItemListener(new OyenteItem());  
  
...  
class OyenteItem implements ItemListener {  
    public void itemStateChanged(ItemEvent e) {  
        ...  
    }  
}
```


Componentes : JList

Listas de elementos para seleccionar uno o varios valores:

```
JList l1 = new JList();
```

```
JList l2 = new JList(Object[] elements);
```

```
String[] cosas = {"Opción 1", "Opción 2", "Opción 3"};
```

```
Jlist l2 = new Jlist(cosas);
```

Registrar evento: `l2.addListSelectionListener(oyente);`

Obtener selección:

```
int[] indices = l2.getSelectedIndices();
```

Componentes : JList

Implementar la interfaz **ListSelectionListener**

Implementar el método **valueChanged(ListSelectionEvent e)**

```
l.addListSelectionListener(new OyenteLista());  
  
...  
class OyenteLista implements ListSelectionListener {  
    public void valueChanged(ListSelectionEvent e) {  
        int[] indices = l.getSelectedIndices();  
        int i;  
        for(i = 0; i < indices.length; i++) {  
            ...  
        }  
    }  
}
```

Componentes : JScrollbar

Creación:

```
bar1 = new Scrollbar(Scrollbar.HORIZONTAL,0,0,0,100);
```

Registrar evento:

```
bar1.addAdjustmentListener(oyente);
```

Implementar la interfaz **AdjustmentListener**

Implementar el método

```
adjustmentValueChanged(AdjustmentEvent e)
```

```
bar1.addAdjustmentListener(new OyenteBarra());
```

...

```
class OyenteBarra implements AdjustmentListener {
```

```
    public void adjustmentValueChanged(AdjustmentEvent e) {
```

```
        ... }
```

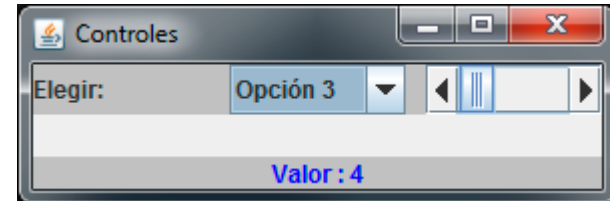
```
}
```

Obtener valor:

```
int val = bar1.getValue(); // val entre 0 y 100
```

Componentes: JScrollBar y JComboBox

```
public class Gui15 extends JFrame {  
  
    Container panel;  
    JPanel p1, p2;  
    JLabel l1, msg;  
    JComboBox ch1;  
    String[] lista = {"Opción 1", "Opción 2", "Opción 3"};  
    JScrollBar bar1;  
  
    public Gui15() {  
        super("Controles");  
        setLayout(new BorderLayout());  
        p1 = new JPanel(new GridLayout(1, 3, 10, 10));  
        p1.setBackground(Color.lightGray);  
        l1 = new JLabel("Elegir:", Label.RIGHT);  
        l1.setBackground(Color.yellow);  
        p1.add(l1);  
        ch1 = new JComboBox();  
        for (int i = 0; i < lista.length; i++) {  
            ch1.addItem(lista[i]);  
        }  
        ch1.addItemListener(new OyenteCombo());  
    }  
}
```



Componentes: JScrollBar y JComboBox

```
bar1 = new JScrollBar(Scrollbar.HORIZONTAL, 0, 0, 0, 100);
/* scroll de 0 a 100*/
bar1.addAdjustmentListener(new OyenteBarra());
p1.add(bar1);
p2 = new JPanel(new BorderLayout());
p2.setBackground(Color.lightGray);
msg = new JLabel("Msg:", Label.LEFT);
msg.setForeground(Color.blue);
p2.add("North", msg);
add(p1, "North");
add(p2, "South");
setSize(300, 100);
setVisible(true);
setDefaultCloseOperation(EXIT_ON_CLOSE);
}

public static void main(String args[]) {
    Gui15 ventana = new Gui15();
}
```

Componentes: JScrollBar y JComboBox

```
class OyenteCombo implements ItemListener {

    public void itemStateChanged(ItemEvent e) {
        int ind = chl.getSelectedIndex();
        msg.setText((String) chl.getSelectedItem());
    }
}

class OyenteBarra implements AdjustmentListener {

    public void adjustmentValueChanged(AdjustmentEvent e) {
        int valor = bar1.getValue();
        String cad = "Valor : " + valor;
        msg.setText(cad);
    }
}
}
```

Movimientos de ratón

