

Introducción Interfaces Gráficas de Usuario

Transparencias cedidas por los
profesores de Programación
Orientada a Objetos

Programación de Sistemas y
Concurrencia
Curso 20-21



Índice

- Construcción de GUIs en Java
 - El patrón MVC (Modelo – Vista – Controlador)
- Vistas
 - Contenedores y Componentes
 - Construcción
 - Gestor de Esquemas
 - Estudio de Componentes
- Controladores
 - Modelos de Eventos
 - Interfaces para Implementar Controladores

Los paquetes **java.awt** y **javax.swing**

- Permiten la construcción de Interfaces Gráficas de Usuario (GUIs).
- Swing se basa en (y extiende) AWT.
- Se verán las características más importantes de Swing para construir GUIs básicos.

AWT y Swing

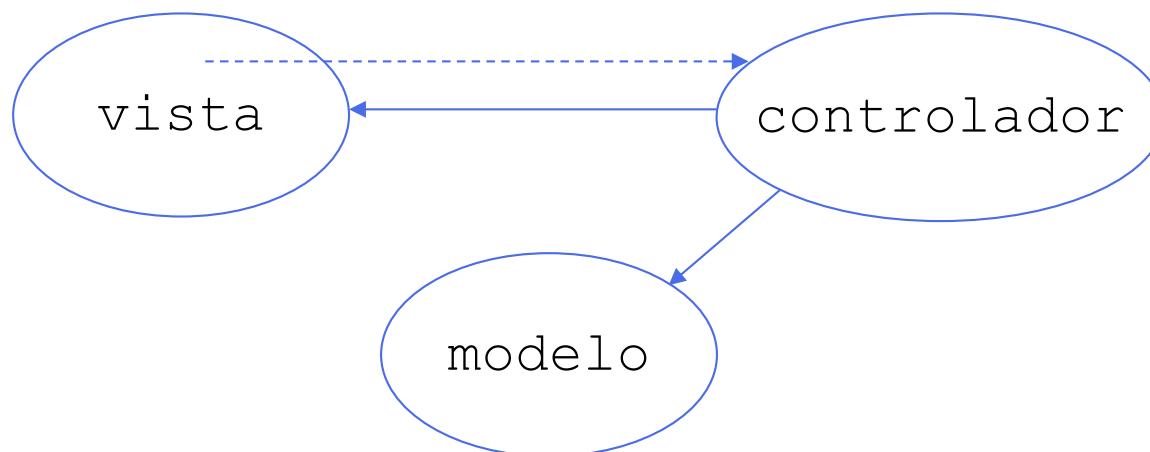
- Por cada componente visible de AWT existe otro en el sistema operativo que lo representa:
 - El resultado final dependerá de este componente.
- Problema

 - Hay facilidades que algún sistema operativo no tiene, por lo que AWT define lo que tienen en común todos.
 - Puede verse de forma diferente en dos sistemas operativos.

- Swing elimina este problema
 - Define todos los componentes usuales en GUIs.
 - Se encarga de mostrar cada componente.
 - Necesita los paquetes (y subpaquetes):
`java.awt, java.awt.event y javax.swing`

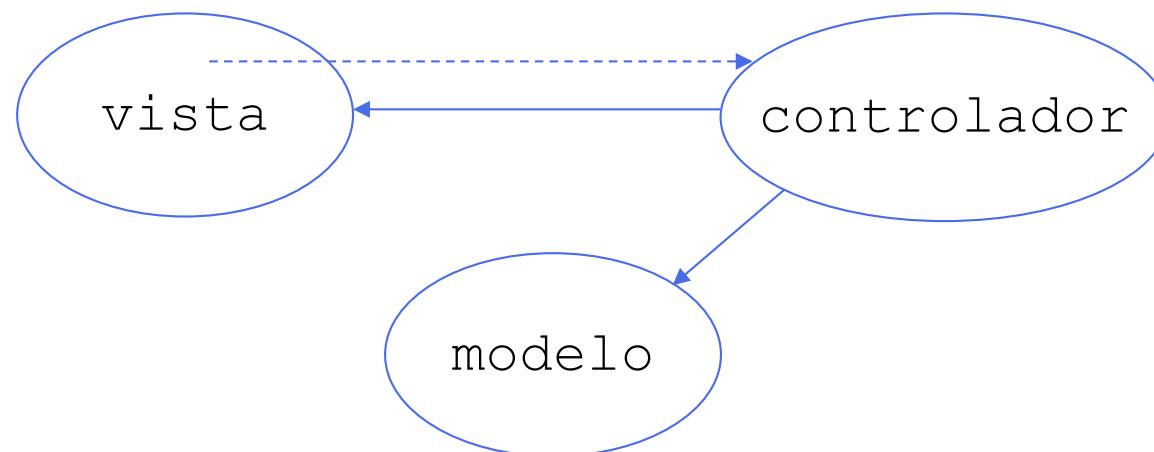
Diseño de interfaces de usuario

- Usaremos el patrón Modelo-Vista-Controlador (MVC)
 - Modelo: los datos a manipular por la aplicación.
 - Vista: la representación de la información.
 - Controlador: la lógica de la aplicación.
 - Está pendiente de las acciones del usuario sobre la vista.
 - Estas acciones provocan una reacción del controlador:
 - » Consultar/actualizar la vista y el modelo.
- Objetivo: Independizar los distintos componentes.



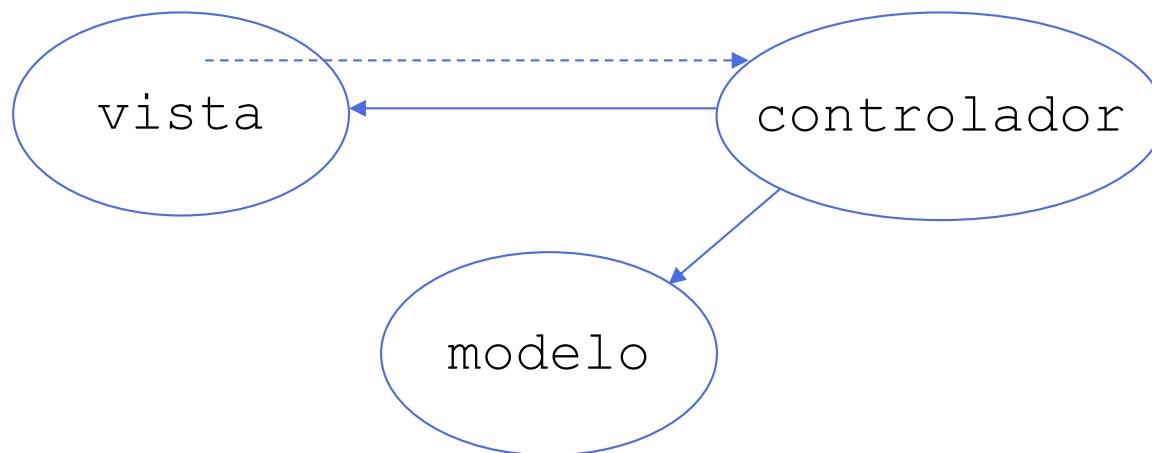
MVC: Modelo

- Información para la que se realiza la interfaz gráfica.
 - Puede ser desde una variable hasta una gran cantidad de objetos.
- Debe ser lo más independiente posible de la vista y del controlador.
 - El modelo existe, independientemente de la interfaz gráfica.



