

# 4.1: Interfaces de usuario gráficas en Java: Swing/AWT



A. Goñi, J. Ibáñez, J. Iturrioz, J.A. Vadillo



OCW  
2013



# Índice

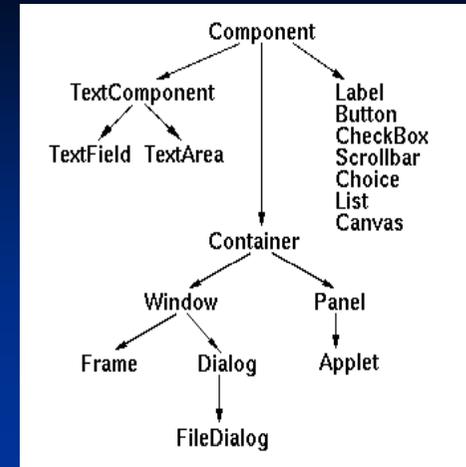
- Introducción
  - Objetivos
  - Jerarquía de clases: patrón de diseño COMPOSITE
- Componentes principales
  - Contenedores AWT/Swing
  - Componentes AWT/Swing
  - Gestores de diseño
- Gestión de eventos
  - Clases Listener
  - Clases Adapter
- Separación interfaz y lógica de negocio

# Introducción

- Cualquier lenguaje de programación moderno ofrece **herramientas** para la construcción de interfaces gráficas de usuario (GUI).
- Java permite al programador:
  - El diseño y programación de interfaces gráficas de usuario de forma rápida y sencilla.
  - El paquete de clases AWT (Abstract Window Toolkit)
  - El paquete de clases Swing
    - Swing es una evolución de AWT, ofreciendo más clases y una mayor flexibilidad.

# Objetivos

- Entender el diseño de la jerarquía de clases Java que se utilizan para la construcción de GUIs.

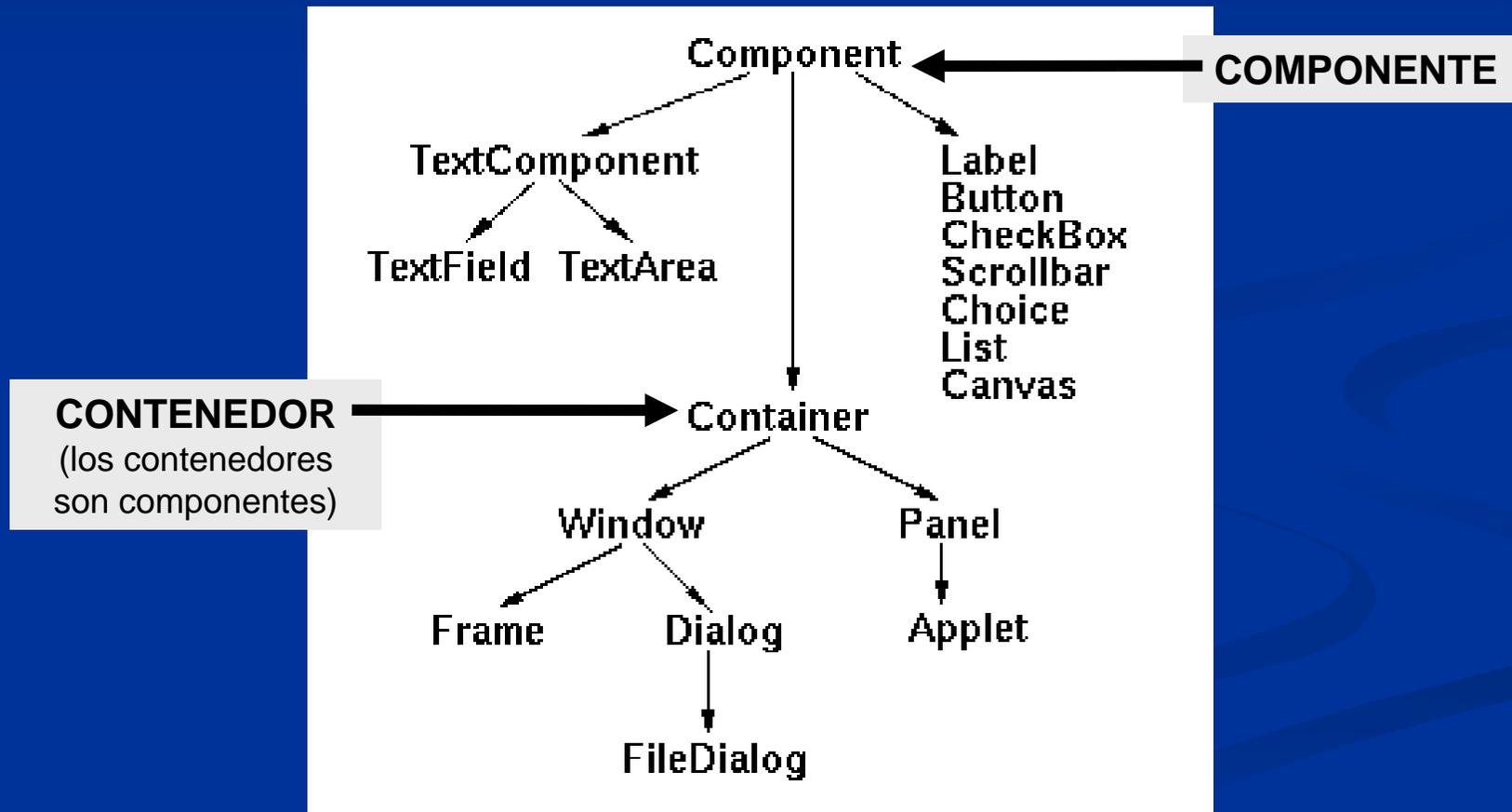


## Evento

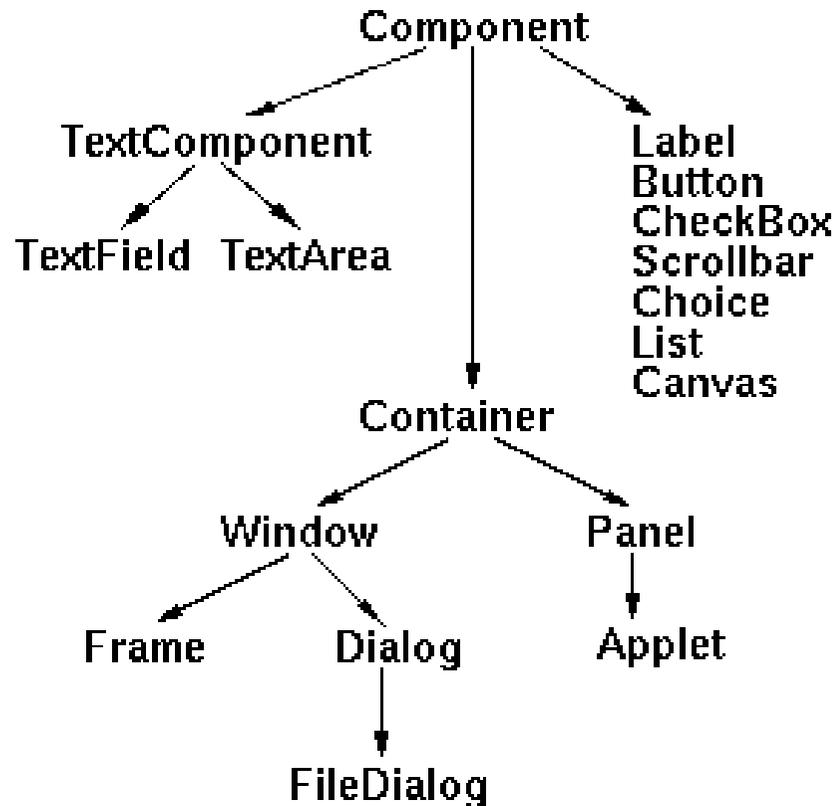
- Entender cómo se realiza la gestión de eventos.
- Aprender a usar un entorno de programación (Eclipse, JDeveloper...) para construir GUIs
- No es un objetivo:
  - conocer todos los nombres de clases, interfaces y demás componentes gráficos

# AWT: componentes principales

## Jerarquía de clases AWT



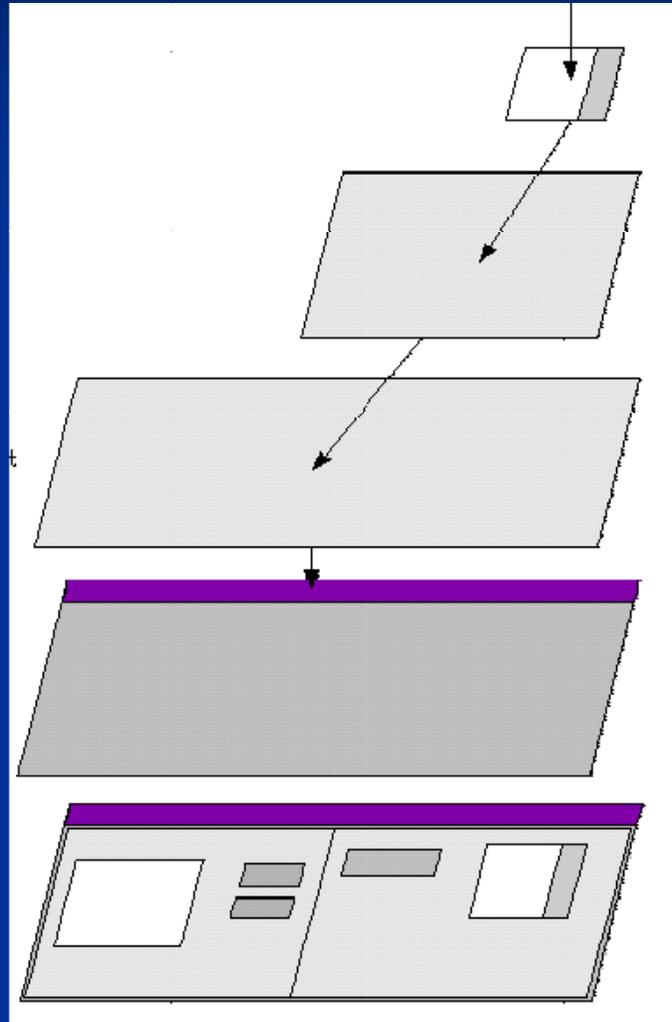
# Jerarquía de clases AWT : patrón de diseño COMPOSITE



Permiten definir infinitas ventanas distintas:

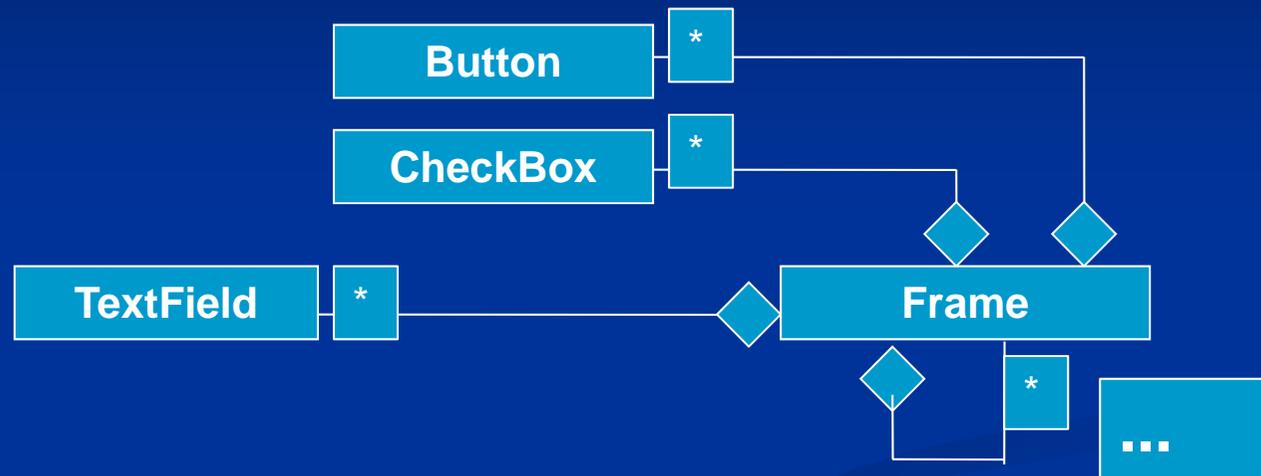
- Una ventana con 2 campos de texto, 3 botones y 2 áreas de texto, además de un panel que contiene 5 casillas de verificación y una lista desplegable.
- Una ventana con 2 etiquetas, 2 áreas de texto y un botón.
- ...

# Jerarquía de clases AWT : patrón de diseño COMPOSITE



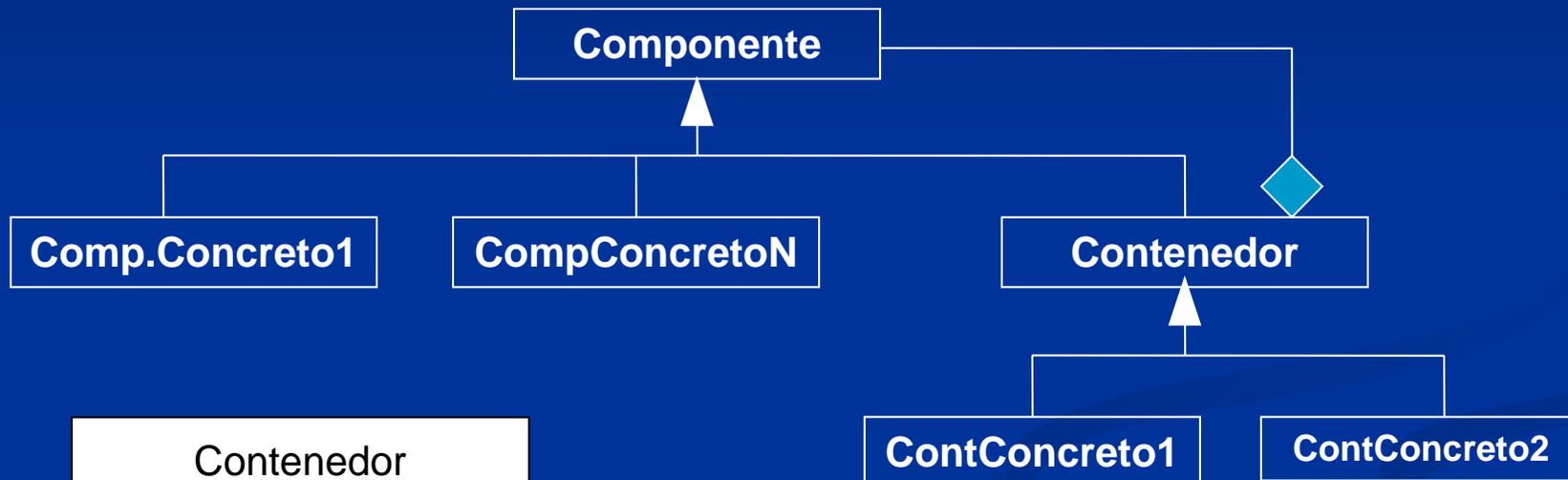
# Jerarquía de clases AWT :

un diseño como el siguiente sería incorrecto..