```
public class Gui17 extends JFrame {
    JButton boton;
    List lista;
    Container panel;
    public Gui17() {
        ...
        this.addMouseMotionListener(new OyenteMover());
        ...
    }
    ...
}

class OyenteMover implements MouseMotionListener {
    public void mouseDragged(MouseEvent e) {
        lista.add("arrastrando..");
    }
    public void mouseMoved(MouseEvent e) {
        lista.add("moviendo..");
    }
}
```

Pulsaciones de ratón

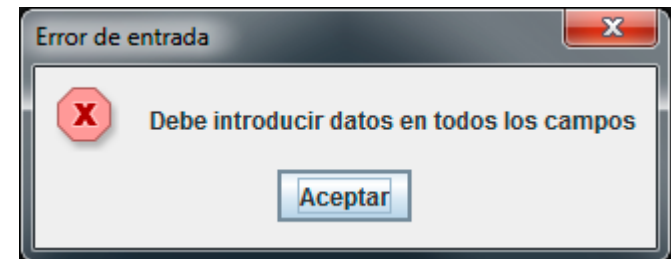
```
class OyenteRaton implements MouseListener {
    public void mouseClicked(MouseEvent e) {
        lista.add("click..");
    }
    public void mouseEntered(MouseEvent e) {
        lista.add("enter.."); }
    public void mouseExited(MouseEvent e) {
        lista.add("exit.."); }
    public void mousePressed(MouseEvent e) {
        lista.add("pulsar.."); }
    public void mouseReleased(MouseEvent e) {
        lista.add("soltar..");
    }
}
```


Ejemplo de cuadro de mensaje

```
public class Gui18 extends JFrame {

    public Gui18() {
        super("Título de la ventana");
        setLayout(new FlowLayout());
        setSize(200, 100); // pack();
        setVisible(true); // show();
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        // if ocurre algo
        JOptionPane.showMessageDialog(null,
            "Debe introducir datos en todos los campos",
            "Error de entrada ",
            JOptionPane.ERROR_MESSAGE);
    }

    public static void main(String[] args) {
        Gui18 f = new Gui18();
    }
}
```



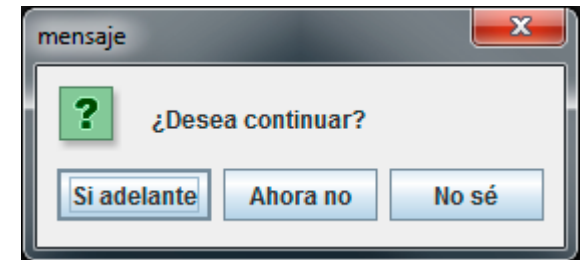
Ejemplo de cuadro de opciones

```
public class Gui19 extends JFrame {

    private final Container p;

    public Gui19() {
        super("Título de la ventana");
        p = getContentPane();
        setLayout(new FlowLayout());
        setSize(200, 100);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Object[] textoOpciones = {"Si adelante", "Ahora no", "No sé"};
        int opcion = JOptionPane.showOptionDialog(null,
            "¿Desea continuar?", "mensaje",
            JOptionPane.YES_NO_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE, null, textoOpciones,
            textoOpciones[0]);
    }

    public static void main(String[] args) {
        Gui19 f = new Gui19();
    }
}
```



Ejemplo de cuadro de entrada de datos

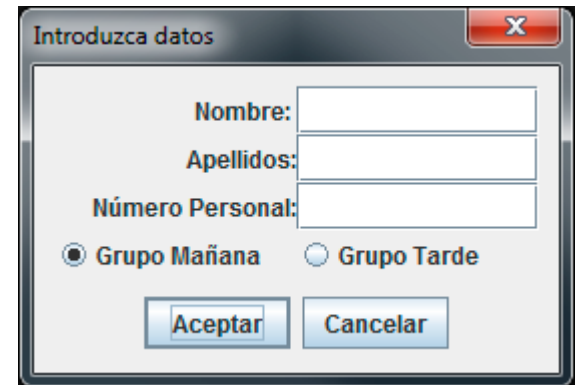
```
class PanelDatos extends JPanel {  
  