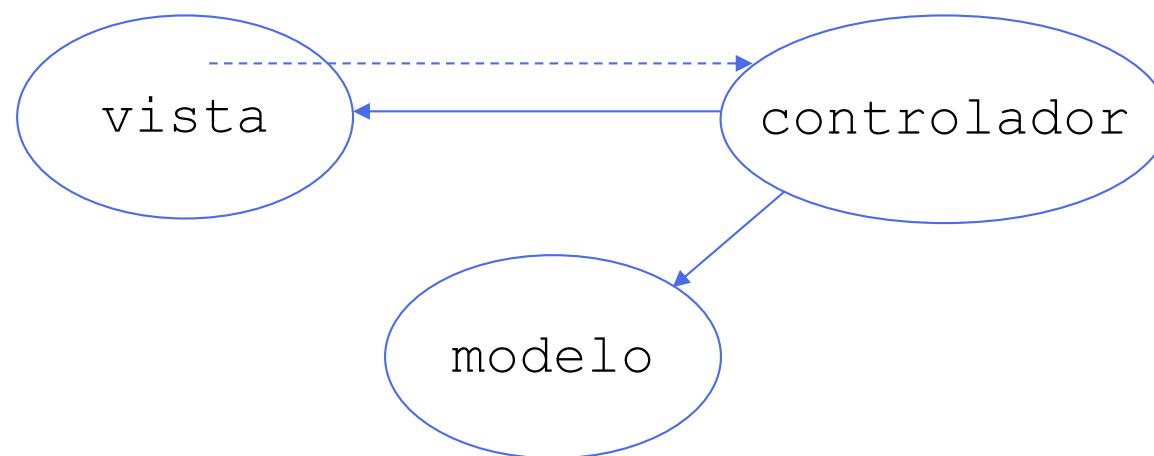
MVC: Vista

- Representación gráfica de la información.
 - Ventana que contiene botones, áreas editables de texto, etiquetas, listas desplegables, etc.
- Interactúa con el controlador.
 - En ciertas ocasiones, también con el modelo.
- Para un mismo modelo es posible generar varias vistas distintas.
 - Ej: Modelo: Carpetas y ficheros del sistema operativo
 - Vistas: Interfaz de comandos, Explor. Windows, Explor. Norton, etc.



MVC: Controlador

- La lógica de la aplicación.
- Es avisado cuando el usuario actúa sobre la vista.
 - Para ello, debe registrarse en ciertos elementos activos de la vista.
- En un buen diseño, varias vistas podrían disponer del mismo controlador.
- También es posible disponer de varios controladores especializados, cada uno controlando distintos eventos.

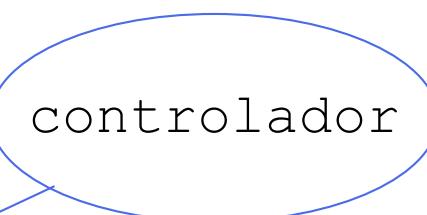


¿Cómo funciona el MVC?



Cuando se crea el controlador se le pasa una referencia a la vista y otra al modelo

Modelo
- acumulador
+ Modelo()
+ acumular(int)
+ resetear()
+ int obtenerAcumulador()



¿Cómo funciona el MVC?

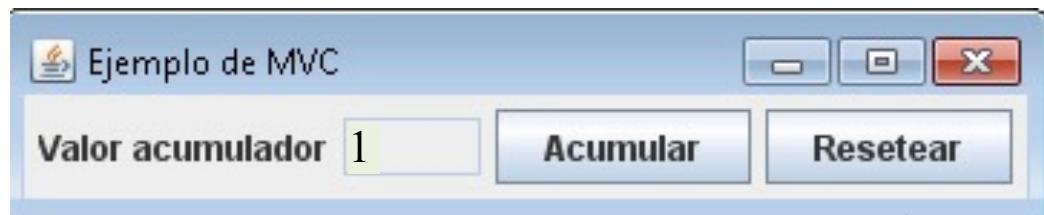


Después el controlador se registra en la vista para que le “avisen” cuando el usuario actúa sobre la vista

Modelo
- acumulador
+ Modelo()
+ acumular(int)
+ resetear()
+ int obtenerAcumulador()

controlador

¿Cómo funciona el MVC?



¿Qué ocurre cuando el usuario pulsa el botón “Acumular” de la vista?

Modelo
- acumulador
+ Modelo()
+ acumular(int)
+ resetear()
+ int obtenerAcumulador()

1. Se ha pulsado “Acumular”
2. Envía el mensaje: acumular(1)
3. Envía el mensaje: obtenerAcumulador()
4. Actualiza el cuadro de texto

controlador

Ejemplo MVC Paso a Paso

- Crearemos:
 - El modelo: Acumulador.
 - Clase **Modelo.java**
 - **Una interfaz Panel.java**
 - La vista: dos vistas distintas.
 - Clases **Panel1.java**, **Panel2.java**
 - El controlador:
 - Clase **Controlador.java**
 - Una aplicación **Principal.java** que crea el modelo, la vista y el controlador y establece las relaciones entre ellos.

Ejemplo MVC: La clase **Modelo**

- Utilizaremos un objeto de esta clase como **modelo**.
 - Permite manipular un número entero (**acumulador**).
 - Métodos para:
 - Incrementar el valor del **acumulador**
`void acumular(int)`
 - Poner a 0 el valor del **acumulador**
`void resetear()`
 - Consultar el valor del **acumulador**
`int obtenerAcumulador()`

El Modelo: La clase Cuenta

```
public class Modelo {  
    private int acum = 0;  
    public void acumular (int ac) {  
        acum+=ac;  
    }  
    public void resetear() {  
        acum = 0;  
    }  
  
    public int obtenerAcumulador() {  
        Return acum;  
    }  
}
```

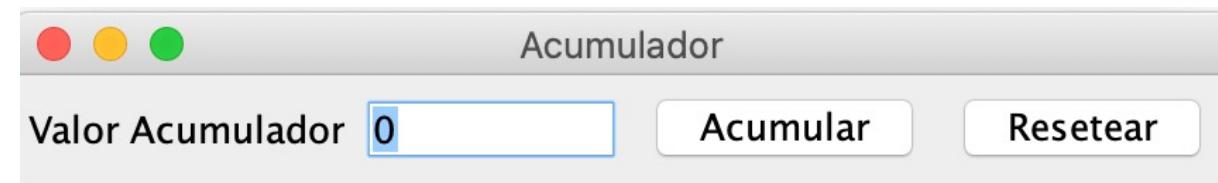
Dos vistas para el Modelo

Vistas

Panell

Controlador

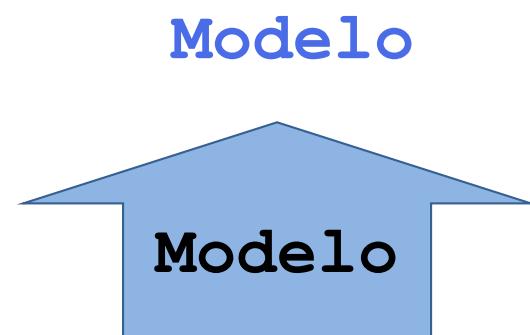
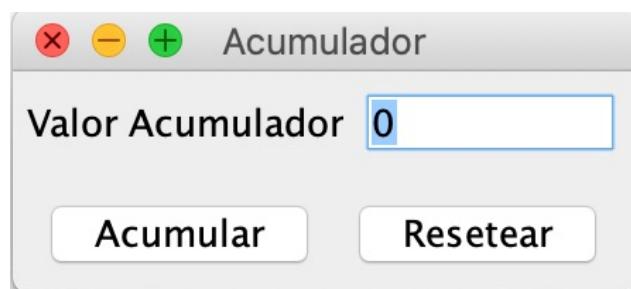
Controlador