- En un `Frame`, necesitaríamos disponer de los métodos `addButton`, `addCheckBox`, `addTextField`, `addFrame`,... .
- Además, el diagrama no está en absoluto completo, dado que `Button` puede ser un componente de `Panel`, o de `Dialog`, ... y el diseño anterior no lo contempla.
- Si se quisiera añadir un nuevo componente `XXX`, deberíamos cambiar la clase `Frame` y añadir el método `addXXX` (esta solución no sería nada extensible)

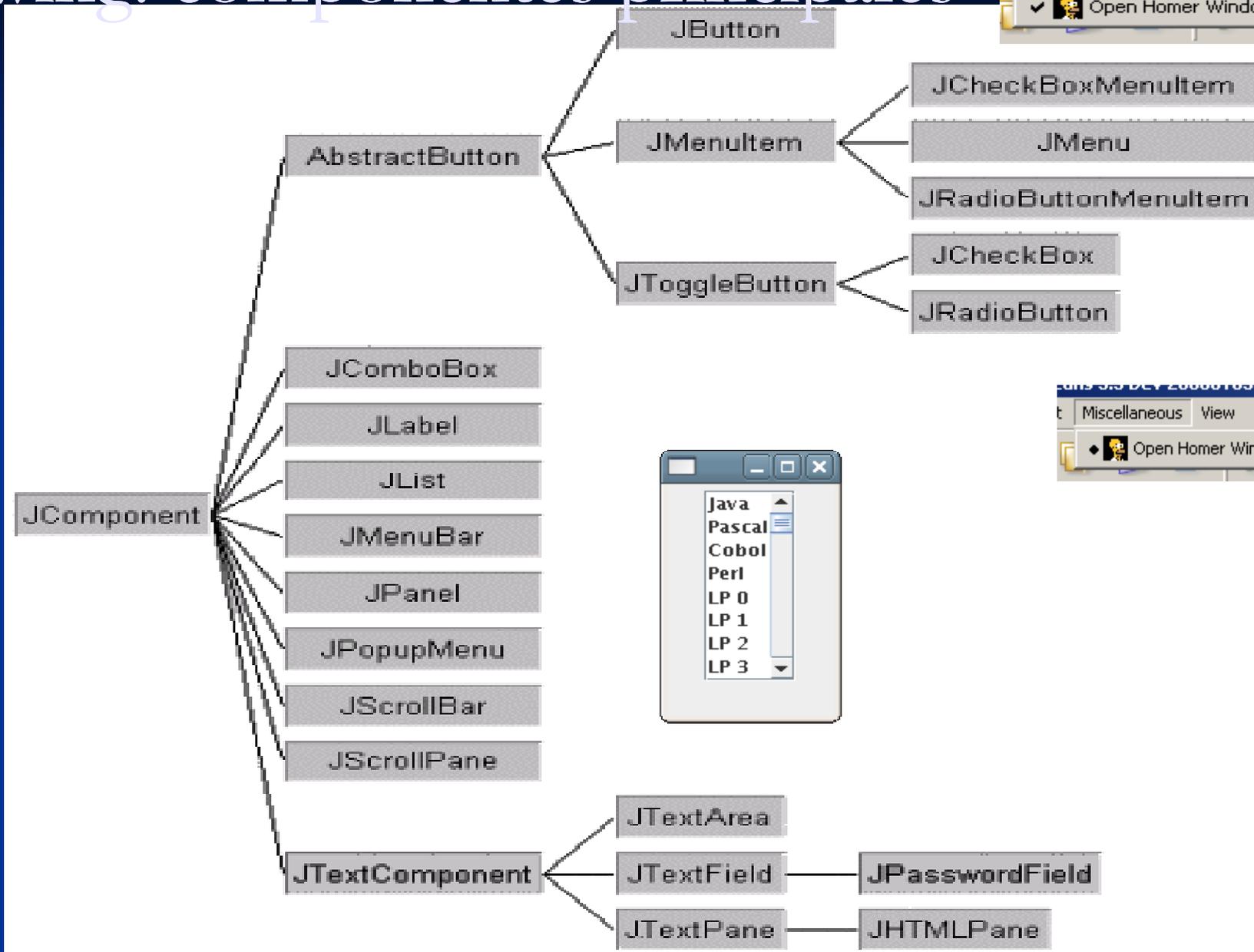
# Jerarquía de clases AWT: patrón de diseño COMPOSITE



Contenedor
add (Componente c)
--El método add
--añade un componente
--al contenedor

Un **contenedor** está formado por **componentes**, siendo éstos componentes concretos o contenedores. A su vez, estos contenedores podrían tener distintos componentes, etc.

# Swing: componentes principales



# ¿Dónde estudiar el API Java?

The screenshot shows a Mozilla Firefox browser window displaying the Java API documentation for the `JFrame` class. The browser's address bar shows the URL `http://java.sun.com/j2se/1.5.0/docs/api/`. The page title is "JFrame (Java 2 Platform SE 5.0) - Mozilla Firefox". The browser's menu bar includes "File", "Edit", "View", "Go", "Bookmarks", "ScrapBook", "Tools", "Diccionarios", and "Help". The browser's toolbar shows navigation buttons and a search bar. The browser's status bar at the bottom shows the URL `http://java.sun.com/j2se/1.5.0/docs/api/help-doc.html`.

The Java API documentation page for `JFrame` is displayed. The page title is "JFrame (Java 2 Platform SE 5.0) - Mozilla Firefox". The page content includes the following sections:

- Overview Package Class Use Tree Deprecated Index Help**
- PREV CLASS NEXT CLASS**
- FRAMES NO FRAMES**
- SUMMARY: NESTED | FIELD | CONSTR | METHOD**
- DETAIL: FIELD | CCNSTR | METHOD**
- Java™ 2 Platform Standard Ed. 5.0**
- javax.swing**
- Class JFrame**
- java.lang.Object**
  - java.awt.Component**
    - java.awt.Container**
      - java.awt.Window**
        - java.awt.Frame**
          - javax.swing.JFrame**

- All Implemented Interfaces:**
- ImageObserver, MenuContainer, Serializable, Accessible, RootPaneContainer, WindowConstants**
- public class JFrame**
- extends Frame**
- implements WindowConstants, Accessible, RootPaneContainer**
- An extended version of java.awt.Frame that adds support for the JFC/Swing component architecture. You can find task-oriented documentation about using JFrame in *The Java Tutorial*, in the section [How to Make Frames](#).**
- The JFrame class is slightly incompatible with Frame. Like all other JFC/Swing top-level containers, a JFrame contains a JRootPane as its only child. The **content pane** provided by the root pane should, as a rule, contain all the non-menu components displayed by the JFrame. This is different from the AWT Frame case. As a convenience add and its variants,**

# AWT/Swing: contenedores

Contenedores {  
• **Frame/JFrame** ← el principal  
• **Panel/JPanel**  
• **Dialog/JDialog**

## ■ Clases Frame/JFrame

- Una simple ventana que ofrece iconos para maximizar, minimizar y cerrarla. Se le puede añadir un título.
- Único contenedor al que se le pueden añadir menús.

## ■ Clases Panel/JPanel

- Contenedores genéricos para agrupar componentes.
- Clase utilizada para meter contenedores dentro de otros contenedores (dentro de un panel, subpaneles)
- En Swing, dentro de la clase JFrame se añade automáticamente un Panel. En AWT no.

# Contenedores AWT/Swing

```
import javax.swing.*;

public class Marco extends JFrame {
    private JPanel jPanel = ...

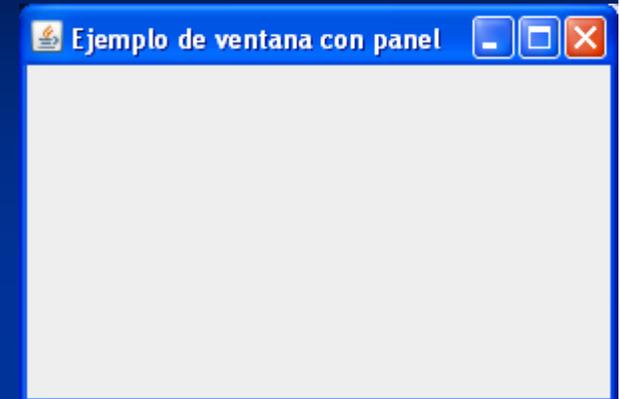
    public Marco() {
        super("Ejemplo de ventana con panel");
        initialize();
    }

    private void initialize() {
        this.setSize(300, 200);
        this.setContentPane(getContentPane());
    }

    public static void main(String[] args) {
        Marco thisClass = new Marco();

        ...

        thisClass.setVisible(true);
    }
}
```



Dentro de la clase JFrame se añade un Panel automáticamente

# Componentes principales AWT/Swing

## Clases TextField/JTextField

- Campo de entrada de texto de una sólo línea.

## Clases TextArea/JTextArea

- Permiten introducir líneas de texto múltiples
- Se puede usar también para mostrar resultados (texto)
- La clase JTextArea no implementa un scroll automático.  
Es necesario añadirlo dentro de un JScrollPane.  
TextArea, sin embargo, sí lo implementa.

```
F:\alfredo\EjsJava\fuentes\ejswing\CajasTexto.java
package ejswing;
import javax.swing.*;
import java.awt.*;
public class CajasTexto extends JFrame {
    JLabel jLabel1 = new JLabel();
    FlowLayout flowLayout1 = new FlowLayout();
    JTextField jTextField1 = new JTextField();
    JLabel jLabel2 = new JLabel();
    JTextArea jTextArea1 = new JTextArea();
    public CajasTexto() {
        super();
        try {
            jbInit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
    private void jbInit() throws Exception {
        jLabel1.setText("Escribe tu nombre:");
        jTextField1.setColumns(25);
        jLabel2.setText("Escribe tu reclamación (máximo 500 palabras):");
        jTextArea1.setColumns(50);
        jTextArea1.setRows(10);
        this.getContentPane().setLayout(flowLayout1);
        this.setSize(new Dimension(600, 275));
        this.setTitle("Ejemplo de uso de Cajas de Texto");
        this.getContentPane().add(jLabel1, null);
        this.getContentPane().add(jTextField1, null);
        this.getContentPane().add(jLabel2, null);
        this.getContentPane().add(jTextArea1, null);
    }
    public static void main (String []arg) {
        Frame b = new CajasTexto();
        b.setVisible(true);
    }
}

Ejemplo de uso de Cajas de Texto
Escribe tu nombre: 
Escribe tu reclamación (máximo 500 palabras):

```

**JLabel para mostrar información**

**JTextField y JTextArea: para entrada de texto**

Source Editor / Visual Designer / Class Designer | 14: 5 | Modified | Insert

# Componentes principales de AWT/Swing

## Clases Button/JButton

- Se usa para construir Botones.
- Al pulsar un botón se generará un evento, que habrá que tratar.

## Clases Label/JLabel

- Utilizado para mostrar información.
- Se usan junto a los cuadros de texto.

F:\alfredo\EjsJava\fuentes\ejsSwing\Botones.java

```
package ejsSwing;
import javax.swing.*.*;
import java.awt.*.*;
/**
 * Ejemplo de clase para mostrar la creación de botones.
 * @author Prof. ISO
 */
public class Botones extends JFrame {
    JPanel jPanel1 = new JPanel();
    JButton jButton1 = new JButton();
    JButton jButton2 = new JButton();
    JButton jButton3 = new JButton();
    /**
     * Método constructor.
     */
    public Botones() {
        this.setTitle("Ejemplo de uso de botones");
        jButton1.setText("Abrir");
        jButton2.setText("Salvar");
        jButton3.setText("Cancelar");
        this.getContentPane().add(jPanel1);
        jPanel1.add(jButton1);
        jPanel1.add(jButton2);
        jPanel1.add(jButton3);
        pack(); // Ajusta el tamaño del JFrame al mínimo necesario
    }
    /**
     * El método principal crea un Frame y lo muestra.
     */
    public static void main (String []arg) {
        Frame b = new Botones();
        b.setVisible(true);
    }
}
```

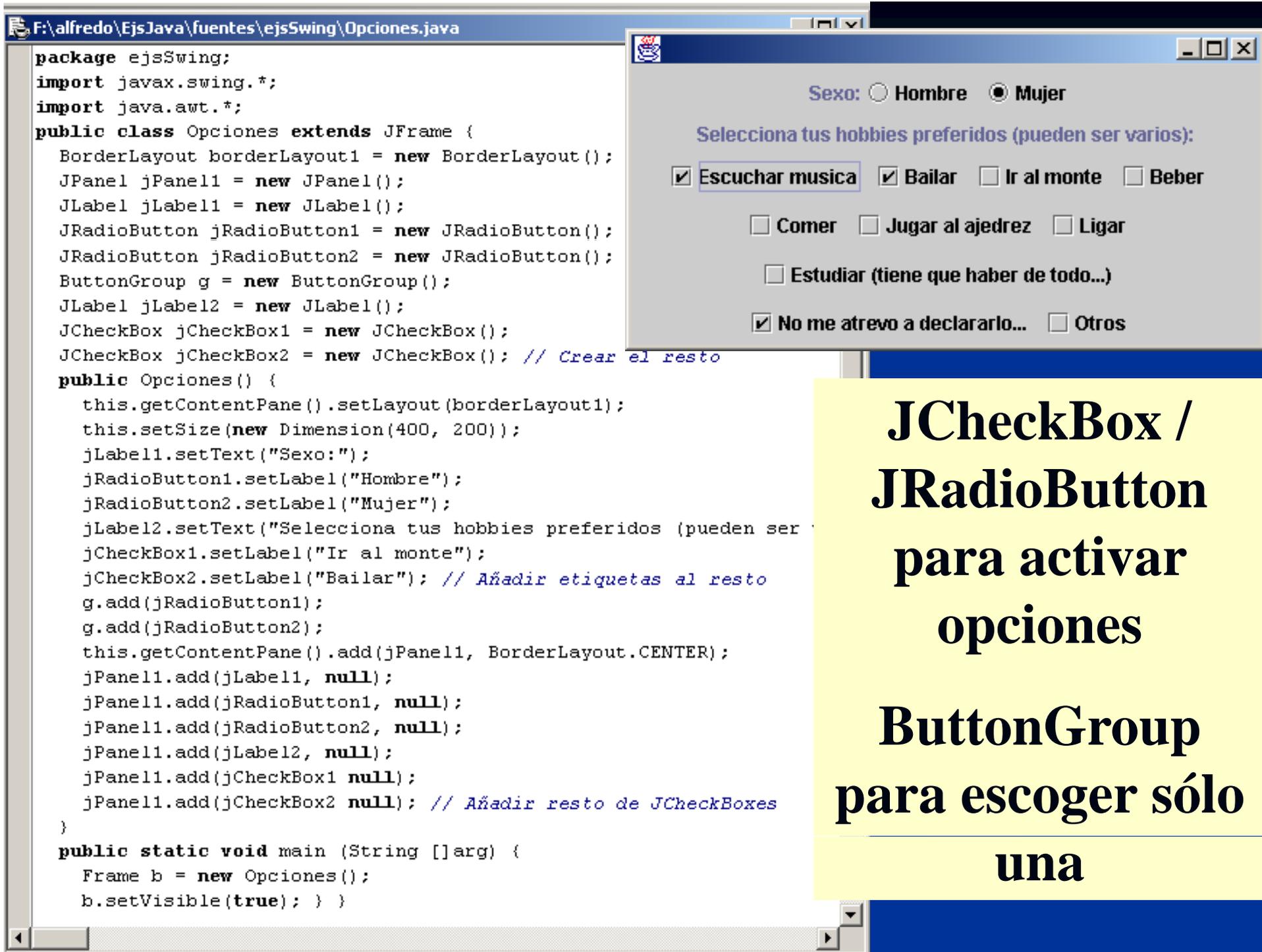


Se usan JButton  
para ejecutar  
alguna acción

# Componentes principales AWT/Swing

## Clases CheckBox/JCheckBox/JRadioButton

- Casillas de verificación. Ofrecen funcionalidad para activar y desactivar opciones.
- Los componentes JRadioButton los agruparemos en un **ButtonGroup** para conseguir seleccionar una (y sólo una) opción del grupo. ButtonGroup no es un componente visual.



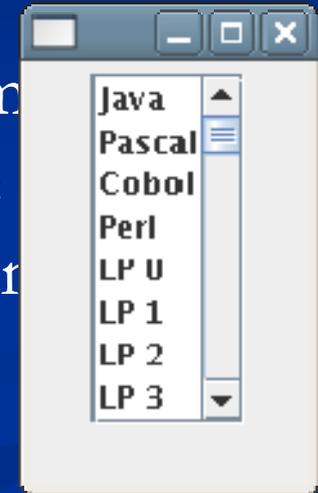
**JCheckBox /  
JRadioButton  
para activar  
opciones**

**ButtonGroup  
para escoger sólo  
una**

# AWT/Swing: componentes principales

## Clases List/JList

- Por medio de las listas desplegadas, mostraremos al usuario un grupo de opciones. A menudo se usan para evitar la saturación de información en pantalla.
- JList no dispone de scroll por defecto. Para ello, se debe añadir a un JScrollPane.