    public PanelDatos() {  
        setLayout(new GridLayout(4, 2));  
        JLabel etiquetaNombre = new JLabel("Nombre: ", JLabel.RIGHT);  
        JTextField campoNombre = new JTextField();  
        add(etiquetaNombre);  
        add(campoNombre);  
        JLabel etiquetaApellidos = new JLabel("Apellidos:", JLabel.RIGHT);  
        JTextField campoApellidos = new JTextField();  
        add(etiquetaApellidos);  
        add(campoApellidos);  
        JLabel etiquetaNP = new JLabel("Número Personal:", JLabel.RIGHT);  
        JTextField campoNP = new JTextField();  
        add(etiquetaNP);  
        add(campoNP);  
        ButtonGroup grupoBotones = new ButtonGroup();  
        JRadioButton mañana = new JRadioButton("Grupo Mañana", true);  
        JRadioButton tarde = new JRadioButton("Grupo Tarde");  
        grupoBotones.add(mañana);  
        grupoBotones.add(tarde);  
        add(mañana);  
        add(tarde);  
    }  
}
```

Ejemplo de cuadro de entrada de datos

```
public class Gui20 extends JFrame {

    public Gui20() {
        super("Título de la ventana");
        setLayout(new FlowLayout());
        // Cuando necesitamos el cuadro de diálogo...
        PanelDatos pd = new PanelDatos();
        if (JOptionPane.showConfirmDialog(this, pd,
            "Introduzca datos",
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.PLAIN_MESSAGE)
            == JOptionPane.OK_OPTION) {
            // ... tratamiento
        }
    }