Modelo

Modelo

Con GUI. Dos vistas para **Modelo**



Vistas: Panel1 , Panel2

- Definimos una interfaz para las vistas:

```
import java.awt.event.*;  
public interface Panel{  
    public void nuevoValor(int v);  
    public void controlador(ActionListener ctr);  
}
```

Lo vemos más adelante

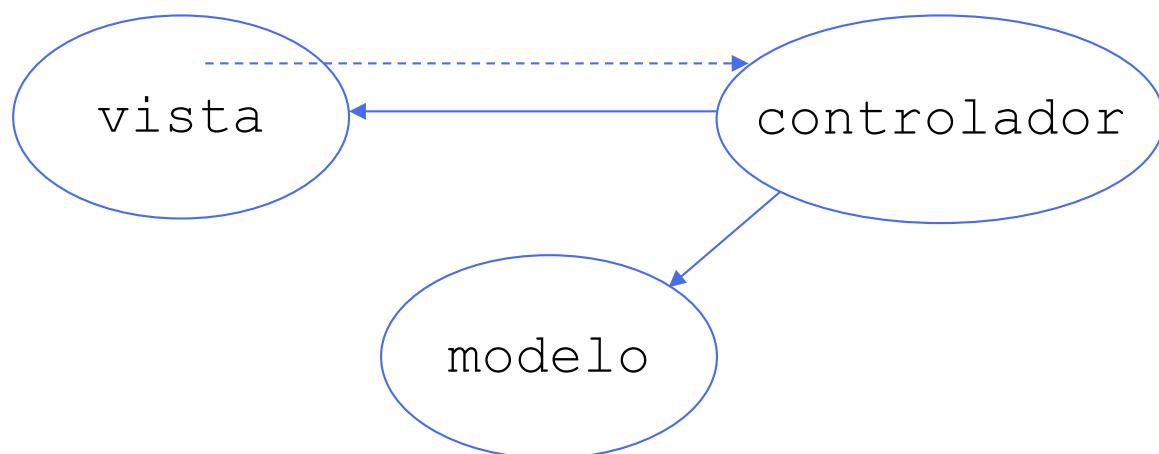
```
public Panel1 extends JPanel implements Panel{  
    // ctes. Comandos  
    // métodos para consultar/actualizar los elementos de la vista  
    public void nuevoValor(int v)  
    // método que registra el controlador en los elementos de la vista  
    void controlador(ActionListener ctr);  
}
```

- El método **controlador (ActionListener ctr)**
 - Registra el controlador **ctr** en los componentes adecuados.

Controlador: Controlador

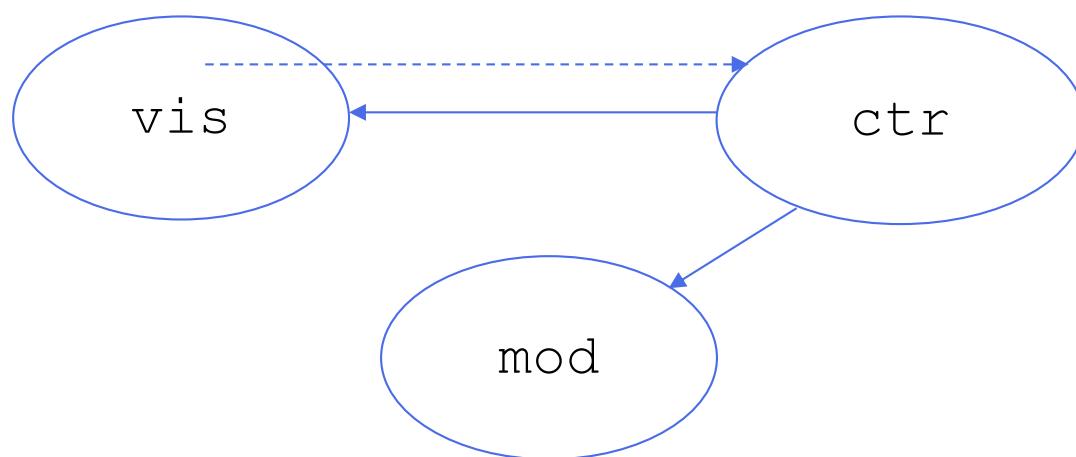
- Mantiene dos variables de instancia:
 - El modelo: **Modelo**
 - La vista : **Panel**

```
public Controlador(Panel panel, Modelo modelo ) {  
    this.panel = panel;      // La vista  
    this.modelo = modelo;     // El modelo  
}
```



Aplicación con MVC

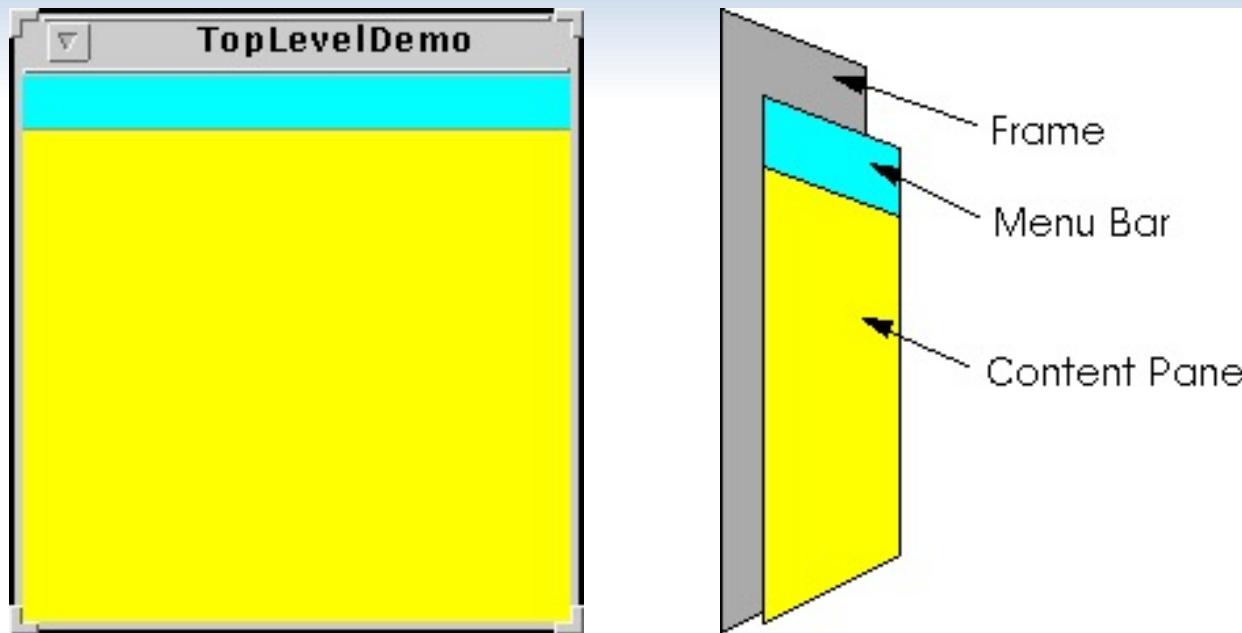
```
import javax.swing.*;  
public class Principal {  
    public static void main(String args[]) {  
  
        Modelo modelo = new Modelo();  
        Panel panel = new Panel1();  
        Controlador ctr = new Controlador(panel,modelo);  
        panel.controlador(ctr);  
    }  
}
```



Elementos de Swing

- Componentes y Contenedores.
 - Componentes. Aspecto visible de la interfaz:
 - Botones, etiquetas, campos de texto, etc.
 - Siempre se sitúan dentro de algún contenedor.
 - Contenedores. Almacenes de componentes:
 - Dos tipos:
 - Superiores: **JApplet**, **JFrame** y **JDialog**.
 - Intermedios: **JPanel**, **JScrollPane**, **JSplitPane**, **JTabbedPane**, **JToolBar** y otros más especializados.
 - Los superiores contienen a uno o varios intermedios.
 - Los intermedios contienen a los componentes y pueden contener otros contenedores intermedios.