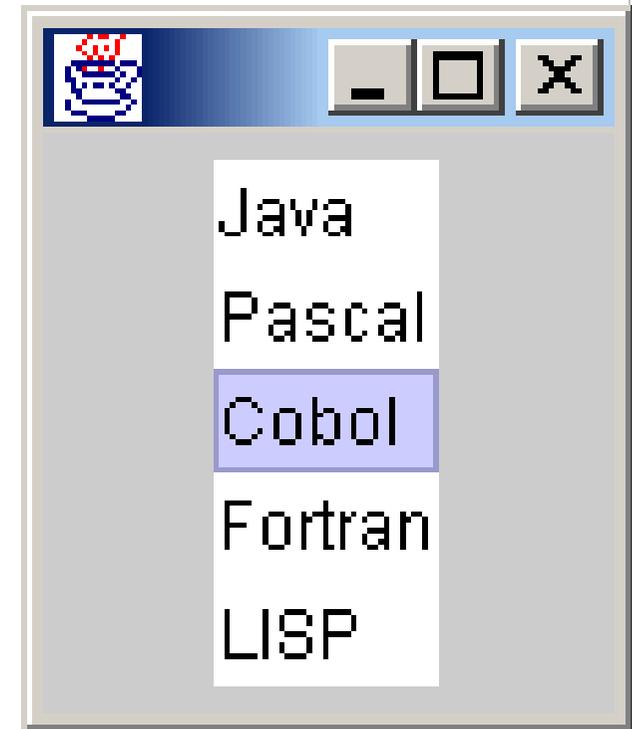
F:\alfredo\EjsJava\fuentes\ejsSwing>Listas.java

```
package ejsSwing;

import javax.swing.*.*;
import java.awt.*.*;
import java.util.*; // Porque se usa la clase Vector
public class Listas extends JFrame {
    JPanel jPanel1 = new JPanel();
    JList jList1; // El new se hará después
    Vector elementos = new Vector();

    public Listas() {
        this.getContentPane().add(jPanel1);
        elementos.addElement("Java");
        elementos.addElement("Pascal");
        elementos.addElement("Cobol");
        elementos.addElement("Fortran");
        jList1 = new JList(elementos);
        jPanel1.add(jList1);
        elementos.addElement("LISP"); //Añadiendo en el Vector se cambia el JList!!
        pack();
    }

    public static void main (String []arg) {
        Frame b = new Listas();
        b.setVisible(true);
    }
}
```



**Pero sólo si se hace antes de que se muestre el JFrame**

**b.elementos.addElement("C++");  
aquí no funcionaría**

**Con vector no funciona si la lista es "dinámica"**

```
package ejsSwing;
```

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

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

```
import java.util.*; // Porque se usa la clase Vector
```

```
public class ListasModificables extends JFrame {
```

```
    JPanel jPanel1 = new JPanel();
```

```
    JList jList1; // El new se hará después
```

```
    DefaultListModel elementos = new DefaultListModel();
```

```
    public ListasModificables() {
```

```
        this.getContentPane().add(jPanel1);
```

```
        elementos.addElement("Java");
```

```
        elementos.addElement("Pascal");
```

```
        elementos.addElement("Cobol");
```

```
        elementos.addElement("Fortran");
```

```
        jList1 = new JList(elementos);
```

```
        jPanel1.add(jList1);
```

```
        elementos.addElement("LISP"); //Añadiendo en el Vector se cambia el JList!!
```

```
        pack();
```

```
    }
```

```
    public static void main (String []arg) {
```

```
        ListasModificables b = new ListasModificables();
```

```
        b.setVisible(true);
```

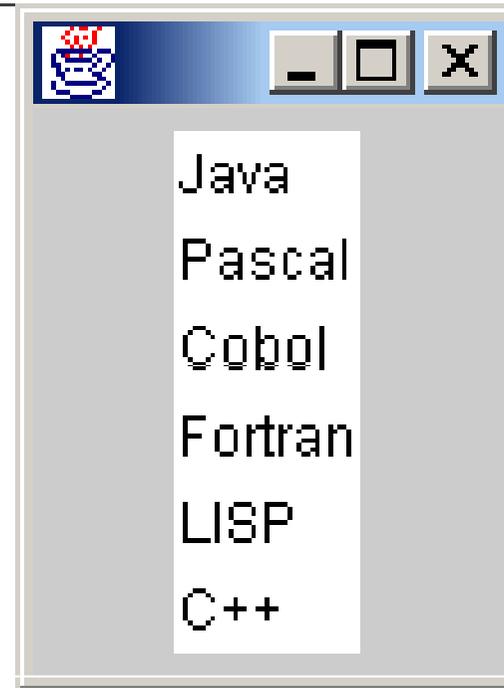
```
        b.elementos.addElement("C++");
```

```
        b.setVisible(true);
```

```
    }
```

```
}
```

```
}
```



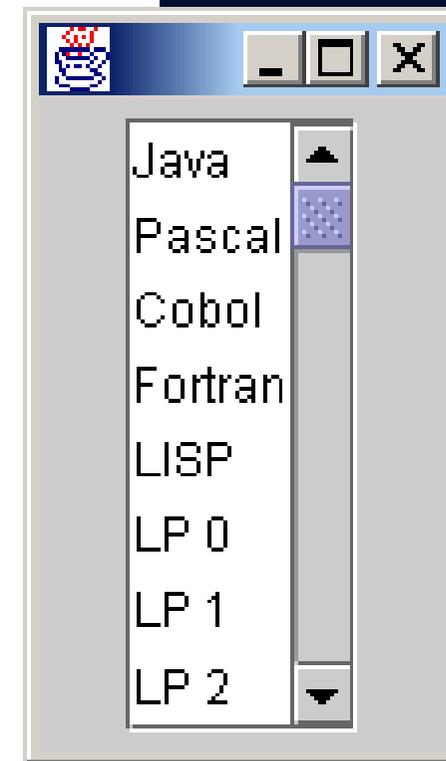
**Por ejemplo, javax.Swing ofrece la clase DefaultListModel, que proporciona los mismos métodos que Vector**

**Para listas “dinámicas” usar un ListModel**

```

package ejsSwing;
import javax.swing.*.*;
import java.awt.*.*;
import java.awt.event.*;
import java.awt.event.*;
public class ListasConScroll extends JFrame {
    JPanel jPanel1 = new JPanel();
    JList jList1; // El new se hará después
    DefaultListModel elementos = new DefaultListModel();
    int k=0;
    public ListasConScroll() {
        this.getContentPane().add(jPanel1);
        elementos.addElement("Java");
        elementos.addElement("Pascal");
        elementos.addElement("Cobol");
        elementos.addElement("Fortran");
        jList1 = new JList(elementos);
        JScrollPane j = new JScrollPane(jList1);
        // Se mete la lista en un Panel donde aparecen scroll
        // tanto vertical como horizontal si es necesario
        jPanel1.add(j); // Añadimos el Panel con scroll
        elementos.addElement("LISP"); //Añadiendo en el Vector se cambia el JList!
        for (int i=0;i<50;i++) elementos.addElement("LP "+i);
        //Añadir muchos para que no quepan y deba aparecer el scroll vertical
        pack();
    }
    public static void main (String []arg) {
        Frame b = new ListasConScroll();
        b.setVisible(true);
    }
}

```



**Listas con Scroll:  
añadir un JList a  
un JScrollPane**

# AWT/Swing: componentes principales

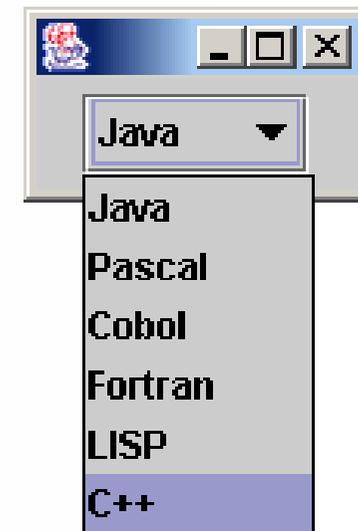
- **Clases Choice/JComboBox**
  - Son listas (desplegables) de opciones.
  - Las ventajas de esto: las listas de opciones no ocupan demasiado espacio en pantalla.

# Listas desplegadas, para escoger una opción

```
package ejsSwing;
import javax.swing.*.*;
import java.awt.*.*;
public class ComboBoxes extends JFrame {
    JPanel jPanel1 = new JPanel();
    JComboBox jComboBox1; // El new se hará después
    DefaultComboBoxModel elementos = new DefaultComboBoxModel();

    public ComboBoxes() {
        this.getContentPane().add(jPanel1);
        elementos.addElement("Java");
        elementos.addElement("Pascal");
        elementos.addElement("Cobol");
        elementos.addElement("Fortran");
        jComboBox1 = new JComboBox(elementos);
        jPanel1.add(jComboBox1);
        elementos.addElement("LISP");
        //Añadiendo en el DefaultComboBoxModel se cambia el JComboBox!!
        pack();
    }

    public static void main (String []arg) {
        ComboBoxes b = new ComboBoxes();
        b.setVisible(true);
        b.elementos.addElement("C++"); // Y aquí también se cambia el JComboBox
    }
}
```



# AWT/Swing: componentes principales

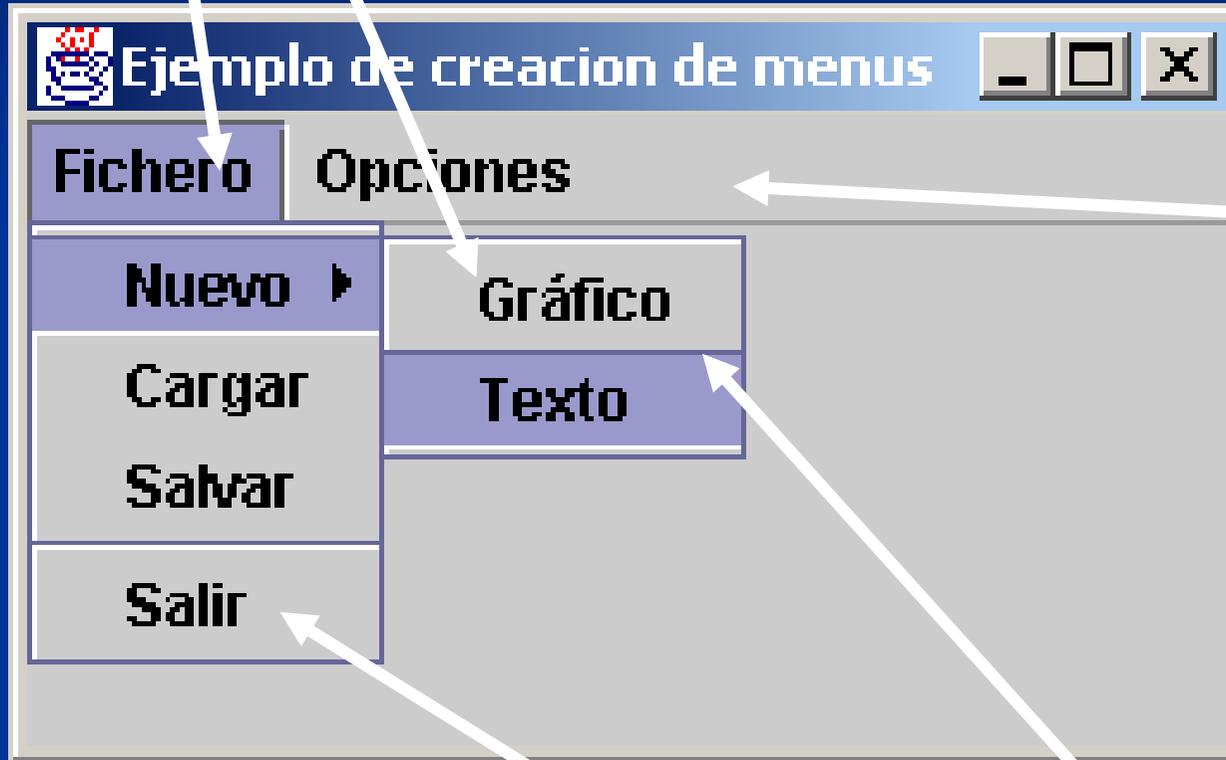
## Creación de Menús

- **En Swing**
  - El único contenedor que puede alojar una barra de menú es **JFrame** .
  - La clase **JMenuBar** crea la barra de menú donde se insertarán las opciones de dicho menú.
  - La clase **JMenu** es la encargada de crear los menús. Estos menús tienen un nombre asociado y muestran una lista desplegable con varios elementos.
  - Los elementos de un menú pueden ser objetos **JMenuItem** u objetos **JMenu** (para crear menús en cascada)

# Menús en Swing

JMenu

JMenu



JMenuBar

JMenuItem

JMenuItem

F:\alfredo\EjsJava\fuentes\ejsSwing\Menus.java

```
package ejsSwing;
import javax.swing.*.*;
public class Menus extends JFrame {
    JPanel jPanel1 = new JPanel();
    JMenuBar barra = new JMenuBar();
    JMenu fichero = new JMenu("Fichero");
    JMenu opciones = new JMenu("Opciones");
    JMenu nuevo = new JMenu("Nuevo");
    JMenuItem cargar = new JMenuItem("Cargar");
    JMenuItem salvar = new JMenuItem("Salvar");
    JMenuItem salir = new JMenuItem("Salir");
    JMenuItem alta = new JRadioButtonMenuItem("Alta resolución", true);
    JMenuItem sinFondo = new JRadioButtonMenuItem("Sin fondo", false);
    JMenuItem bn = new JRadioButtonMenuItem("Blanco y negro", false);
    ButtonGroup bg = new ButtonGroup();
    JMenuItem grafico = new JMenuItem("Gráfico");
    JMenuItem texto = new JMenuItem("Texto");
    public Menus() {
        this.setTitle("Ejemplo de creacion de menus");
        opciones.add(alta); opciones.add(sinFondo); opciones.add(bn);
        bg.add(alta); bg.add(sinFondo); bg.add(bn);
        nuevo.add(grafico); nuevo.add(texto);

        fichero.add(nuevo); fichero.add(cargar); fichero.add(salvar);
        fichero.addSeparator();
        fichero.add(salir);

        barra.add(fichero); barra.add(opciones);
        this.setJMenuBar(barra);
    }
    public static void main (String []arg) {
        JFrame b = new Menus();
        b.setVisible(true);
    }
}
```

Ejemplo de creacion de menus

Fichero Opciones

Nuevo ▶ Gráfico

Cargar Texto

Salvar

Salir

Ejemplo de creacion de menus

Fichero Opciones

Alta resolución

Sin fondo

Blanco y negro

# AWT/Swing: componentes principales

- Inserción de imágenes

- Meteremos la imagen en un JLabel

```
public class FrameIrudiekin extends JFrame {
    JButton jButton1 = new JButton();
    JButton jButton2 = new JButton();

    public FrameIrudiekin() {
        this.getContentPane().setLayout(null);
        this.setSize(new Dimension(400, 300));
        this.setTitle("Irudiak nola bistaratu");

        JLabel lb1 = new JLabel(new ImageIcon(getClass().getResource("katua.jpg")));
        this.getContentPane().add(lb1);
        lb1.setSize(lb1.getPreferredSize());
        lb1.setLocation(20,20);
        JLabel lb2 = new JLabel(new ImageIcon(getClass().getResource("armiarma.jpg")));
        this.getContentPane().add(lb2);
        lb2.setSize(lb2.getPreferredSize());
        lb2.setLocation(20,120);

        jButton1.setText("Katua erosi");
        jButton1.setBounds(new Rectangle(180, 40, 160, 30));
        this.getContentPane().add(jButton1, null);
        jButton2.setText("Armiarma erosi");
        jButton2.setBounds(new Rectangle(180, 140, 160, 30));
        this.getContentPane().add(jButton2, null);
    }
}
```