    public static void main(String[] args) {
        Gui20 f = new Gui20();
    }
}
```



Menús

Java ofrece varias clases para poner menús en una ventana:

JMenuBar

JMenu

JMenuItem

JCheckBoxMenuItem

JRadioButtonMenuItem

Un **JFrame** o **JApplet** puede guardar un barra de menú donde se cuelgan menús desplegables.

Los menús tienen elementos de menú que puede seleccionar el usuario.

Las barras de menús se pueden contemplar como una estructura que soporta menús.

Menús

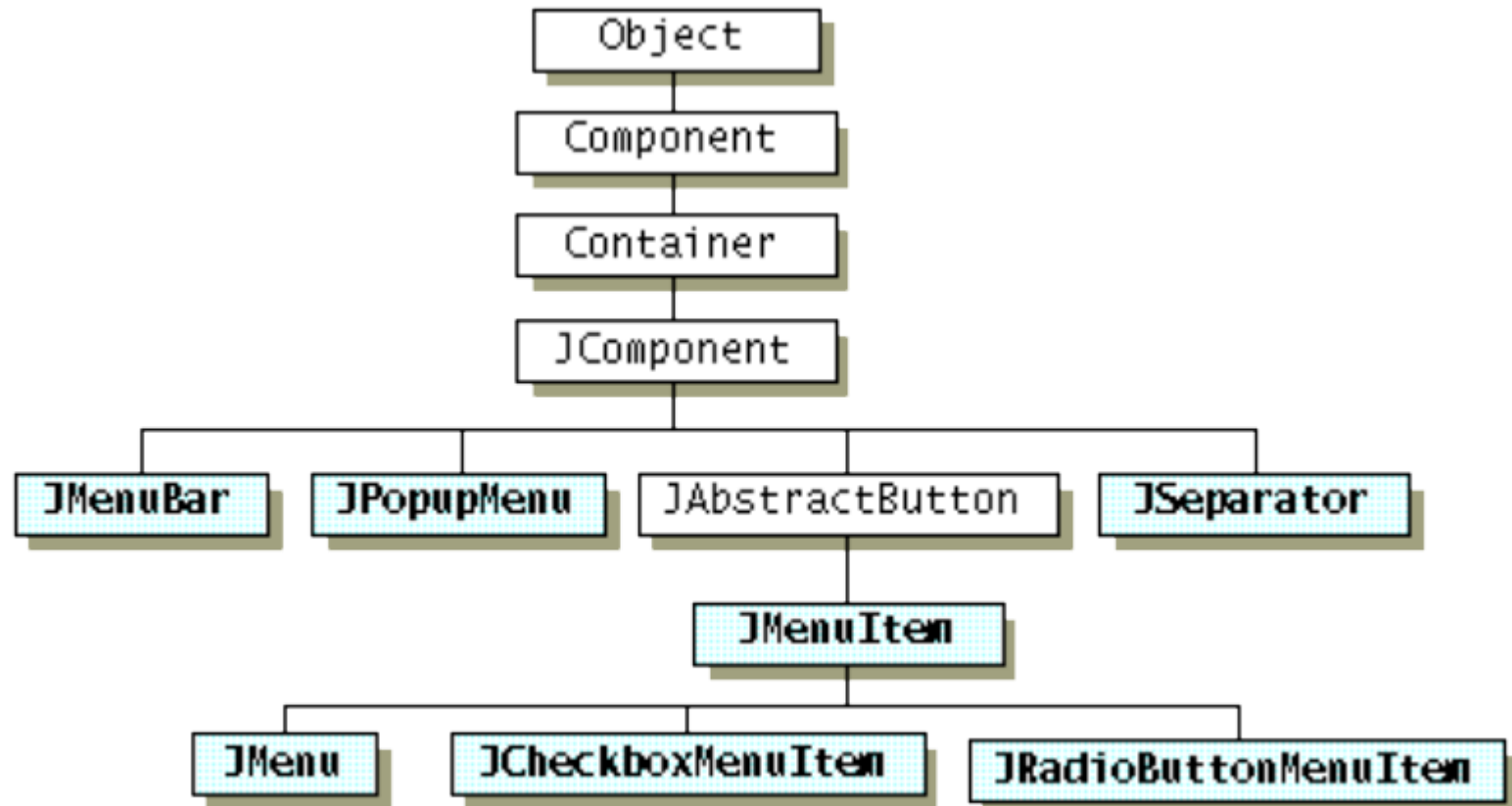
Una ventana (frame) sólo puede tener una barra de menús (objeto MenuBar), que se sitúa en la parte de arriba del mismo.

Los submenús son botones JMenu.

Los elementos de los menús son botones JMenuItem.

Cuando se activa un menú se despliegan automáticamente las opciones del menú.

Menús



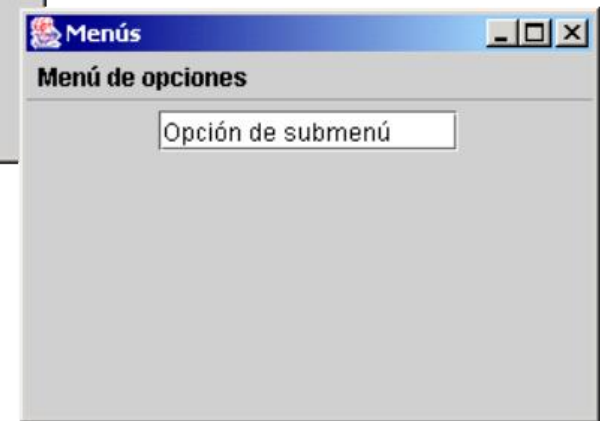
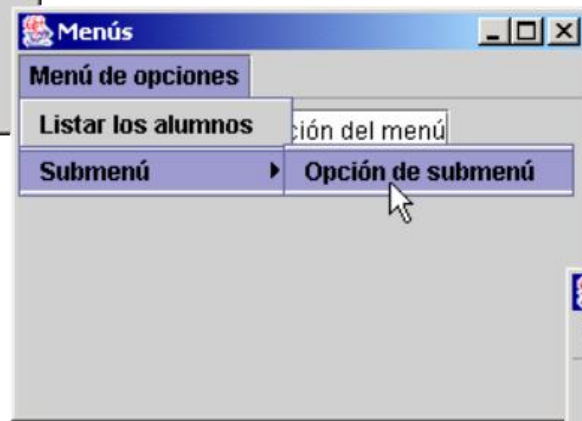
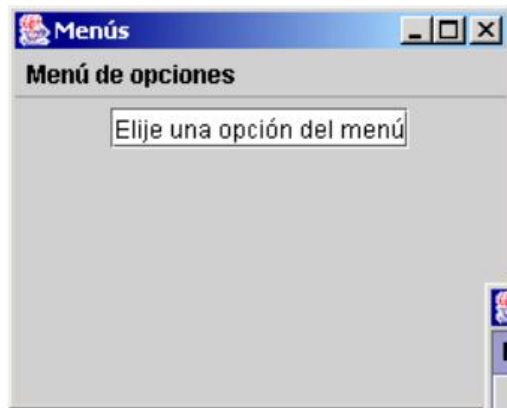
Ejemplo de menú