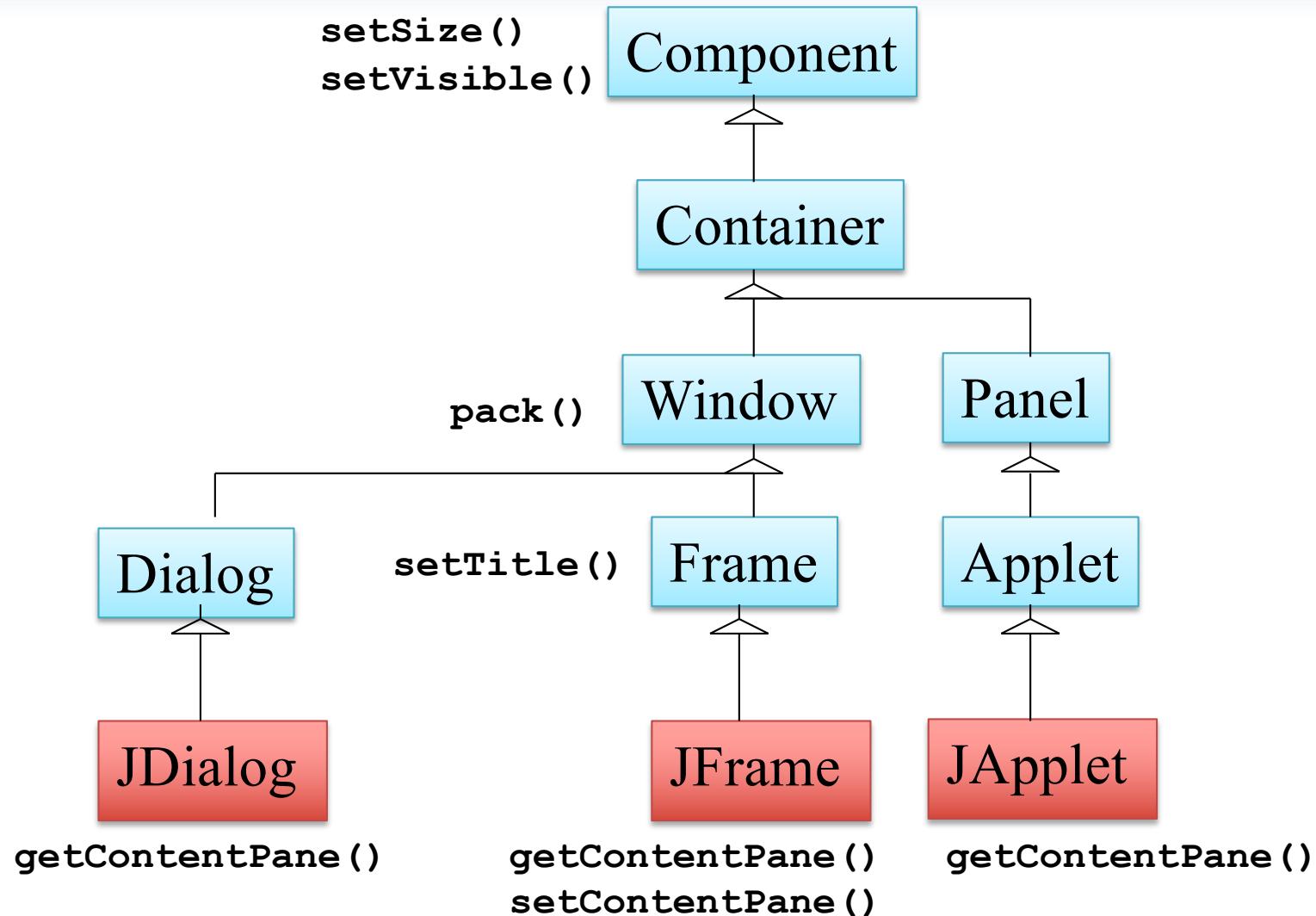
Contenedores superiores I



- Disponen de un panel de contenidos (contentPane) donde generalmente colocaremos la vista (este es el contenedor intermedio)
- Pueden opcionalmente disponer de un menú.

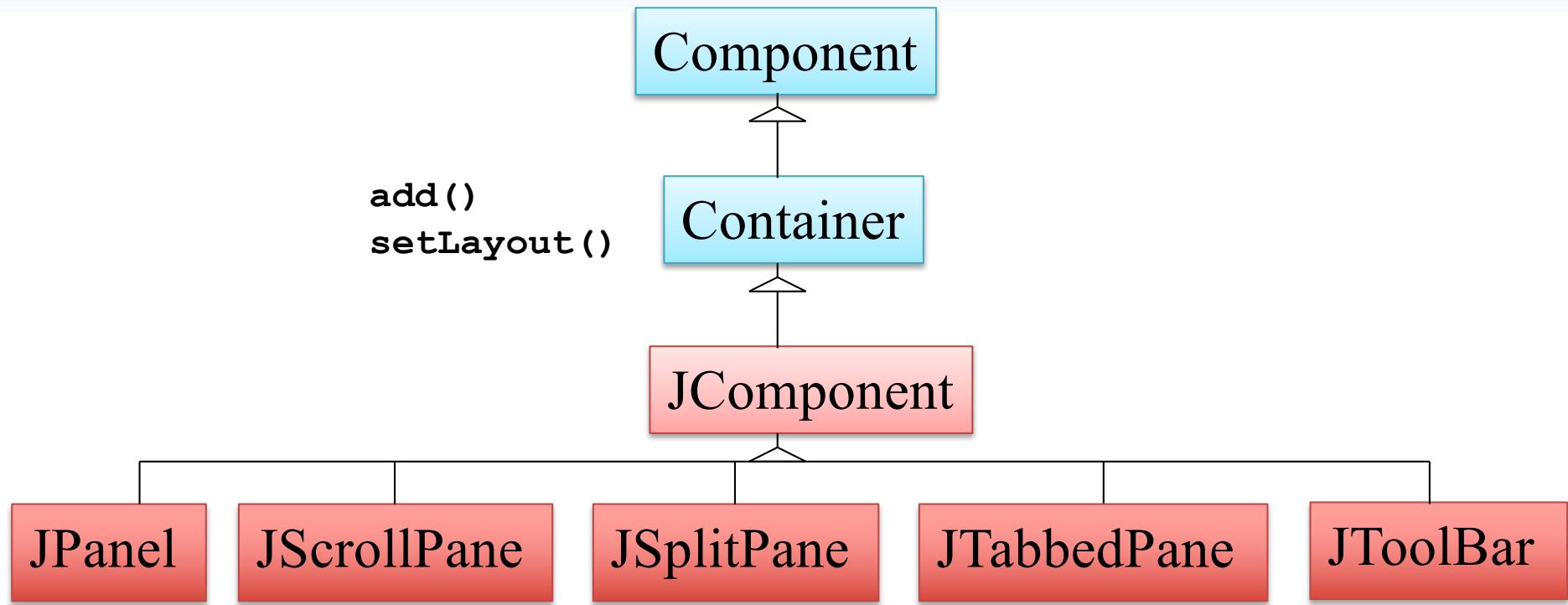
```
JFrame ventana = new JFrame("GUI simple");
Panel panel = new Panel1(); //los elementos visuales
ventana.setContentPane((Jpanel) panel);
```

Contenedores superiores II



AWT
Swing

Contenedores intermedios



- El contenedor más utilizado es **JPanel**.