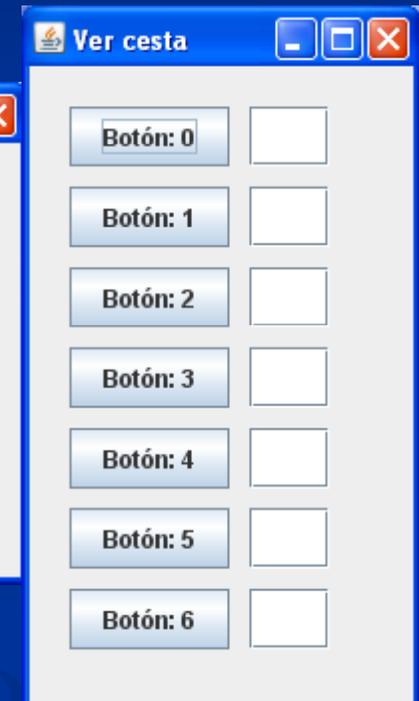
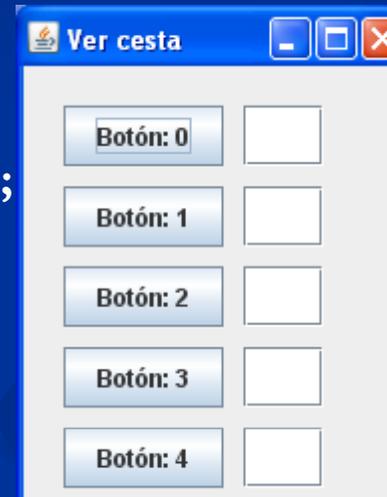


# AWT/Swing: componentes principales

## Creación dinámica de componentes

```
import java.awt.*;
import javax.swing.*;
public class CestaCompra extends JFrame {
    int N=5;
    JButton[] botones= new JButton[N];
    JTextField[] cajasTexto = new JTextField[N];
    public CestaCompra(){
        for (int i=0; i<N; ++i){
            botones[i] = new JButton();
            cajasTexto[i] = new JTextField();
            this.getContentPane().setLayout(null);
            this.setSize(new Dimension(200, N*50));
            this.setTitle("Ver cesta");
            for (int i=0; i<N; ++i){
                botones[i].setBounds(new Rectangle(20, 20+i*40, 80, 30));
                botones[i].setText("Botón: "+i);
                this.getContentPane().add(botones[i], null);
                cajasTexto[i].setBounds(new Rectangle(110, 20+i*40, 40, 30));
                this.getContentPane().add(cajasTexto[i], null); }}}}
```

int N=7



# Gestores de Diseño: Layout

- Se usan para definir dónde colocar un componente dentro de un contenedor.

```
contenedor.add( componente );
```

- **FlowLayout** (Gestor predeterminado para Panel)
  - Los componentes se van añadiendo a una línea. Al completar la línea, se pasa a la siguiente.
- **BorderLayout** (Gestor predeterminado para Frame y Dialog)
  - Los componentes se añaden en una de estas 5 zonas: norte, sur, este, oeste y centro.
- Se puede cambiar el gestor predeterminado:

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

# Gestores de Diseño: Layout

- Si se quiere poner el componente en unas coordenadas concretas, se debe eliminar el gestor de diseño.

```
contenedor.setLayout(null);
```

- Sean **this** un contenedor y **textField1** uno de sus componentes:

```
setLayout(null);
```

```
textField1.setBounds(15,20,50,60);
```

x

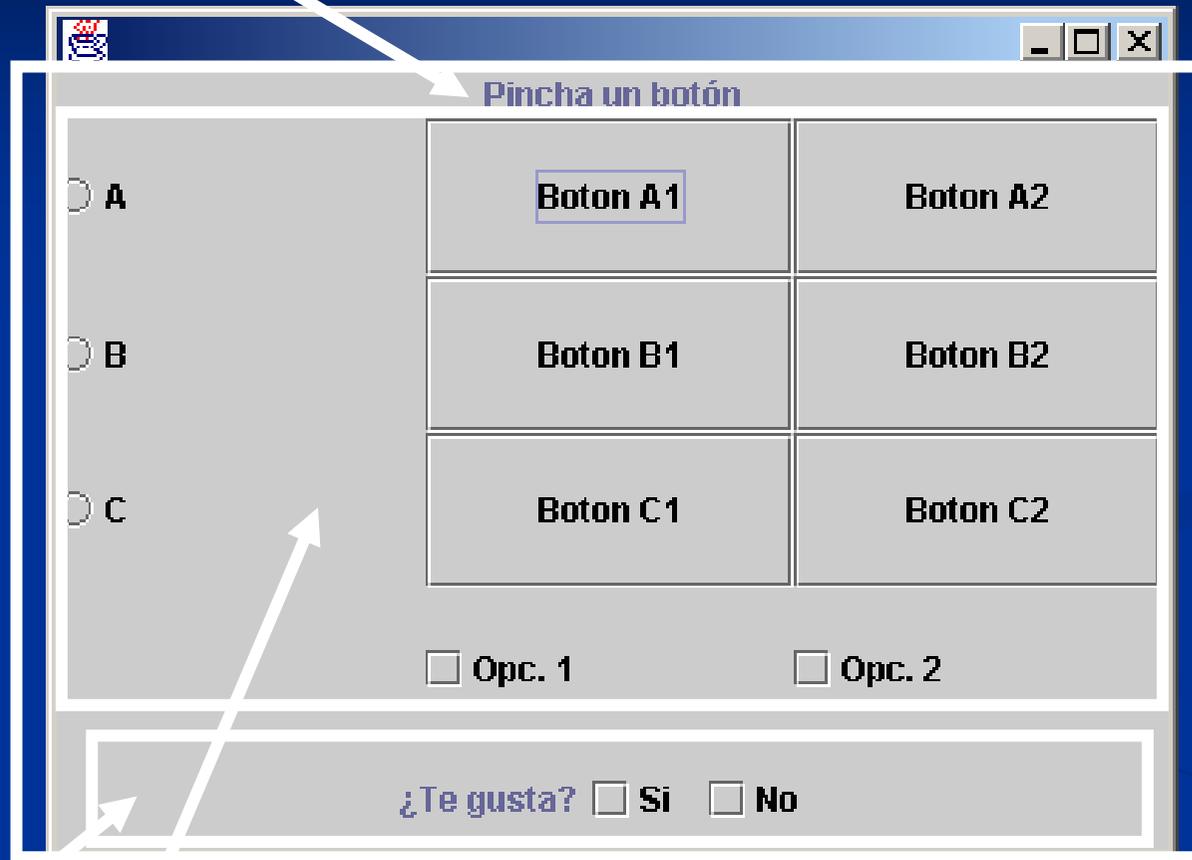
y

ancho

alto

Label  
en BorderLayout.NORTH

Panel  
con BorderLayout



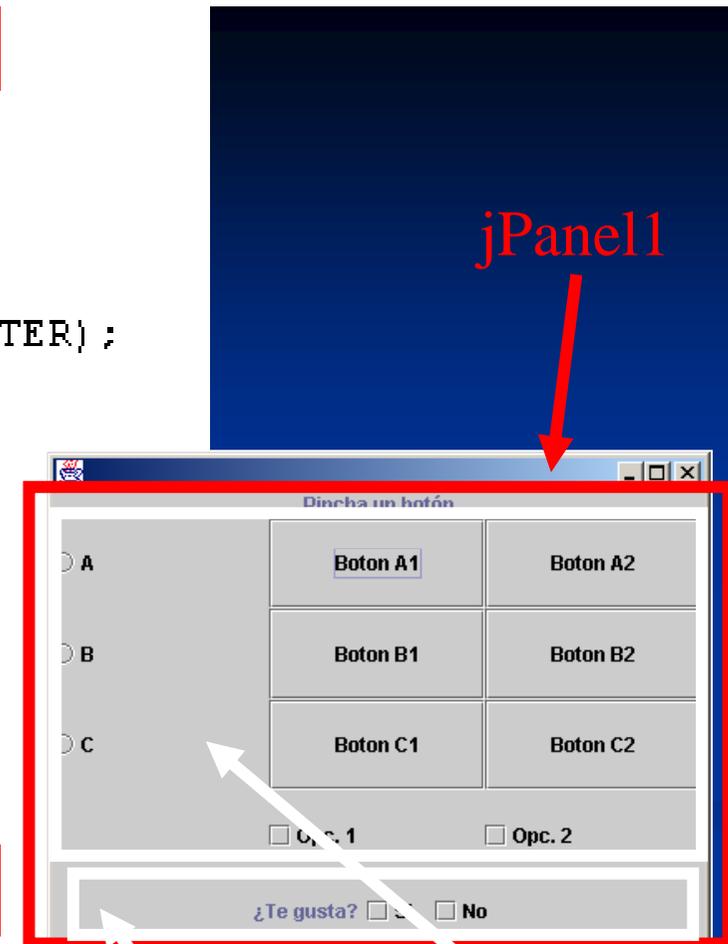
Panel con GridLayout(4,3) situado en BorderLayout.CENTER

Panel con FlowLayout situado en BorderLayout.SOUTH

```

jPanel1.setLayout(new BorderLayout());
jCheckBox1.setLabel("Opc. 1");
jCheckBox2.setLabel("Opc. 2");
jPanel2.setLayout(new FlowLayout());
jLabel1.setText("Pincha un botón");
jLabel1.setHorizontalAlignment(JLabel.CENTER);
jRadioButton1.setLabel("C");
jRadioButton2.setLabel("B");
jRadioButton3.setText("A");
jButton2.setText("Boton A1");
...
jLabel2.setText("¿Te gusta?");
jCheckBox3.setLabel("No");
jCheckBox4.setLabel("Si");
jPanel4.setLayout(new GridLayout(4,3));
jPanel1.add(jPanel2, BorderLayout.SOUTH);
jPanel2.add(jLabel2, null);
jPanel2.add(jCheckBox4, null);
jPanel2.add(jCheckBox3, null);
jPanel1.add(jLabel1, BorderLayout.NORTH);
jPanel1.add(jPanel4, BorderLayout.CENTER);
jPanel4.add(jRadioButton1, null);
jPanel4.add(jRadioButton2, null);
jPanel4.add(jRadioButton3, null);
jPanel4.add(jButton2, null);
...

```



Panel2

jPanel4

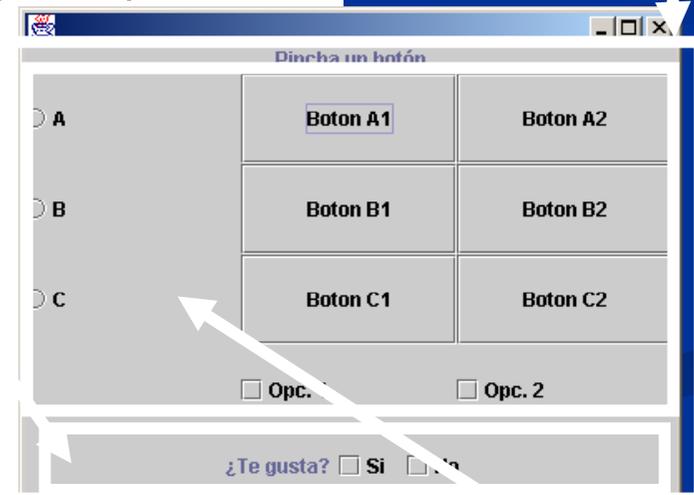
```

jPanel1.setLayout(new BorderLayout());
jCheckBox1.setLabel("Opc. 1"); jCheckBox2.setLabel("Opc. 2");
jPanel2.setLayout(new FlowLayout());
jLabel1.setText("Pincha un botón");
jLabel1.setHorizontalAlignment(JLabel.CENTER);
jRadioButton1.setLabel("C"); jRadioButton2.setLabel("B");
jRadioButton3.setText("A"); jButton2.setText("Boton A1");
jButton3.setLabel("Boton C2"); jButton4.setLabel("Boton B2");
jButton5.setLabel("Boton A2"); jButton6.setLabel("Boton B1");
jButton7.setLabel("Boton C1"); jLabel2.setText("¿Te gusta?");
jCheckBox3.setLabel("No"); jCheckBox4.setLabel("Si");
jPanel4.setLayout(new GridLayout(4,3));
this.getContentPane().add(jPanel1, BorderLayout.CENTER);
jPanel1.add(jPanel2, BorderLayout.SOUTH); jPanel2.add(jLabel2, null);
jPanel2.add(jCheckBox4, null); jPanel2.add(jCheckBox3, null);
jPanel1.add(jLabel1, BorderLayout.NORTH);
jPanel1.add(jPanel4, BorderLayout.CENTER);
jPanel4.add(jRadioButton3, null);
jPanel4.add(jButton2, null);
jPanel4.add(jButton5, null);
jPanel4.add(jRadioButton2, null);
jPanel4.add(jButton6, null);
jPanel4.add(jButton4, null);
jPanel4.add(jRadioButton1, null);
jPanel4.add(jButton7, null);
jPanel4.add(jButton3, null); jPanel4.add(jPanel3, null);
jPanel4.add(jCheckBox1, null); jPanel4.add(jCheckBox2, null);

```

jPanel1

jPanel2



jPanel4

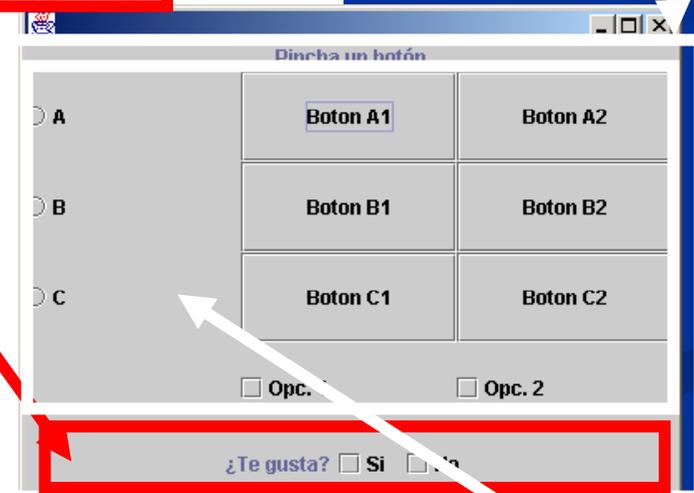
```

jPanel1.setLayout(new BorderLayout());
jCheckBox1.setLabel("Opc. 1"); jCheckBox2.setLabel("Opc. 2");
jPanel2.setLayout(new FlowLayout());
jLabel1.setText("Pincha un botón");
jLabel1.setHorizontalAlignment(JLabel.CENTER);
jRadioButton1.setLabel("C"); jRadioButton2.setLabel("B");
jRadioButton3.setText("A"); jButton2.setText("Boton A1");
jButton3.setLabel("Boton C2"); jButton4.setLabel("Boton B2");
jButton5.setLabel("Boton A2"); jButton6.setLabel("Boton B1");
jButton7.setLabel("Boton C1"); jLabel2.setText("¿Te gusta?");
jCheckBox3.setLabel("No"); jCheckBox4.setLabel("Si");
jPanel4.setLayout(new GridLayout(4,3));
this.getContentPane().add(jPanel1, BorderLayout.CENTER);
jPanel1.add(jPanel2, BorderLayout.SOUTH); jPanel2.add(jLabel2, null);
jPanel2.add(jCheckBox4, null); jPanel2.add(jCheckBox3, null);
jPanel1.add(jLabel1, BorderLayout.NORTH);
jPanel1.add(jPanel4, BorderLayout.CENTER);
jPanel4.add(jRadioButton3, null);
jPanel4.add(jButton2, null);
jPanel4.add(jButton5, null);
jPanel4.add(jRadioButton2, null);
jPanel4.add(jButton6, null);
jPanel4.add(jButton4, null);
jPanel4.add(jRadioButton1, null);
jPanel4.add(jButton7, null);
jPanel4.add(jButton3, null); jPanel4.add(jPanel3, null);
jPanel4.add(jCheckBox1, null); jPanel4.add(jCheckBox2, null);