```
JMenuBar barraMenu = new JMenuBar();
setJMenuBar(barraMenu); // La barra de menús de este frame
// El menú:
JMenu menuOpciones = new JMenu("Menú de opciones");
barraMenu.add(menuOpciones); // Añadimos el menú a la barra
// Un elemento de menú:
JMenuItem listar = new JMenuItem("Listar los alumnos");
menuOpciones.add(listar); // Añadimos el elemento al menú
// Inserción de una línea separadora en el menú:
menuOpciones.add(new JSeparator());
// Un menú que será un submenú del anterior:
JMenu subMenu = new JMenu("Submenú");
// Un elemento de menú para el submenú:
JMenuItem opcionSubmenu = new JMenuItem("Opción de submenú");
subMenu.add(opcionSubmenu); // La añadimos al submenú
// Añadimos el submenú como elemento del menú:
menuOpciones.add(subMenu);
```


Ejemplo de menú

```
// Establecemos oyentes para las opciones elegibles:
listar.addActionListener(new OyenteMenu());
opcionSubmenu.addActionListener(new OyenteMenu());
...
class OyenteMenu implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String actionCommand = e.getActionCommand();
        if(e.getSource() instanceof JMenuItem) {
            if("Listar los alumnos".equals(actionCommand)) {
                texto.setText("Listar los alumnos");
            }
            if("Opción de submenú".equals(actionCommand)) {
                texto.setText("Opción de submenú");
            }
        }
    }
}
```

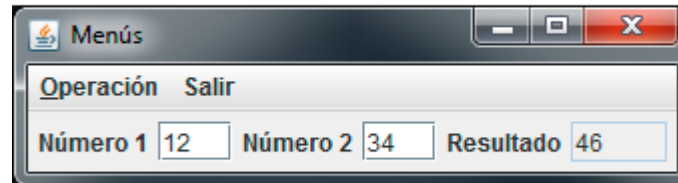
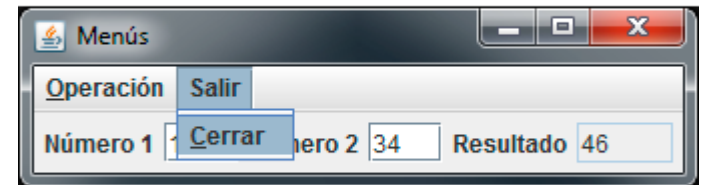
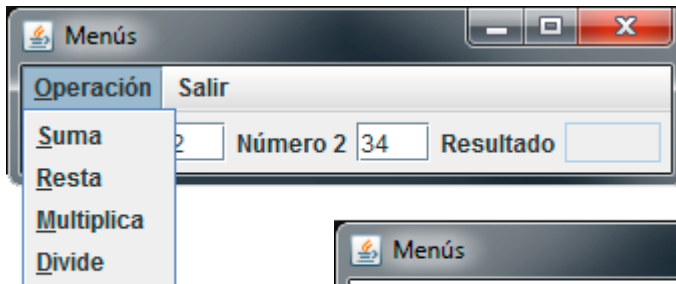
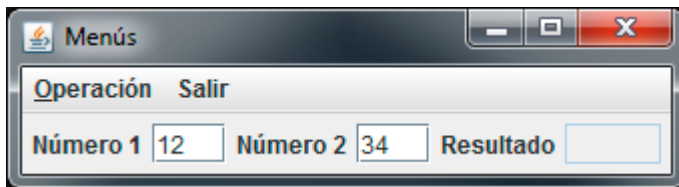
Ejemplo de menú



Otro ejemplo

Aplicación que permite realizar operaciones aritméticas.

La interfaz contiene etiquetas y campos de texto para los operandos y el resultado. La operación se selecciona en el menú:



Otro ejemplo

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

public class Gui22 extends JFrame {

    Container panel;
    JTextField jtfNum1, jtfNum2, jtfResult;
    JMenuItem jmiSuma, jmiResta, jmiMul, jmiDiv, jmiCerrar;

    public Gui22() {
        super("Menús");
        JMenuBar jmb = new JMenuBar();
        setJMenuBar(jmb);
        JMenu operationMenu = new JMenu("Operación");
        operationMenu.setMnemonic('O'); // Letra distinguida
        jmb.add(operationMenu);
        operationMenu.add(jmiSuma = new JMenuItem("Suma", 'S'));
        operationMenu.add(jmiResta = new JMenuItem("Resta", 'R'));
        operationMenu.add(jmiMul = new JMenuItem("Multiplica", 'M'));
        operationMenu.add(jmiDiv = new JMenuItem("Divide", 'D'));
        JMenu exitMenu = new JMenu("Salir");
        jmb.add(exitMenu);
    }
}
```