Construcción de una GUI

Un posible esquema (hay otros):

1. Generar la vista heredando de un **contenedor intermedio**.
 1. Seleccionar un “gestor de esquemas” para dicho contenedor.
 2. Crear los componentes visuales.
 3. Agregarlos al contenedor intermedio.
2. Crear un objeto de una clase **contenedora superior** (**JFrame**, **JDialog**)
 1. Usar el objeto vista creado en el paso 1 como su panel de contenidos.
 2. Dimensionar el contenedor superior.
 3. Hacer visible el contenedor superior.

Construcción de una GUI



1. Generar la vista heredando de un contenedor intermedio.
 1. Seleccionar un gestor de esquemas para dicho contenedor.
 2. Crear los componentes visuales.
 3. Agregarlos al contenedor intermedio.
2. Crear un objeto de una clase contenedora superior (**JFrame**, **JDialog**)
 1. Usar el objeto vista creado en el paso 1 como su panel de contenidos.
 2. Dimensionar el contenedor superior.
 3. Hacer visible el contenedor superior.

1. Obtener un contenedor intermedio

- Hay varias formas: Por ejemplo, creamos una subclase de un contenedor intermedio.
 - El contenedor intermedio más simple es **JPanel**.

```
public class PanelVentana extends JPanel {  
    ...  
}
```

- Los contenedores intermedios disponen de métodos para cambiar el gestor de esquemas:

```
public void setLayout(AbstractLayout)
```

y otros para añadir componentes:

```
public void add(Component)
```

- En el constructor se genera la vista de nuestra interfaz gráfica.

1. Gestor de esquemas para contenedores intermedios

- Determinan cómo se distribuyen los componentes dentro de los contenedores.
 - Los gestores (clases) existentes son:
 - **FlowLayout, BorderLayout, GridLayout, GridBagConstraints, BoxLayout, ...**
 - Cada contenedor tiene un gestor propio:
 - Por defecto **JPanel** tiene **FlowLayout**.
 - El método **void setLayout(AbstractLayout)** permite cambiar de gestor.
- Ejemplo: para asignar un gestor de esquemas **FlowLayout**:
setLayout(new FlowLayout())
- La asignación del gestor de esquemas suele hacerse en el constructor del panel.

1. Crear componentes

- Cada componente viene determinado por una clase. Hay que crear un objeto de esa clase:

```
JButton bSí = new JButton("SÍ");
JButton bNo = new JButton("NO");
JLabel l    = new JLabel(" Es verdad?");

...
```

- Algunos componentes:
`JButton, JLabel, JTextField, JTextArea, JCheckBox,
JRadioButton, JList, JComboBox, JSlider,` etc.
- La creación de los componentes suele hacerse en el constructor del panel.
- La definición de los componentes puede realizarse:
 - como variable de instancia del panel y es visible en todo el panel.
 - como variable local del constructor del panel (no es visible en el panel).

1. Agregar componentes al contenedor

- Se realiza a través del método **void add(Component)**:

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

public class Panel extends JPanel {
    private JButton bSí;// Componentes declarados como variables de instancia
    private JButton bNo;

    public PanelVentana() {
        setLayout(new FlowLayout()); // Gestor de esquemas
        bSí = new JButton("SÍ");           // Creación de componentes
        bNo = new JButton("NO");
        JLabel l = new JLabel(" Es verdad?");
                           //Componente declarado como vble local del constructor
        add(l);                  // Se agregan al panel. El orden es importante
        add(bSí);
        add(bNo);
    }
}
```

- A un contenedor intermedio también se le pueden agregar otros contenedores intermedios.

2. Crear un contenedor superior (JFrame)

- Hay tres clases de contenedores superiores:
 - **JFrame** -> Aplicaciones.
 - Ventana de nivel superior con borde y título.
 - `setTitle(String)`, `getTitle()`, `setIconImage(Image)`.
 - **JApplet** -> Applets. Aplicaciones que corren en un navegador.
 - **JDialog** -> Diálogos. Tienen otro contenedor superior del que dependen.
- Disponen de un panel de contenidos para la representación del GUI.
 - Ese panel de contenidos es un contenedor intermedio que puede cambiarse.
- Métodos de instancia ...

```
void pack()
Container getContentPane()      // Obtiene el panel de contenidos
void setContentPane(Container)   // Cambia el panel de contenidos
void setJMenuBar(Menu)          // Coloca un menú
void setDefaultCloseOperation(int) // Para cerrar la ventana
                                JFrame.EXIT_ON_CLOSE // cierra y termina
...
...
```

2. Hacer del contenedor intermedio su panel de contenidos

- Todo contenedor superior dispone del método:

```
void setContentPane(Container)
```

para cambiar el panel de contenidos.

- Por ejemplo:

```
JFrame ventana = new JFrame("Un ejemplo");  
ventana.setContentPane(new Panel());
```

- También disponen del método:

```
Container getContentPane()
```

para obtener su panel de contenidos por defecto.

- Esto proporciona otra alternativa para crear GUIs

2. Dimensionar el contenedor superior

- Se puede especificar el tamaño del contenedor superior.
`void setSize(int anchura, int altura)`
- En lugar de `setSize()` es preferible utilizar el método `pack()`, que calcula el tamaño de la ventana teniendo en cuenta:
 - El gestor de esquemas.
 - El número y orden de los componentes añadidos.
 - La dimensión (preferida) de los componentes:
 - `void setPreferredSize(Dimension)`
 - `void setMinimumSize(Dimension)`
 - `void setMaximumSize(Dimension)`
- Ésta es la forma recomendada para ajustar el tamaño.

```
JFrame ventana = new JFrame("Un ejemplo");
...
ventana.pack();
```

2. Mostrar el contenedor superior

- Para hacerlo visible o invisible se utiliza el método: `setVisible(boolean)`
- Este método también es válido para mostrar u ocultar componentes y contenedores.