```

jPanel1

jPanel2



jPanel4

```

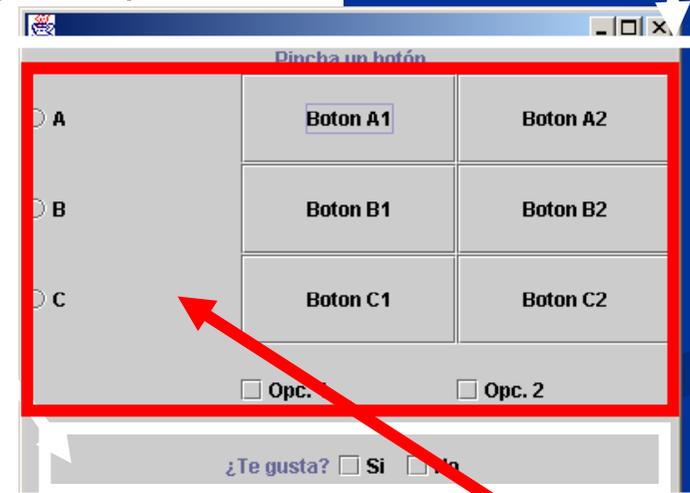
jPanell.setLayout(new BorderLayout());
jCheckBox1.setLabel("Opc. 1"); jCheckBox2.setLabel("Opc. 2");
jPanel2.setLayout(new FlowLayout());
jLabell.setText("Pincha un botón");
jLabell.setHorizontalAlignment(JLabel.CENTER);
jRadioButton1.setLabel("C"); jRadioButton2.setLabel("B");
jRadioButton3.setText("A"); jButton2.setText("Boton A1");
jButton3.setLabel("Boton C2"); jButton4.setLabel("Boton B2");
jButton5.setLabel("Boton A2"); jButton6.setLabel("Boton B1");
jButton7.setLabel("Boton C1"); jLabel2.setText("¿Te gusta?");
jCheckBox3.setLabel("No"); jCheckBox4.setLabel("Si");
jPanel4.setLayout(new GridLayout(4,3));
this.getContentPane().add(jPanell, BorderLayout.CENTER);
jPanell.add(jPanel2, BorderLayout.SOUTH); jPanel2.add(jLabel2, null);
jPanel2.add(jCheckBox4, null); jPanel2.add(jCheckBox3, null);
jPanell.add(jLabell, BorderLayout.NORTH);
jPanell.add(jPanel4, BorderLayout.CENTER);
jPanel4.add(jRadioButton3, null);
jPanel4.add(jButton2, null);
jPanel4.add(jButton5, null);
jPanel4.add(jRadioButton2, null);
jPanel4.add(jButton6, null);
jPanel4.add(jButton4, null);
jPanel4.add(jRadioButton1, null);
jPanel4.add(jButton7, null);
jPanel4.add(jButton3, null); jPanel4.add(jPanel3, null);
jPanel4.add(jCheckBox1, null); jPanel4.add(jCheckBox2, null);

```

```
JPanel jPanel3 = new JPanel();
```

jPanel1

jPanel2



jPanel4

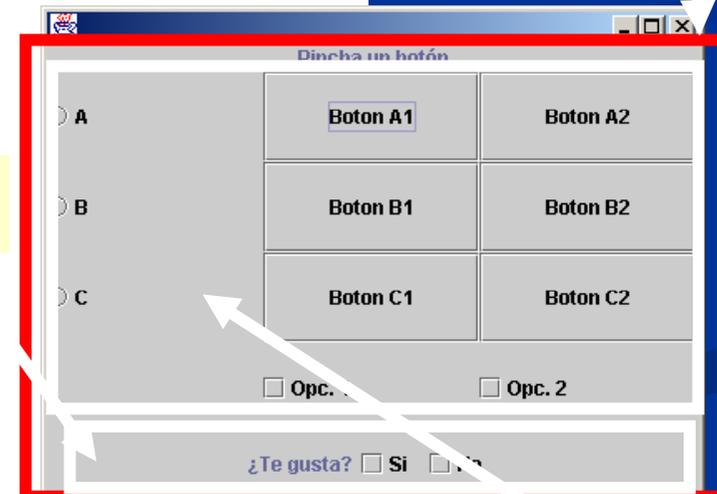
```

jPanell.setLayout(new BorderLayout());
jCheckBox1.setLabel("Opc. 1"); jCheckBox2.setLabel("Opc. 2");
jPanel2.setLayout(new FlowLayout());
jLabell.setText("Pincha un botón");
jLabell.setHorizontalAlignment(JLabel.CENTER);
jRadioButton1.setLabel("C"); jRadioButton2.setLabel("B");
jRadioButton3.setText("A"); jButton2.setText("Boton A1");
jButton3.setLabel("Boton C2"); jButton4.setLabel("Boton B2");
jButton5.setLabel("Boton A2"); jButton6.setLabel("Boton B1");
jButton7.setLabel("Boton C1"); jLabel2.setText("¿Te gusta?");
jCheckBox3.setLabel("No"); jCheckBox4.setLabel("Si");
jPanel4.setLayout(new GridLayout(4,3));
this.getContentPane().add(jPanell, BorderLayout.CENTER);
jPanell.add(jPanel2, BorderLayout.SOUTH); jPanel2.add(jLabel2, null);
jPanel2.add(jCheckBox4, null); jPanel2.add(jCheckBox3, null);
jPanell.add(jLabell, BorderLayout.NORTH);
jPanell.add(jPanel4, BorderLayout.CENTER);
jPanel4.add(jRadioButton3, null);
jPanel4.add(jButton2, null);
jPanel4.add(jButton5, null);
jPanel4.add(jRadioButton2, null);
jPanel4.add(jButton6, null);
jPanel4.add(jButton4, null);
jPanel4.add(jRadioButton1, null);
jPanel4.add(jButton7, null);
jPanel4.add(jButton3, null); jPanel4.add(jPanel3, null);
jPanel4.add(jCheckBox1, null); jPanel4.add(jCheckBox2, null);

```

jPanell

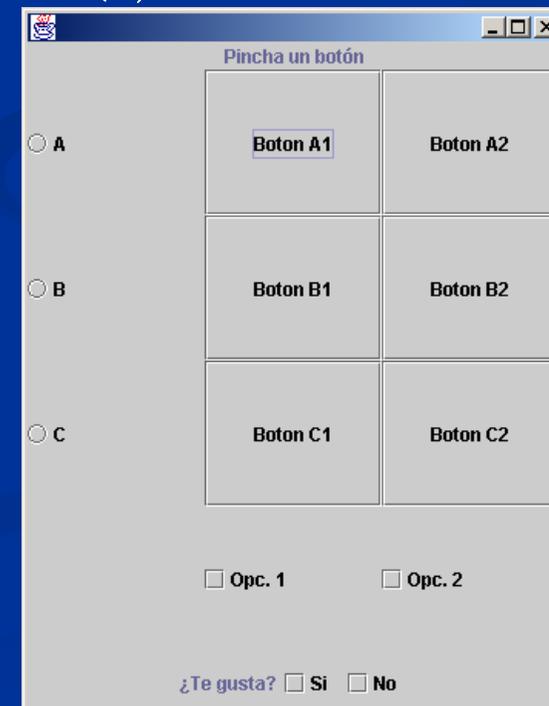
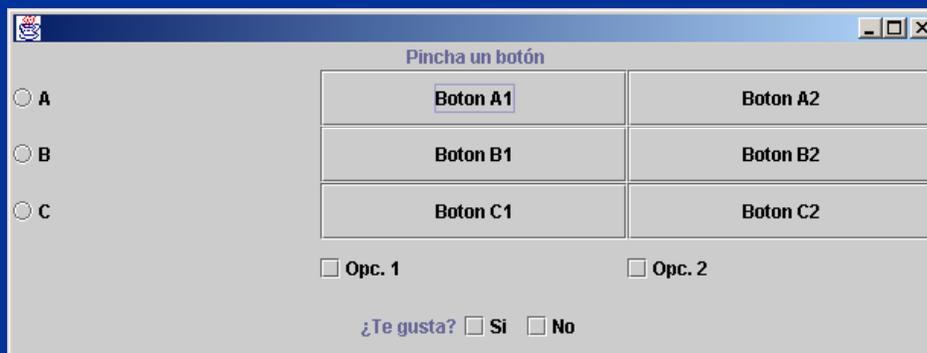
jPanel2



jPanel4

# Gestores de Diseño: Layout

- Ventajas de definir un GUI con gestor de diseño:
  - los componentes se redibujan automáticamente al ajustar el tamaño de la ventana (ajustándose al tamaño disponible).

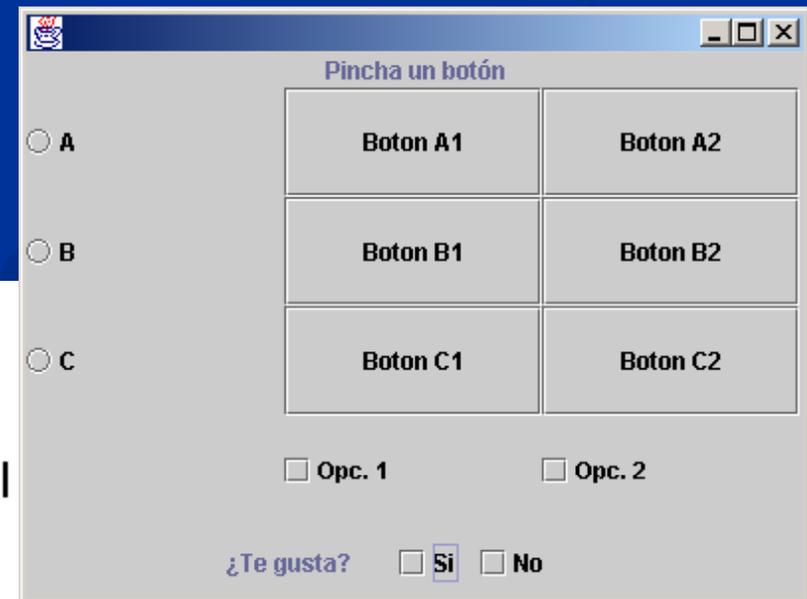


# Gestores de Diseño: Layout

Sin gestor de diseño hay que proporcionar las coordenadas de todos los componentes

- fácil de hacer con herramientas visuales.

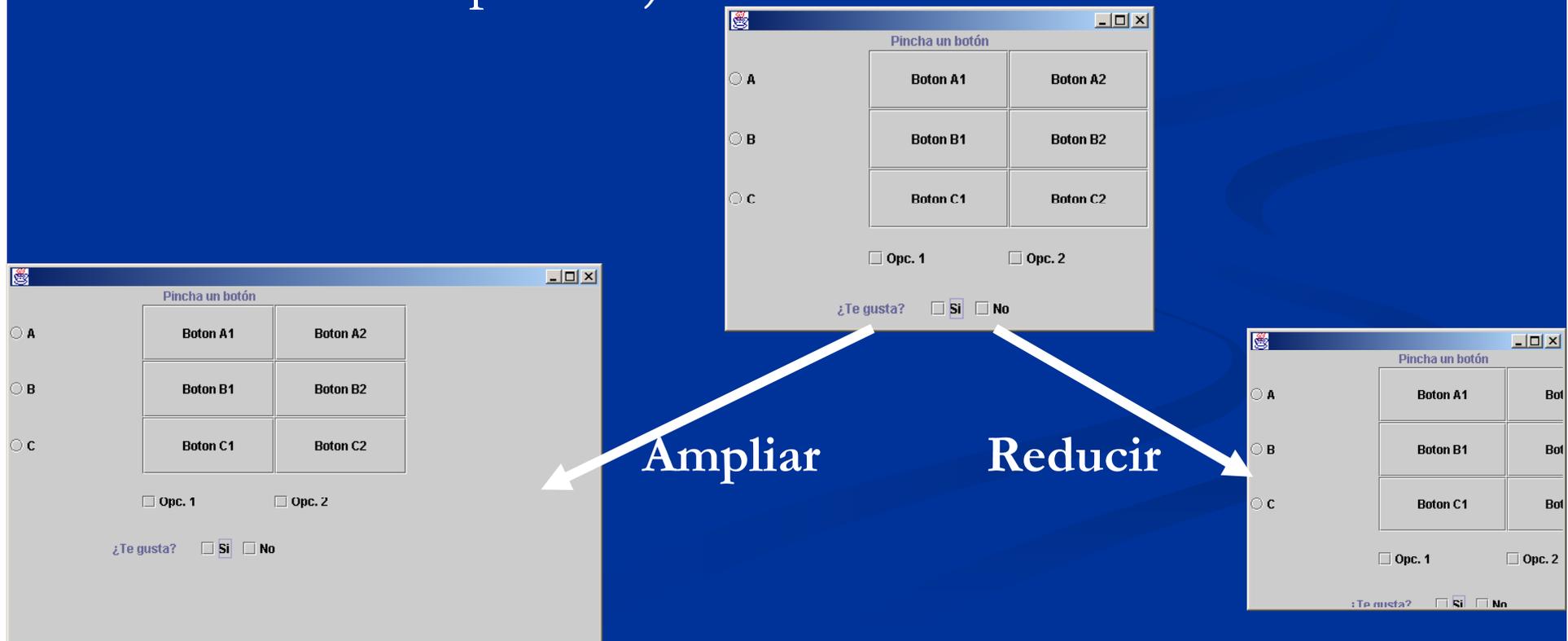
```
this.setSize(new Dimension(400, 300));
jPanel1.setBounds(new Rectangle(0, 0, 392, 273));
jPanel1.setLayout(null);
jCheckBox1.setLabel("Opc. 1");
jCheckBox1.setBounds(new Rectangle(130, 165, 130, 55));
jCheckBox2.setLabel("Opc. 2");
jCheckBox2.setBounds(new Rectangle(260, 165, 130, 55));
jPanel3.setBounds(new Rectangle(0, 165, 392, 273));
jPanel3.setLayout(null);
jPanel2.setBounds(new Rectangle(0, 238, 392, 35));
jPanel2.setLayout(null);
```



# Gestores de Diseño: Layout

Desventajas de no usar un gestor de diseño:

- Al redimensionar el Frame, los componentes se mantienen sin cambiar su posición (no se ajustan al tamaño disponible).



# Otros contenedores

## ■ Clases Dialog/JDialog

- Ventana que permite leer datos del usuario
- Puede asignársele la característica de ser MODAL, para no permitir cambiar a otra ventana mientras esté activa.