Otro ejemplo

```
exitMenu.add(jmiCerrar = new JMenuItem("Cerrar", 'C'));
JPanel p1 = new JPanel();
p1.setLayout(new FlowLayout());
p1.add(new JLabel("Número 1"));
p1.add(jtfNum1 = new JTextField(3));
p1.add(new JLabel("Número 2"));
p1.add(jtfNum2 = new JTextField(3));
p1.add(new JLabel("Resultado"));
p1.add(jtfResult = new JTextField(4));
jtfResult.setEditable(false);
getContentPane().setLayout(new BorderLayout());
getContentPane().add(p1, BorderLayout.CENTER);
// Registramos oyentes
jmiSuma.addActionListener(new OyenteMenu());
jmiResta.addActionListener(new OyenteMenu());
jmiMul.addActionListener(new OyenteMenu());
jmiDiv.addActionListener(new OyenteMenu());
jmiCerrar.addActionListener(new OyenteMenu());
}
```

Otro ejemplo

```
public static void main(String args[]) {
    Gui22 ventana = new Gui22();
    ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    ventana.pack();
    ventana.setVisible(true);
}

private void calculate(char operator) {
    int num1 = (Integer.parseInt(jtfNum1.getText().trim()));
    int num2 = (Integer.parseInt(jtfNum2.getText().trim()));
    int result = 0;
    switch (operator) {
        case '+':
            result = num1 + num2;
            break;
        case '-':
            result = num1 - num2;
            break;
        case '*':
            result = num1 * num2;
            break;
        case '/':
            result = num1 / num2;
    }
}
```

Otro ejemplo

```
        jtfResult.setText(String.valueOf(result));
    }

    class OyenteMenu implements ActionListener {

        public void actionPerformed(ActionEvent e) {
            String actionCommand = e.getActionCommand();
            if (e.getSource() instanceof JMenuItem) {
                if ("Suma".equals(actionCommand)) {
                    calculate('+');
                } else if ("Resta".equals(actionCommand)) {
                    calculate('-');
                } else if ("Multiplica".equals(actionCommand)) {
                    calculate('*');
                } else if ("Divide".equals(actionCommand)) {
                    calculate('/');
                } else if ("Cerrar".equals(actionCommand)) {
                    System.exit(0);
                }
            }
        }
    }
}
```

Clases adaptadoras

```
class OyenteRaton implements MouseListener {  
    public void mouseClicked(MouseEvent e) {  
        lista.add("click..");  
    }  
    public void mouseEntered(MouseEvent e) {}  
    public void mouseExited(MouseEvent e) {}  
    public void mousePressed(MouseEvent e) {}  
    public void mouseReleased(MouseEvent e) {}  
}
```

¡ Poco eficiente, sólo implementamos un método !