```
JFrame ventana = new JFrame("Un ejemplo");  
...  
ventana.setVisible(true);
```

Ejemplo completo

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

public class PanelVentana extends JPanel {
    private JButton bSí;
    private JButton bNo;
    private JLabel l;

    public PanelVentana() {
        setLayout(new FlowLayout());
        bSí = new JButton("SÍ");
        bNo = new JButton("NO");
        l = new JLabel("¿Verdad?");
        add(l);
        add(bSí);
        add(bNo);
    }
}

class Principal {
    public static void main(String[] args) {
        JFrame ventana = new JFrame("Un ejemplo"); // Creamos el contenedor superior
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setContentPane(new PanelVentana()); // Cambiamos el panel de contenidos
        ventana.pack(); // Empaquetamos
        ventana.setVisible(true); // La hacemos visible
    }
}
```



Panel y Principal

- Sólo las funciones de maximizar y minimizar, cambiar tamaño y mover están operativas.
- Los botones SÍ y NO ceden cuando se pulsan pero no realizan ninguna acción.
- La ventana se cierra normalmente:

```
ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

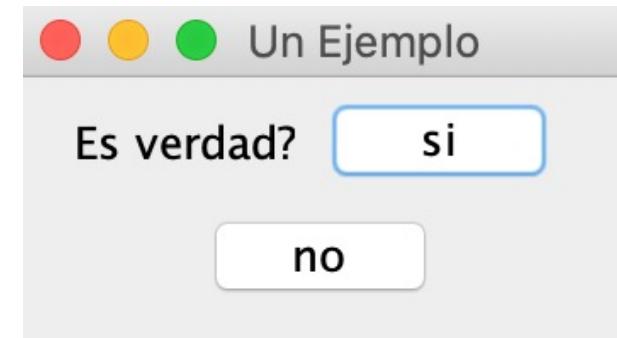


Gestores de esquemas

- Clases que determinan cómo se distribuirán los componentes dentro de un contenedor intermedio.
- La mayoría están definidas en **java.awt**
 - **FlowLayout**
 - **BorderLayout**
 - **GridLayout**
 - **GridBagLayout**
 - **BoxLayout**
 - ...
- **JPanel** por defecto dispone de un **FlowLayout**.

FlowLayout

- Los componentes fluyen de izquierda a derecha y de arriba abajo.
- Su tamaño se ajusta al texto que presentan.
- Al cambiar el tamaño de la ventana, puede cambiar la disposición.



BorderLayout

- Divide el contenedor en 5 partes:
 - NORTH, SOUTH, EAST, WEST y CENTER.
 - Los componentes se ajustan hasta llenar completamente cada parte.
 - Si algún componente falta, se ajusta con el resto (menos el centro si hay cruzados).
 - Para añadir al contenedor se utiliza una versión de **add** que indica la zona en la que se añade (constantes definidas en la clase).
`add(bSí, BorderLayout.NORTH)`

BorderLayout



Sin Norte ni Este



GridLayout

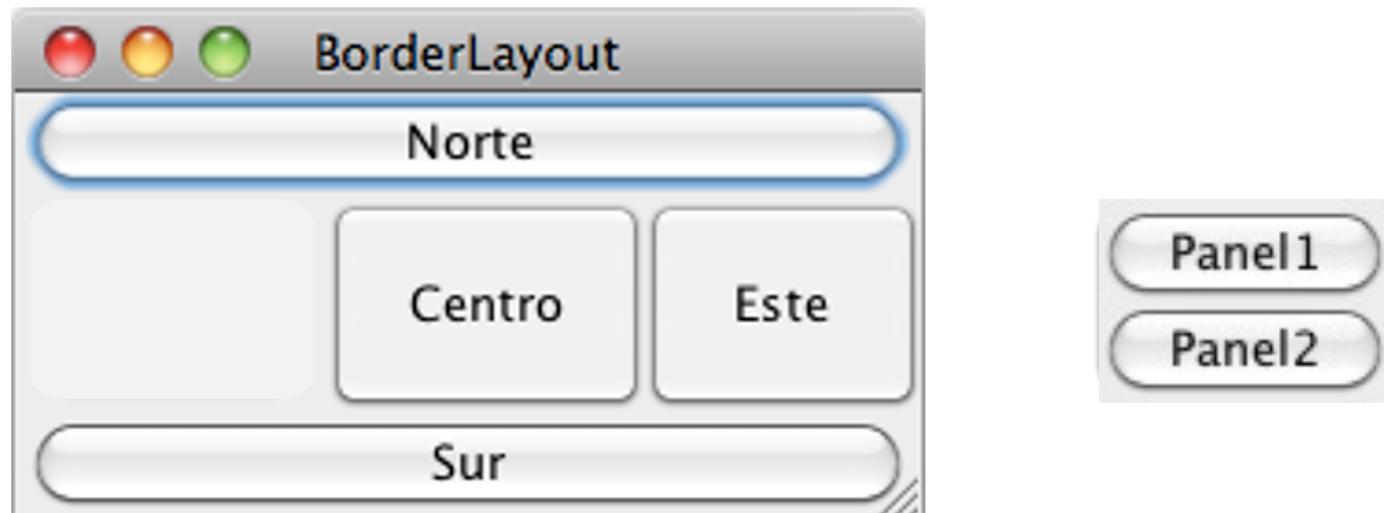
- Divide al componente en una rejilla (*grid*).
- En el constructor debemos indicar el número de filas y de columnas.
- Los componentes se mantienen de igual tamaño dentro de cada celda.
- El orden a la hora de agregar determina la posición (de izquierda a derecha y de arriba a abajo).

```
cpane.setLayout(new GridLayout(2, 3))
```

Dos filas y tres columnas.

GUI complejos I

- Podemos utilizar un contenedor intermedio en lugar de un componente para agregarlo a otro contenedor intermedio.
- Este nuevo contenedor intermedio podrá:
 - Incorporar sus propios componentes.
 - Tener su propio gestor de esquemas.



GUI Complejos II

```
setLayout(new BorderLayout());  
  
JPanel p = new JPanel();  
JButton bp1 = new JButton("Panel1");  
JButton bp2 = new JButton("Panel2");  
  
p.setLayout(new GridLayout(2, 1));  
p.add(bp1);  
p.add(bp2);  
...  
add(p, BorderLayout.WEST);  
...
```



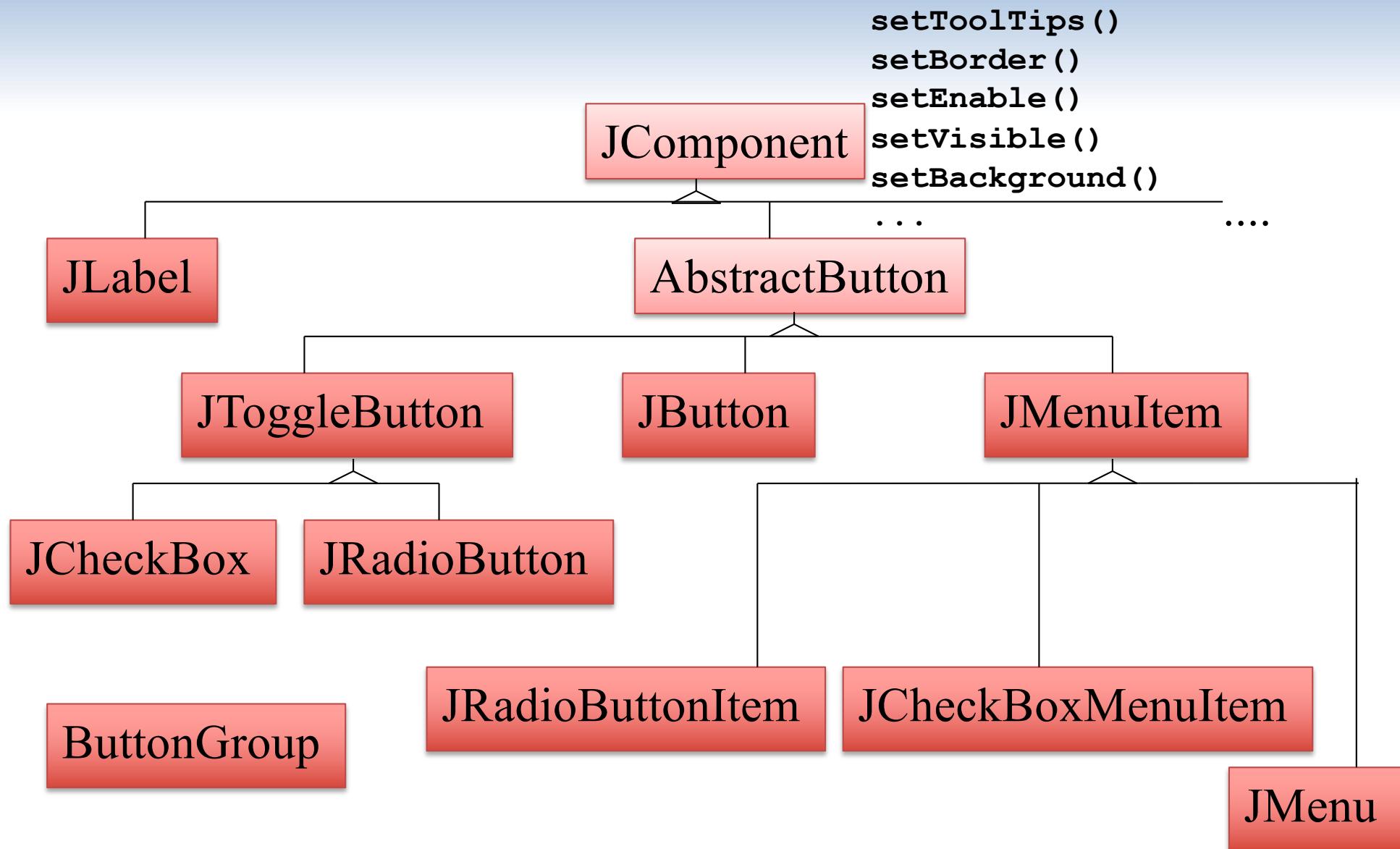
JScrollPane