## ■ Clase FileDialog (Sólo en AWT)

- Ventana que permite abrir/guardar ficheros (modos FileDialog.LOAD y FileDialog.SAVE)
- Se utiliza la misma ventana de diálogo del Sistema Operativo en el que se está ejecutando la máquina virtual Java
- Ofrece ya cierta funcionalidad. No hay que reprogramar el caso en el que se intenta sobrescribir un fichero (muestra ventana de alerta)

## ■ En Swing es JFileChooser

```

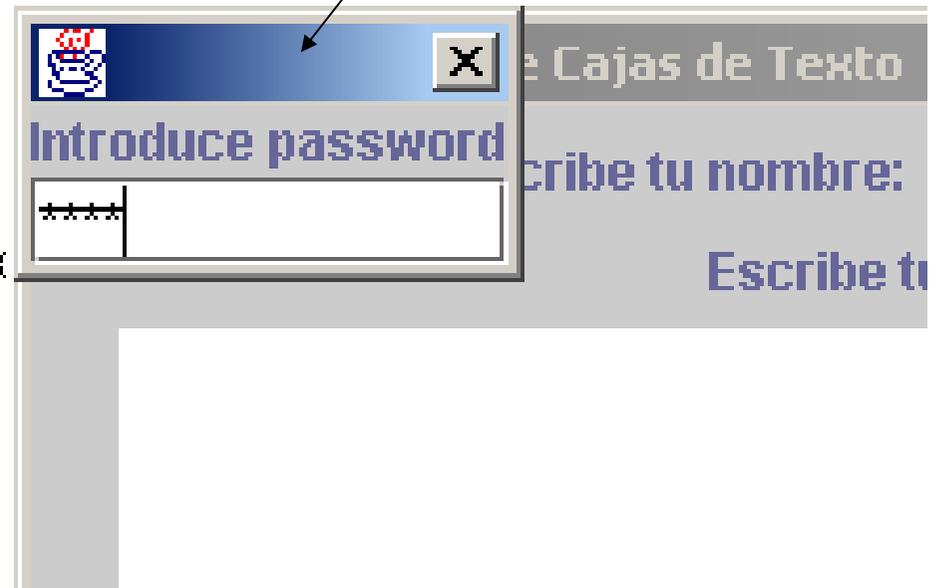
package ejsSwing;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DialogPassword extends JDialog {
    BorderLayout borderLayout1 = new BorderLayout();
    JPanel jPanel1 = new JPanel();
    JPasswordField jPasswordField1 = new JPasswordField();
    JLabel jLabel1 = new JLabel();
    public DialogPassword(Frame parent, String title, boolean modal) {
        super(parent, title, modal);
        jPanel1.setLayout(borderLayout1);
        jLabel1.setText("Introduce password");
        getContentPane().add(jPanel1);
        jPanel1.add(jPasswordField1, BorderLayout.CENTER);
        jPanel1.add(jLabel1, BorderLayout.NORTH);
        pack();
    }
    public DialogPassword() {
        this(null, "", true);
    }
    public static void main (String []arg) {
        Frame a = new CajasTexto();
        JDialog b = new DialogPassword();
        a.setVisible(true);
        b.setVisible(true);
    }
}

```

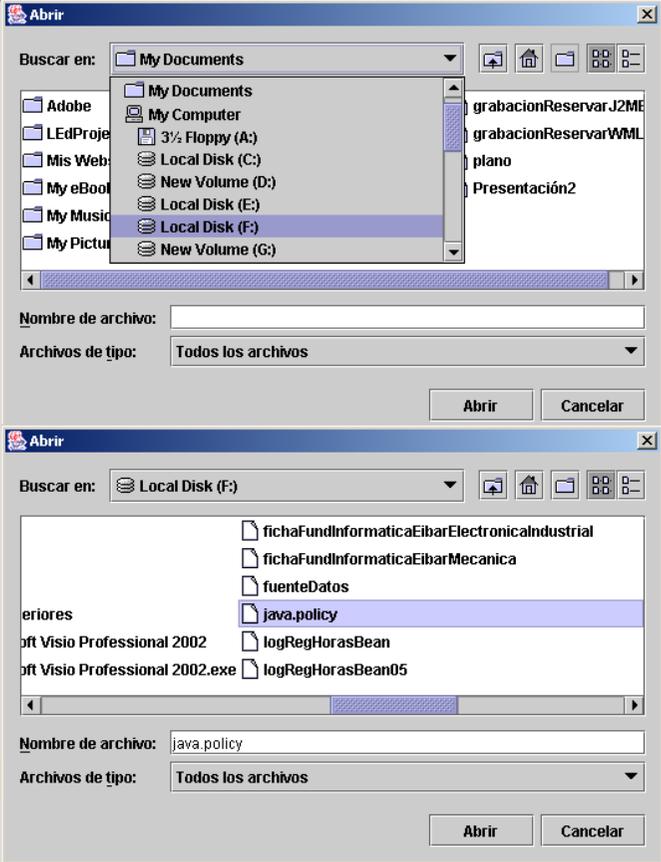
**Ventana MODAL: hasta que no se cierre no puede cambiarse de ventana**

**modal  
a true**



# JFileChooser

```
F:\alfredo2003_04\Docencia\ISO\EjsJava\jdev9i\fuente\ejswing\EscogerFichero.java
1 package ejswing;
2 import javax.swing.*;
3
4
5 public class EscogerFichero
6 {
7     public EscogerFichero()
8     {
9     }
10
11    public static void main(String[] args)
12    {
13        JFileChooser j = new JFileChooser();
14        j.showOpenDialog(null);
15        System.out.println("Escogido: "+j.getSelectedFile());
16    }
17 }
```



Abriendo el diálogo de selección de archivos en el código de ejemplo, se muestran dos capturas de pantalla de un explorador de archivos. La primera captura muestra el diálogo de selección de archivos con el directorio 'My Documents' seleccionado. La segunda captura muestra el diálogo de selección de archivos con el disco local 'Local Disk (F:)' seleccionado y el archivo 'java.policy' resaltado.

```
Messages | ejemsTemaA3N_Swing.jpr | ejemsTemaA3N_Swing.jpr
Process exited.
C:\Program Files\Java\j2rel.4.0_03\bin\javaw.exe -client -classpath F:\alfredo2003_04\Docencia\ISO\l
Escogido: F:\java.policy
```

# Gestión de Eventos

- Al diseñar una interfaz gráfica debemos tener en cuenta que a consecuencia de las acciones del usuario se generarán distintos **eventos**.
- Se deben programar **métodos** para responder a estos **eventos** provocados por el usuario.
- Un evento:
  - es generado por una acción del usuario.
  - está ligado a un componente del GUI.
  - Ejemplos:
    - pulsar una tecla, mover el ratón, cambiar el tamaño de una ventana, cerrarla, minimizarla, pulsar un botón, perder u obtener el foco de un componente, cambiar el valor de un campo de texto, elegir una opción de menú...

# Gestión de Eventos

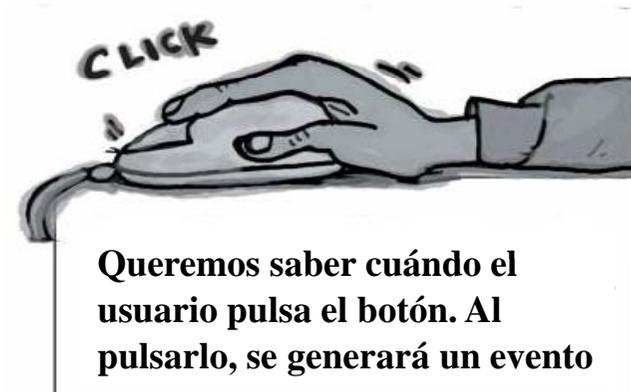
```
import javax.swing.*;

public class GUISimple extends JFrame {

    JButton button;

    public void ejecuta(){
        button = new JButton("Pulsa");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().add(button);
        setSize(300,300);
        setVisible(true);
    }

    public static void main(String[] args){
        SimpleGUI frame = new GUISimple();
        frame.setTitle("Gestión de Eventos");
        frame.ejecuta();
    }
}
```

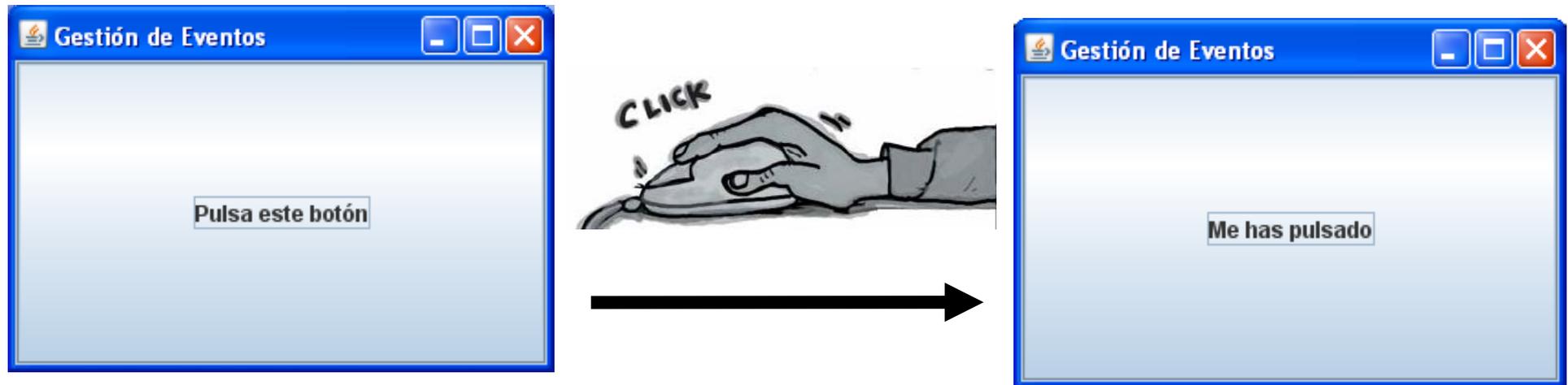


Queremos saber cuándo el usuario pulsa el botón. Al pulsarlo, se generará un evento

# Gestión de Eventos

Si queremos hacer algo cuando se pulse el botón:

- 1) deberemos programar un **método**, para responder al evento que se genera.
- 2) tendremos que saber **cuándo** se genera el evento.



```
jButton.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent e) {  
        jButton.setText("Me has pulsado"); // TODO Auto-generated  
    }  
});
```

# Los eventos SON OBJETOS

- Eventos de bajo nivel
  - Relacionados con aspectos físicos de la interfaz de usuario. Ejemplos: pulsación de teclas, movimientos de ratón, hacer click, ganar/perder el foco en un componente, abrir/cerrar ventana..
- Eventos de alto nivel o semánticos
  - Tienen que ver con la semántica de los componentes. Ejemplos: pulsar un botón, cambiar el texto de un campo, seleccionar un item en un menú/lista/choice
  - Generalmente combinaciones de eventos de bajo nivel.

**MEJOR TRATAR EVENTOS SEMÁNTICOS**

**(Por ejemplo: definir `actionPerformed`)**

# Interfaces Listener

- Para gestionar eventos, Java proporciona unas interfaces “oyentes” (Listeners), donde cada una de ellas contiene métodos que hay que implementar.
- La implementación proporcionada para cada método es la respuesta apropiada a cada evento.

# obj1.addXXXListener(obj2);

**obj1** contiene una referencia a un OBJETO GRÁFICO (botón, ventana, lista desplegable, checkbox, etc) sobre el que se quiere definir un comportamiento ante EVENTOS

**obj2** contiene una referencia a un objeto de una clase que implementa el interface XXXListener.

```
public interface XXXListener {  
    ....  
    void accionYYY (AWTEvent e);  
}
```

En tiempo de ejecución sucede un EVENTO que afecta al objeto de **obj1**

Se crea un objeto evento (AWTEvent) y se pasa el control al objeto oyente (el de **obj2**)

El objeto oyente ejecutará el método correspondiente a la acción (Ej.: **accionYYY**) usando el objeto Evento generado como parámetro

```

package ejsSwing;

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

public class EventoBoton extends JFrame {
    BorderLayout borderLayout1 = new BorderLayout();
    JPanel jPanel1 = new JPanel();
    JButton jButton1 = new JButton();

    public EventoBoton() {
        this.setSize(new Dimension(400, 100));
        jButton1.setLabel("Escribir HOLA");
        jButton1.addActionListener(new OyenteBoton());
        jPanel1.setLayout(borderLayout1);
        this.getContentPane().add(jPanel1, BorderLayout.CENTER);
        jPanel1.add(jButton1, BorderLayout.NORTH);
    }

    public static void main (String []arg) {
        Frame b = new EventoBoton();
        b.setVisible(true);
    }
}

```

```

class OyenteBoton implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        jPanel1.add(new JLabel("HOLA",JLabel.CENTER), BorderLayout.CENTER);
        jButton1.disable();
        jButton1.setText("Me han desactivado ... ");
        setVisible(true);
    }
}

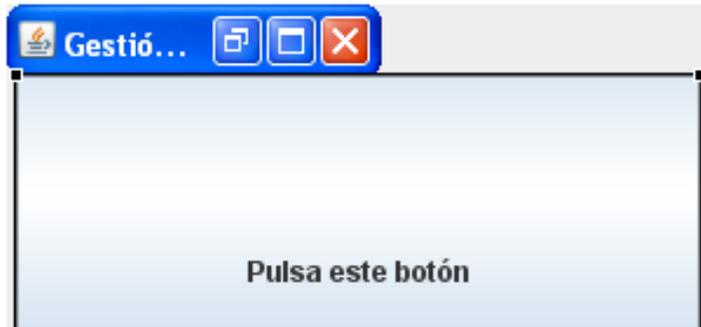
```



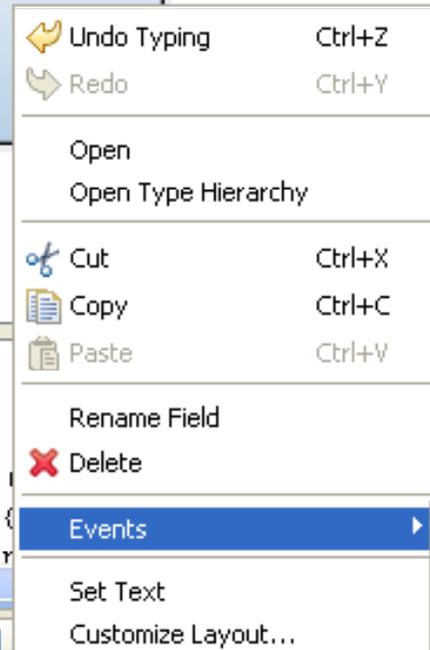
Tras pulsar  
el botón



OyenteBoton es INNER CLASS (clase definida dentro de otra).  
Por eso se puede acceder a los atributos jPanel1, jButton1,...



1) Hacer click derecho en el componente



2) Seleccionar el Evento (actionPerformed) y hacer click

3) Escribir el código donde indica Eclipse

```
return javax.swing.JPanel  
  
private JPanel getJContentPane()  
if (jContentPane == null) {  
    jContentPane = new JPanel
```

```
actionPerformed()
```

```
jButton.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(java.awt.event.ActionEvent e) {  
        System.out.println("actionPerformed()"); // TODO Auto-generated  
    }  
});
```

## JDeveloper genera las clases Oyente como INNER CLASS sin nombre

```
package ejsSwing;

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

public class EventoBoton extends JFrame {
    BorderLayout borderLayout1 = new BorderLayout();
    JPanel jPanel1 = new JPanel();
    JButton jButton1 = new JButton();

    public EventoBoton() {
        this.setSize(new Dimension(400, 100));
        jButton1.setLabel("Escribir HOLA");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                jButton1_actionPerformed(e);
            }
        });
        jPanel1.setLayout(borderLayout1);
        this.getContentPane().add(jPanel1, BorderLayout.CENTER);
        jPanel1.add(jButton1, BorderLayout.NORTH);
    }

    void jButton1_actionPerformed(ActionEvent e) {
        jPanel1.add(new JLabel("HOLA",JLabel.CENTER), BorderLayout.CENTER);
        jButton1.disable();
        jButton1.setText("Me han desactivado ... ");
        setVisible(true);
    }

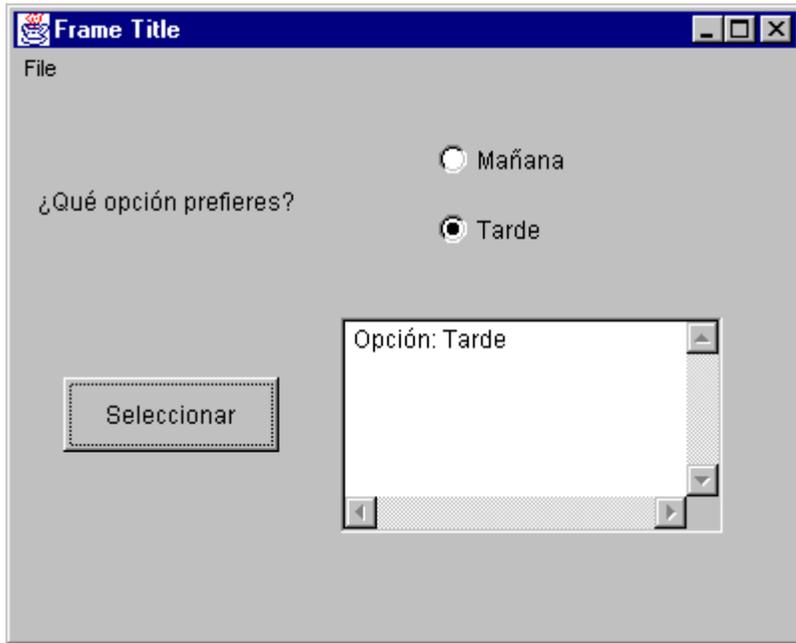
    public static void main (String []arg) {
        Frame b = new EventoBoton();
        b.setVisible(true);
    }
}
```



¿Qué es esto?