Clases adaptadoras

```
class OyenteRatonconAdap extends MouseAdapter {  
    // La clase adaptadora MouseAdapter  
    // se encargan de implementar todos los métodos  
    // de la clase de escucha. Sólo necesitaremos  
    // redefinir aquellos métodos que nos van a ser útiles  
    // para gestionar eventos , sin preocuparnos del resto  
  
    public void mouseClicked(MouseEvent e) {  
        //redefinido  
        lista.add("click..");  
    }  
}
```

Clases adaptadoras

- ▶ Si por ejemplo una de nuestras clases implementa la interfaz `WindowListener`, deberá implementar todos los métodos asociados, aún cuando sólo utilicemos uno de ellos.
- ▶ Las clases adaptadoras se encargan de implementar todos los métodos de la clase de escucha. Así sólo necesitaremos redefinir aquellos métodos que nos van a ser útiles para gestionar eventos , sin preocuparnos del resto.
- ▶ Para ello dedemos indicar que nuestra clase es una subclase del adaptador:

```
class UnaClase extends Adaptador{ ... }
```

Clases adaptadoras

```
class MiAdaptador extends WindowAdapter {  
    public void windowClosing(WindowEvent e) {  
        System.exit(0);  
    }  
}
```

```
. . .  
this.addWindowListener(new MiAdaptador());  
. . .
```

Clases adaptadoras

Sólo las clases que poseen más de un método tienen adaptador , y son las siguientes:

ComponentListener posee ComponentAdapter

ContainerListener posee ContainerAdapter

FocusListener posee FocusAdapter

KeyListener posee KeyAdapter

MouseListener posee MouseAdapter

MouseMotionListener posee MouseMotionAdapter

WindowListener posee WindowAdapter

Ejemplo

```
public class Gui17b extends JFrame {

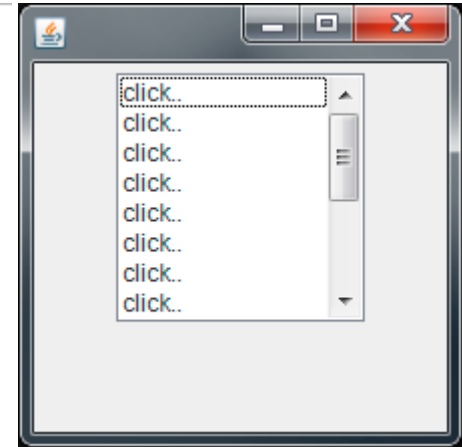
    JButton boton;
    List lista;
    Container panel;

    public Gui17b() {
        panel = this.getContentPane();
        panel = getContentPane();
        panel.setLayout(new FlowLayout());
        lista = new List(8, false);
        lista.setBounds(100, 50, 150, 150);
        add(lista);
        this.addMouseListener(new OyenteAdaptador());
        setSize(100, 100);
        setVisible(true);
    }

    class OyenteAdaptador extends MouseAdapter {

        public void mouseClicked(MouseEvent e) {
            lista.add("click..");
        }
    }
}
```

85.....



Arquitectura MVC (Modelo-Vista-Controlador)

La arquitectura MVC es una arquitectura típica de diseño de GUI.

- ✓ El modelo representa la estructura lógica de la GUI, independientemente de su representación visual.
- ✓ La vista constituye la representación visual de la GUI.
- ✓ El controlador constituye la lógica de interacción con el usuario.

El mejor modelo separa en las tres partes:

```
public static void main(String args[]) {  
    Modelo modelo = new Modelo();  
    Controlador controlador = new Controlador(modelo);  
    GUI gui = new GUI(controlador); }
```

Applet mínimo con Swing (JApplet)

```
//MiApplet.java
import java.swing.*;
class MiApplet extends JApplet { }
```

```
//MiApplet.html
<HTML>
<HEAD><TITLE>Applet mínima</TITLE> </HEAD>
  <BODY>
    <APPLET CODE="MiApplet.class" WIDTH=100 HEIGHT=50>
    </APPLET>
  </BODY>
</HTML>
```

Esquema de applet con swing

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

public class Applet0 extends JApplet {
    // constantes y componentes (atributos)
    public void init() {
        // configurar componentes
        // configurar layout;
        // configurar Manejadores Eventos;
    }
}
```


Ejemplo

```
import javax.swing.*;
import java.awt.*;
public class Applet01 extends JApplet
{
    JLabel etiqueta;
    public void init(){
        // configurar componentes;
        etiqueta = new JLabel("Mi primer Applet");
        add(etiqueta);
        // configurar layout;
        FlowLayout milayout = new FlowLayout();
        setLayout(milayout);
        ...
    }
}
```