Tras pulsar el botón



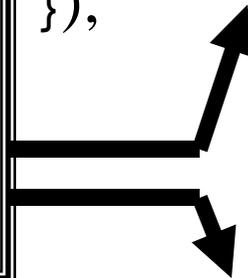


Definición de una clase  
SIN NOMBRE que  
implementa el interface  
**ActionListener**



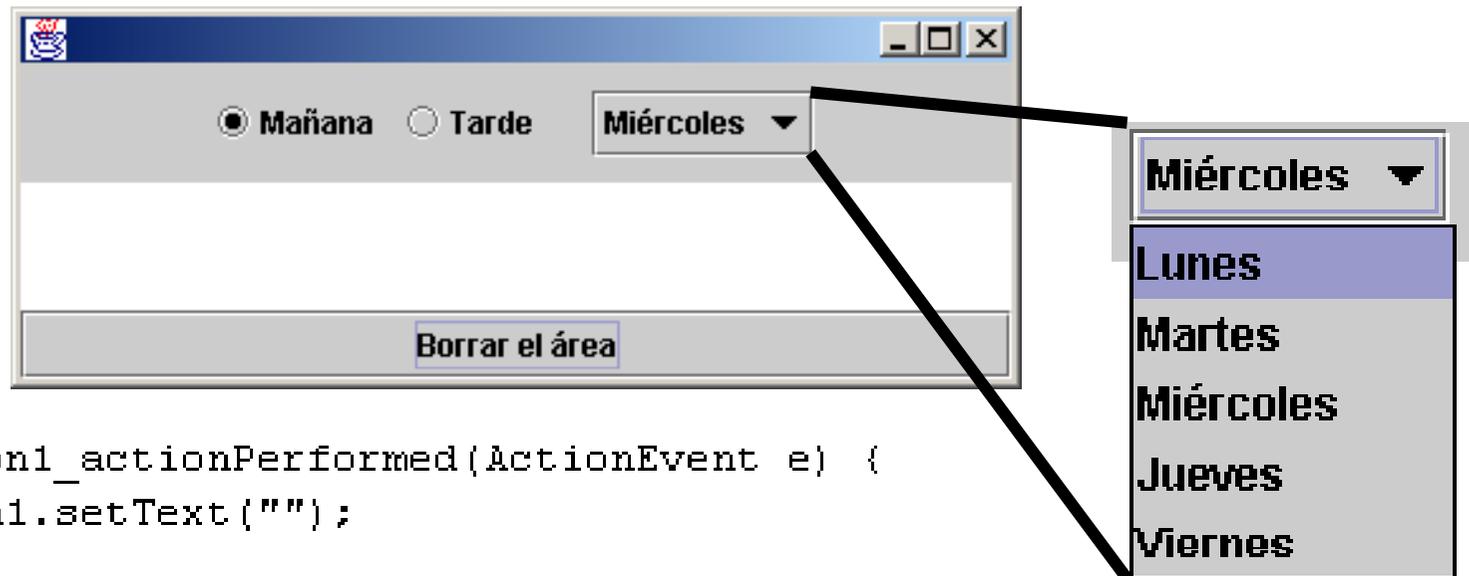
```
button1.addActionListener(new java.awt.event.ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        button1_actionPerformed(e);  
    }  
});
```

**ESTE CÓDIGO  
SE GENERA DE  
MANERA  
AUTOMÁTICA**

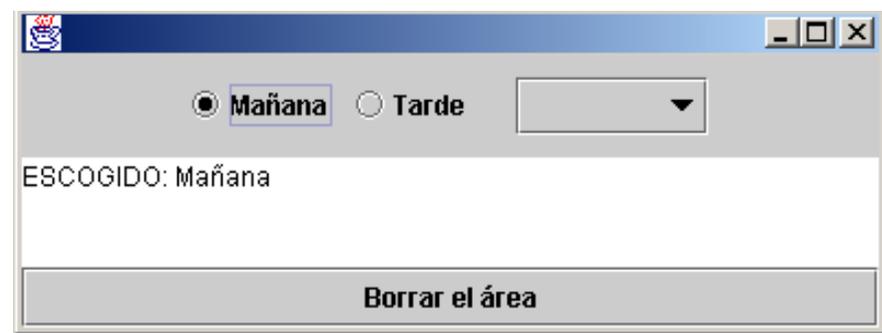


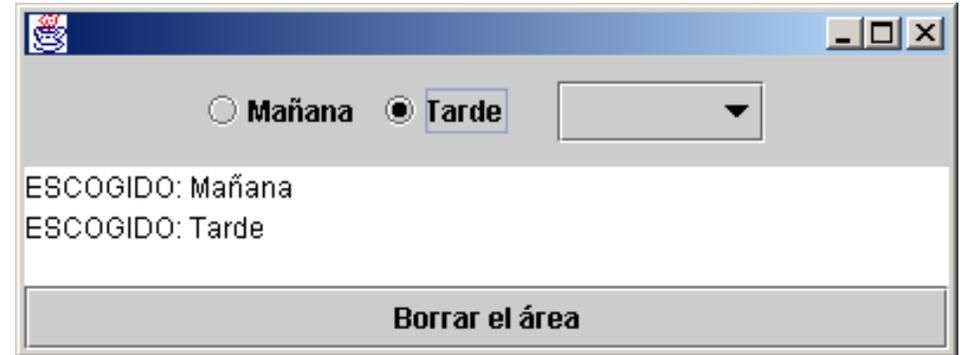
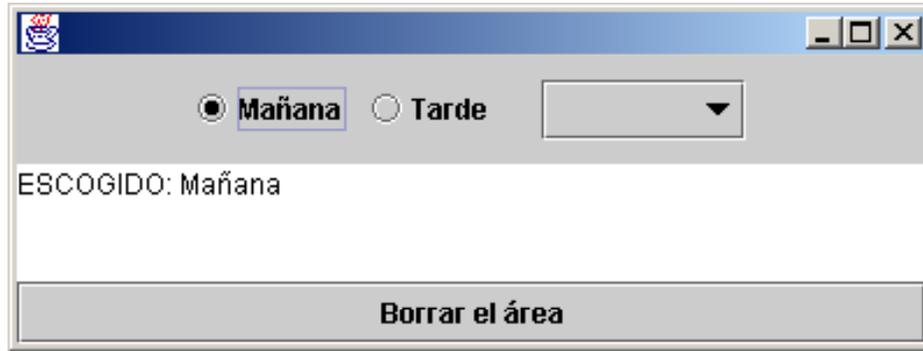
**LA ACCIÓN HAY QUE  
PROGRAMARLA, CLARO**

```
void button1_actionPerformed(ActionEvent e) {  
    textArea1.setText("Opción: "+checkboxGroup1.getCurrent().getLabel());  
}
```



```
void jButton1_actionPerformed(ActionEvent e) {
    jTextArea1.setText("");
}
void jButton2_actionPerformed(ActionEvent e) {
    jTextArea1.append("ESCOGIDO: Mañana\n");
}
void jButton3_actionPerformed(ActionEvent e) {
    jTextArea1.append("ESCOGIDO: Tarde\n");
}
void jButton4_actionPerformed(ActionEvent e) {
    jTextArea1.append("ESCOGIDO: "+jComboBox1.getSelectedItem()+"\n");
}
```





```
void jRadioButtons_actionPerformed(ActionEvent e) {  
    JTextArea1.append("ESCOGIDO: "+e.getActionCommand()+"\n");  
}
```

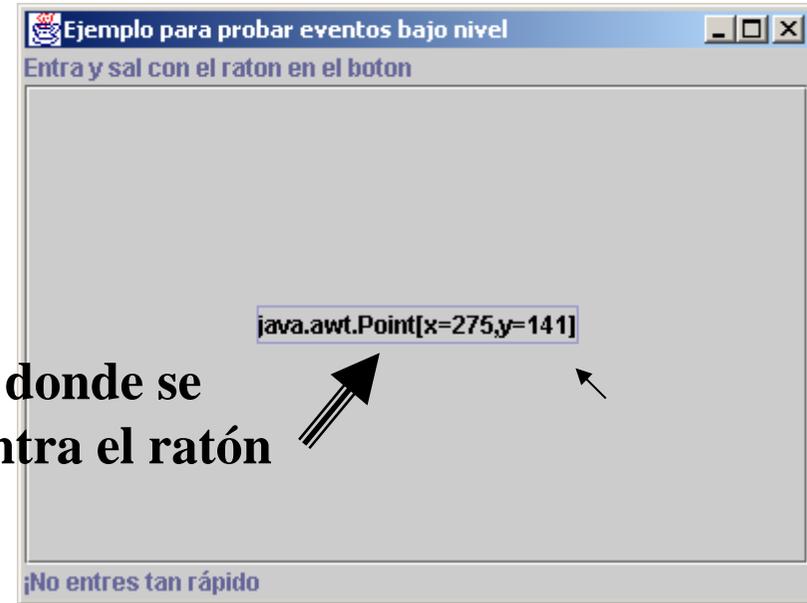
**Se puede usar un único método para tratar los eventos de los dos JRadioButton. El texto se puede obtener del objeto EVENTO (ActionEvent e)**

**Para obtener información de contexto del evento (como la etiqueta del componente gráfico sobre el que se ha producido el evento)**

Inicialmente...

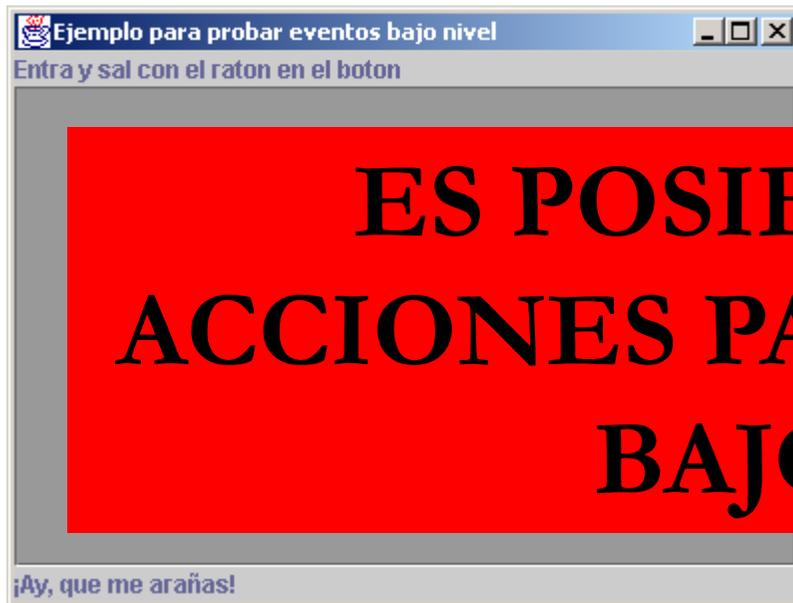


Cuando entra el ratón en el botón

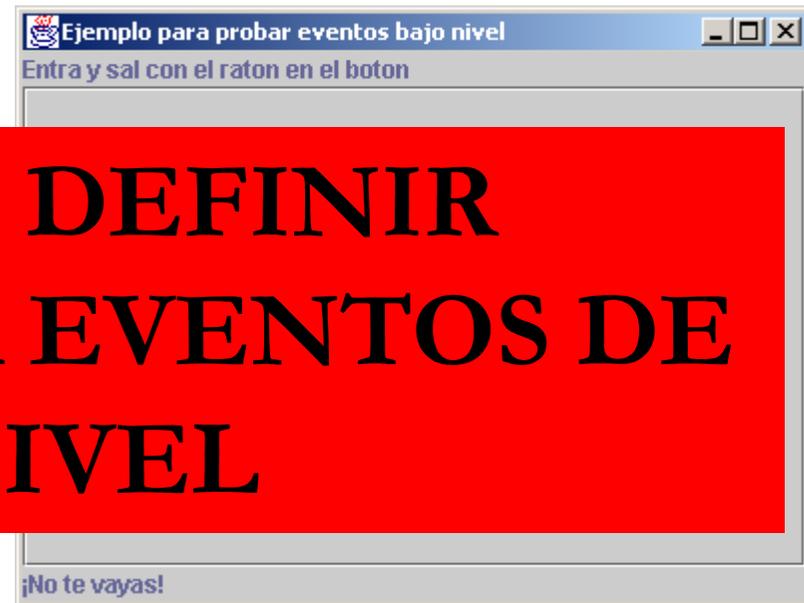


Punto donde se encuentra el ratón

Al hacer "drag" dentro del botón



Al salir el ratón del botón



**ES POSIBLE DEFINIR  
ACCIONES PARA EVENTOS DE  
BAJO NIVEL**

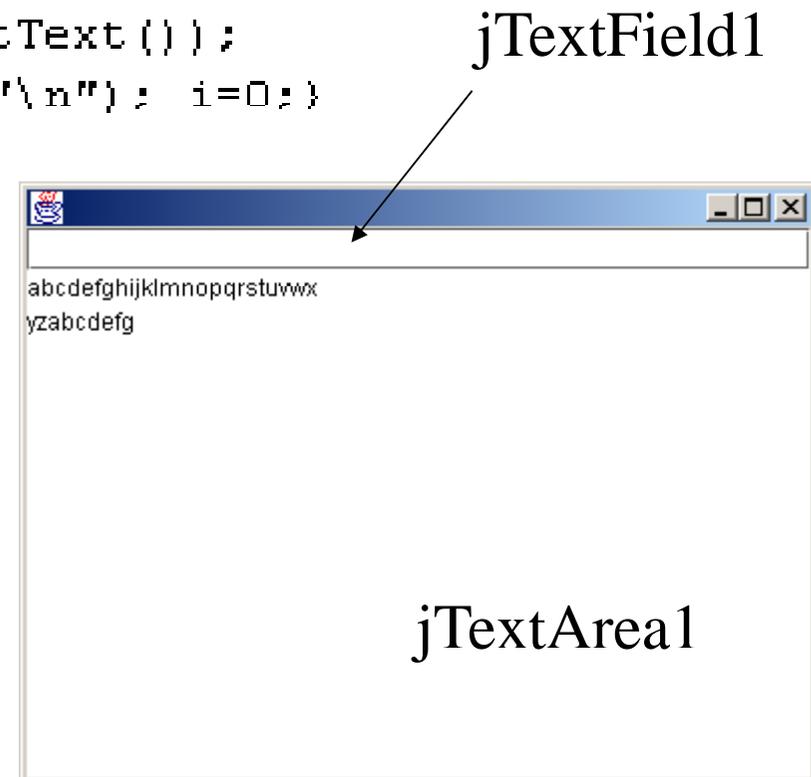
```
jButton1.addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    public void mouseMoved(MouseEvent e) {
        jButton1_mouseMoved(e);
    }
    public void mouseDragged(MouseEvent e) {
        jButton1_mouseDragged(e);
    }
});
jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseEntered(MouseEvent e) {
        jButton1_mouseEntered(e);
    }
    public void mouseExited(MouseEvent e) {
        jButton1_mouseExited(e);
    }
});

void jButton1_mouseEntered(MouseEvent e) {
    jLabel2.setText(";No entres tan rápido");
}
void jButton1_mouseExited(MouseEvent e) {
    jLabel2.setText(";No te vayas!");
    jButton1.setLabel("Entra, por favor");
}
void jButton1_mouseMoved(MouseEvent e) {
    jButton1.setLabel(e.getPoint().toString());
}
void jButton1_mouseDragged(MouseEvent e) {
    jLabel2.setText(";Ay, que me arañas!");
}
```

```
jTextField1.addKeyListener(new java.awt.event.KeyAdapter() {  
  
    public void keyPressed(KeyEvent e) {  
        jTextField1_keyPressed(e);  
    }  
});
```

```
void jTextField1_keyPressed(KeyEvent e) {  
    jTextArea1.append(jTextField1.getText());  
    if (i++==24) {jTextArea1.append("\n"); i=0;}  
    jTextField1.setText("");  
}
```

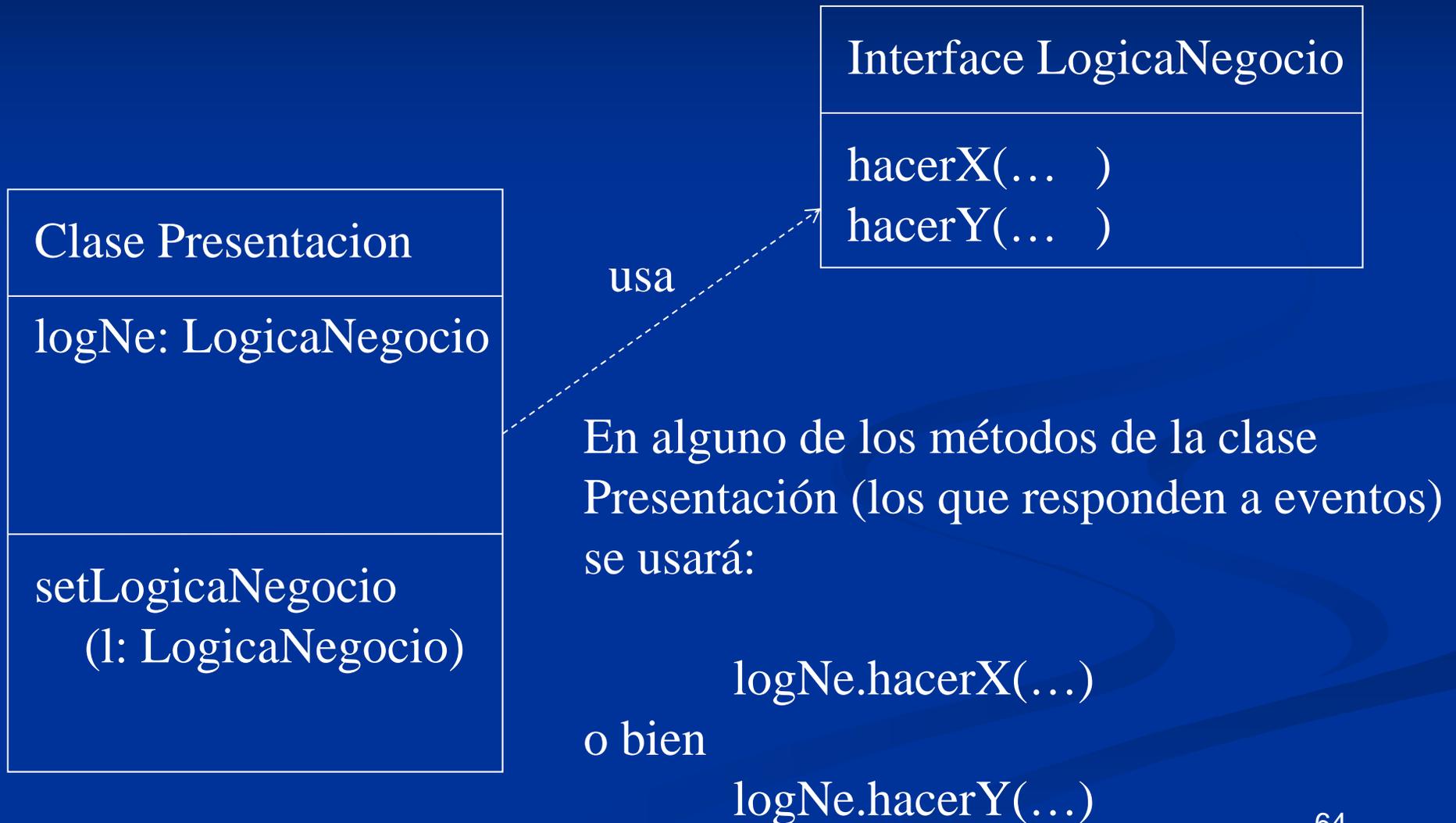
**Cada vez que se escribe un carácter en la caja de texto, se añade al área de texto y se borra de la caja de texto. Cada 25 caracteres se salta de línea en el área de texto.**

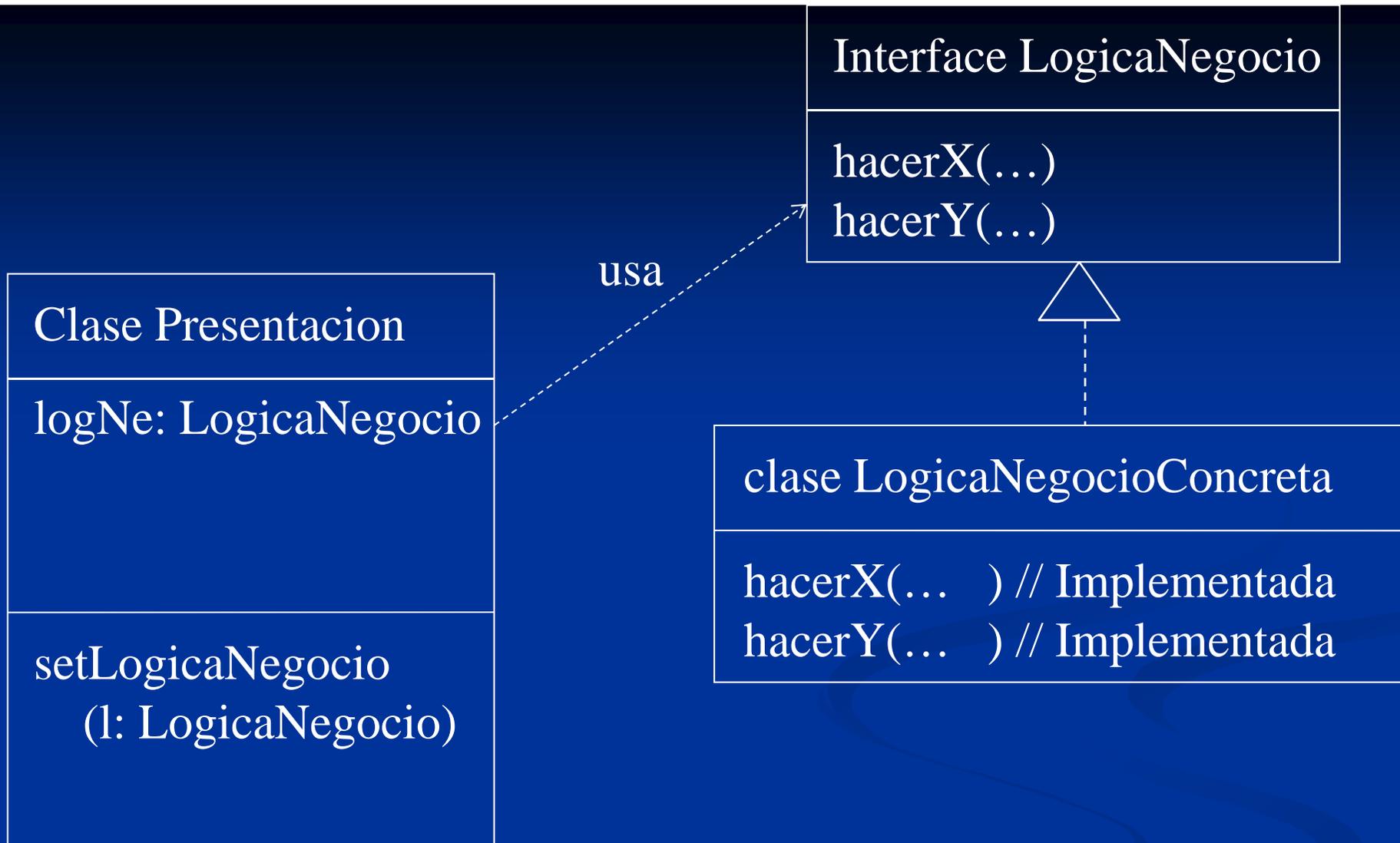


# Separación entre Nivel de Presentación y Lógica del Negocio

- Es conveniente separar el nivel de presentación del de la lógica del negocio
- El nivel de presentación está formado por las clases de AWT/Swing
- La llamada al nivel lógica del negocio se realizará en algún método de respuesta a un evento.
- Se puede incluir un atributo que contenga el objeto con la lógica del negocio (DE TIPO interface JAVA)
  - Podría conseguirse cambiar la lógica del negocio SIN NECESIDAD DE cambiar el nivel de presentación. Incluso sin RECOMPILAR, e incluso haciéndolo EN TIEMPO DE EJECUCIÓN (sin relanzar el objeto de presentación)

# Separación entre Nivel de Presentación y Lógica del Negocio

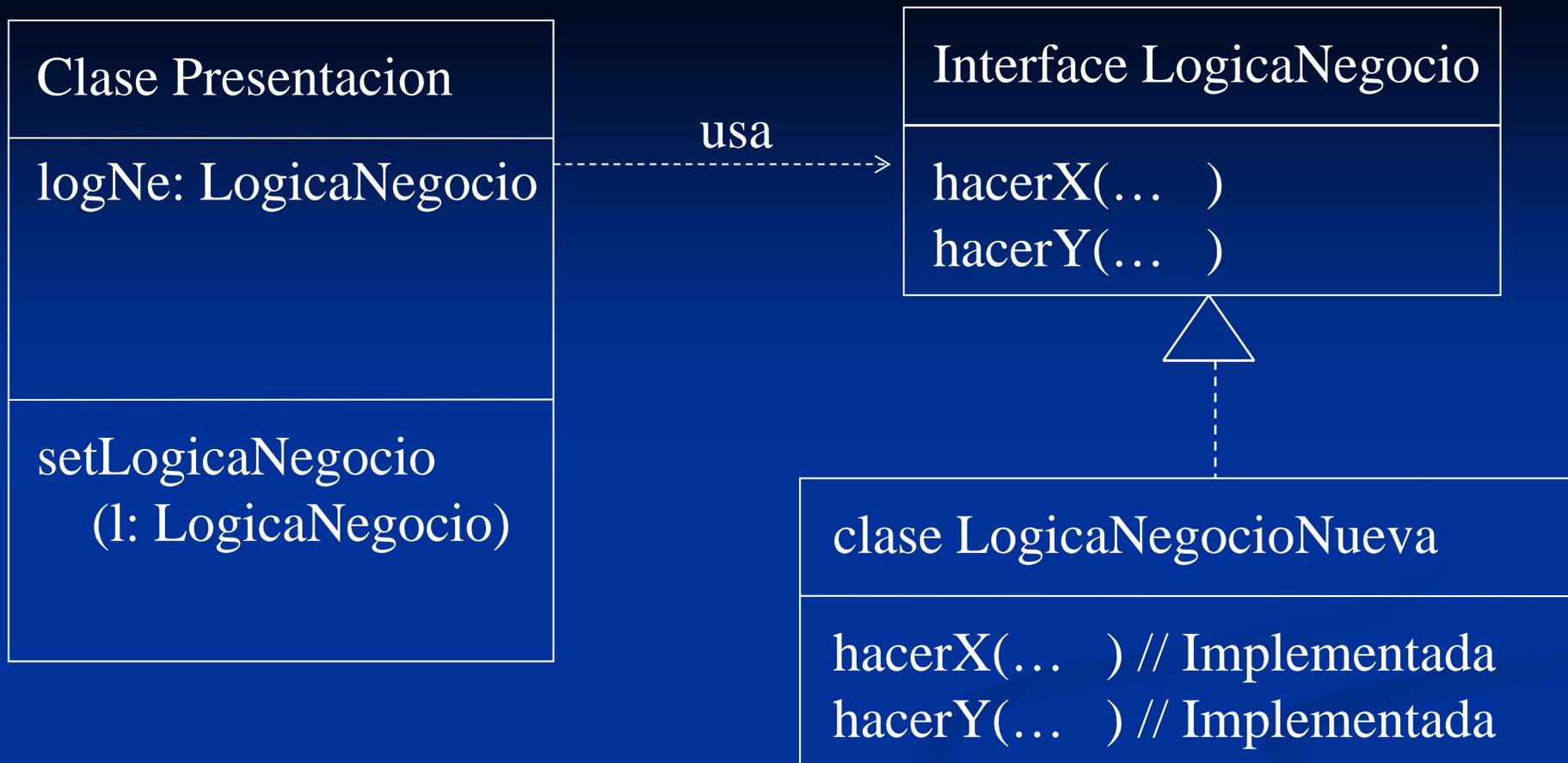




**PARA CREAR LA INTERFAZ GRÁFICA CON SU LÓGICA DEL NEGOCIO:**

```

Presentacion p = new Presentacion();
p.setLogicaNegocio(new LogicaNegocioConcreta());
p.setVisible(true);
  
```



Si ahora se quisiera cambiar la lógica del negocio, bastaría con hacer: `p.setLogicaNegocio(new LogicaNegocioNueva());`

**NO HACE FALTA RECOMPILAR LA CLASE Presentacion, y, si se conoce la referencia del objeto, SE PUEDE CAMBIAR LA LÓGICA DEL NEGOCIO EN TIEMPO DE EJECUCIÓN**

```

public interface GestorBilletes {
    /** Método para obtener un billete al que se le asocia un nombre
     * @param n Nombre que se asocia al billete
     * @return Referencia del billete (número natural).
     * Si es un número negativo, entonces no se ha podido obtener el billete.
     */
    int getBillete(String n);
}

import javax.swing.*;
import java.awt.*;
import ejsSwing.GestorBilletes;
import java.awt.event.*;

public class PedirBillete extends JFrame {
    JPanel jPanel1 = new JPanel();
    JLabel jLabel1 = new JLabel();
    JTextField jTextField1 = new JTextField();
    JButton jButton1 = new JButton();
    JTextArea jTextArea1 = new JTextArea();
    GestorBilletes gestorBilletes; // Objeto con la lógica del negocio

    public PedirBillete() {
        ...
    }

    public void setGestorBilletes(GestorBilletes g) {
        gestorBilletes=g;
    }

    void jButton1_actionPerformed(ActionEvent e) {
        int res = gestorBilletes.getBillete(jTextField1.getText());
        if (res<0) jTextArea1.append("Error al asignar billete");
        else jTextArea1.append("Asignado. \nReferencia: "+res+"\n");
    }
}

```



```
PedirBillete b = new PedirBillete();
b.setGestorBilletes(new GestorBilletes() {
    public int getBillete(String n) {
        try {return Integer.parseInt(n);}
        catch(Exception e) {return -1;}}});
b.setVisible(true);
```

Sólo falta proporcionar  
la clase que implemente  
GestorBilletes...  
(con o sin nombre)

```
PedirBillete b = new PedirBillete();
b.setGestorBilletes(new GestorBilletesImpl());
b.setVisible(true);
```

```
public class GestorBilletesImpl implements GestorBilletes
    public int getBillete(String n) {
        try {return Integer.parseInt(n);}
        catch(Exception e) {return -1;}}
```