```
import java.awt.*;
import javax.swing.*;
public class Principal {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        JFrame ventana = new JFrame("Ejemplo Scroll");
        Panel panel = new Panel();
        Controlador ctr = new Controlador(panel);
        panel.controlador(ctr);
        ventana.setContentPane(panel);
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.pack();
        ventana.setVisible(true);
    }
}

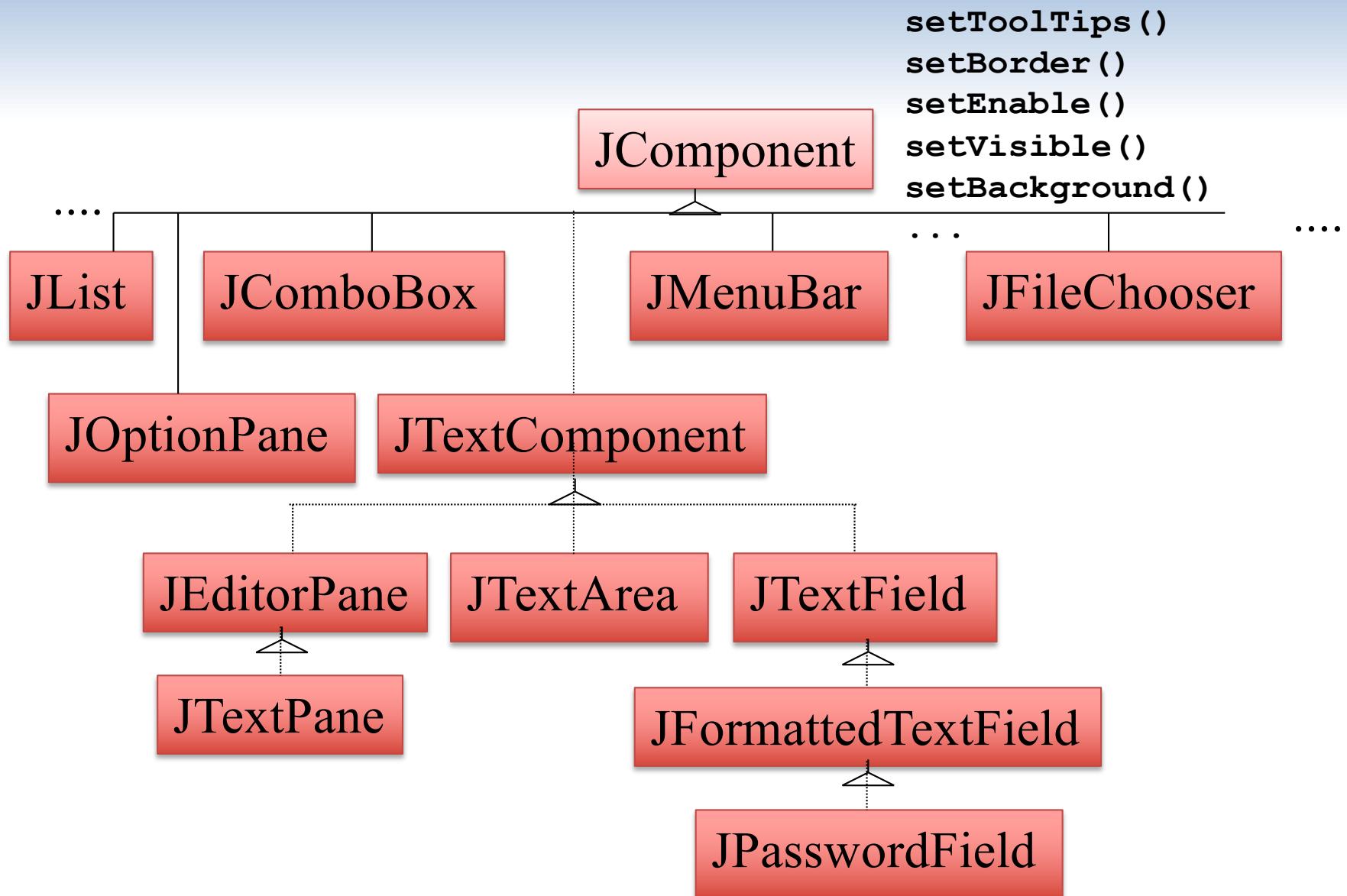
private JTextArea area = new JTextArea(20,20);
private JScrollPane scrol = new JScrollPane(area);
```



Componentes I



Componentes II



JButton

- Crea botones que ceden ante una pulsación.
- Constructores:

```
JButton()  
JButton(String)           // Puede ser HTML  
JButton(String, Icon)  
JButton(Icon)
```

- Métodos:

```
String getText()  
void setText(String) // Puede ser HTML  
...
```

JLabel

- Es una etiqueta con una línea de texto o gráfico.

- Constructores:

```
JLabel([String,] [Icon,] [int]) // Puede ser HTML
```

- Constantes desde la interfaz **javax.swing.SwingConstants** (3^{er} arg.):

LEFT	RIGHT	CENTER
-------------	--------------	---------------

- Métodos de instancia:

```
String getText()
```

```
void setText(String) // Puede ser HTML
```

```
...
```

JCheckBox

- Marcadores que pueden activarse o desactivarse con una pulsación.

- Constructores:

```
JCheckBox([String,] [Icon,] [boolean]) // Puede ser HTML
```

- Métodos de instancia:

```
String getText()
```

```
void setText(String) // Puede ser HTML
```

```
boolean isSelected()
```

```
void setSelected(boolean)
```

```
...
```

JRadioButtons y ButtonGroup

- Botones circulares (utilizados para selección alternativa).
- Se agrupan de manera que sólo uno esté pulsado.
- Constructores:

```
JRadioButtons([String,] [Icon,] [boolean]) // Puede ser HTML
```

- Métodos de instancia:
Igual que **JCheckBox**.
- Para agruparlos, se crea una instancia de **ButtonGroup** y se añaden con **add(AbstractButton)** .



Ejemplo con botones I

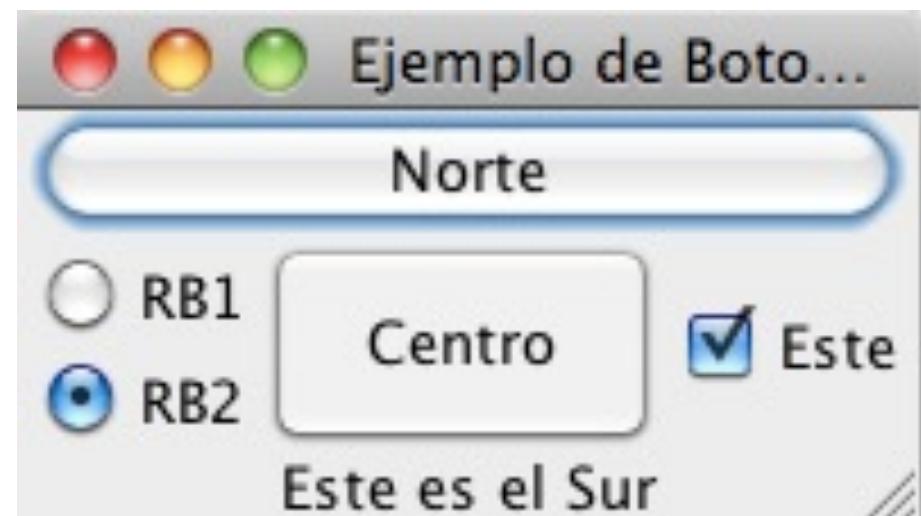
```
import java.awt.*;
import javax.swing.*;

public class PanelBotones extends JPanel {
    public PanelBotones() {
        JButton bNorte = new JButton("Norte");
        JLabel lSur = new JLabel("Este es el Sur", JLabel.CENTER);
        JCheckBox cEste = new JCheckBox("Este", true);
        JButton bCentro = new JButton("Centro");
        JRadioButton cp1 = new JRadioButton("RB1");
        JRadioButton cp2 = new JRadioButton("RB2", true);

        ButtonGroup gcb = new ButtonGroup();
        gcb.add(cp1);
        gcb.add(cp2);

        JPanel prb = new JPanel();
        prb.setLayout(new GridLayout(2, 1));
        prb.add(cp1);
        prb.add(cp2);

        setLayout(new BorderLayout());
        add(bNorte, BorderLayout.NORTH);
        add(lSur, BorderLayout.SOUTH);
        add(cEste, BorderLayout.EAST);
        add(prb, BorderLayout.WEST);
        add(bCentro, BorderLayout.CENTER);
    }
}
```



Ejemplo con botones II

```
...  
class PanelBotonesDemo {  
    public static void main(String[] args) {  
        JFrame ventana = new JFrame("Ejemplo de Botones");  
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        ventana.getContentPane(new PanelBotones());  
        ventana.pack();  
        ventana.setVisible(true);  
    }  
}
```



JTextField

- Permite editar texto en una línea.
 - Constructores:
`JTextField([String,] [int])`
 - Métodos de instancia:
`String getText()`
`String getText(int, int) // offset y len`
`void setEditable(boolean)`
`boolean isEditable()`
...
 - Existe una subclase que enmascara el eco (* u otro símbolo),
`JPasswordField`
con un método de instancia:
`char [] getPassword()`

JTextArea

- Permite editar texto en un área.

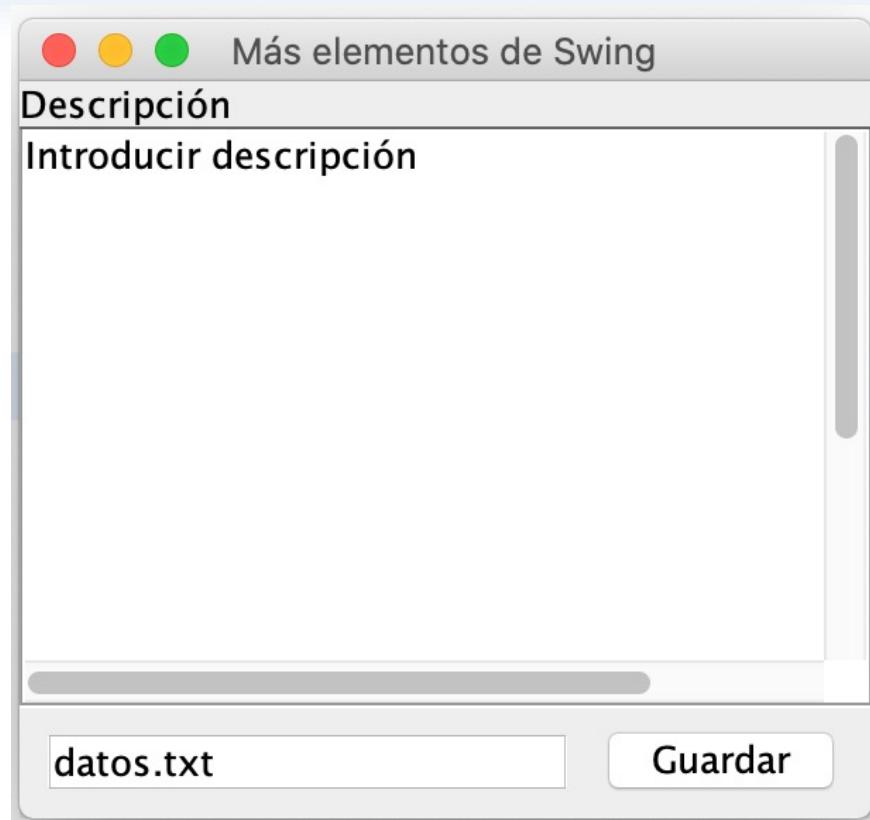
- Constructores:

```
JTextArea([String,] [int, int])  
//1er parámetro: texto a mostrar  
//2º y 3er parámetros: alto y ancho del área
```

- Métodos de instancia:

```
void append(String)  
void insert(String, int)  
igual que JTextField.  
...  
void setText(String);  
...
```

Ejemplo completo



Controladores: El modelo de eventos

- Un componente (o menú componente) puede disparar un evento

`java.awt.event`

`javax.swing.event`

- Cuando un evento se dispara, es recogido por objetos “controladores” u “oyentes” (*listeners*) que realizan la acción apropiada.
- Cada controlador debe pertenecer a una clase que implemente cierta interfaz dependiendo del evento.
- Ejemplo:
 - Evento: `ActionEvent`
 - Interfaz: `ActionListener`

El modelo de eventos

- Para que un controlador esté pendiente de un componente, se debe registrar en él.
- El registro es realizado a través de un método del componente sobre el que se registra:

addXXXXListener (XXXXListener)

- El receptor es el componente al que queremos oír.
 - El argumento será el objeto controlador.
 - **XXXXListener** indica la interfaz que va a implementar.
- Por ejemplo, dada la interfaz **ActionListener**, un objeto **ctr** que implementa la interfaz **ActionListener** se registra por medio de:

addActionListener (ctr)

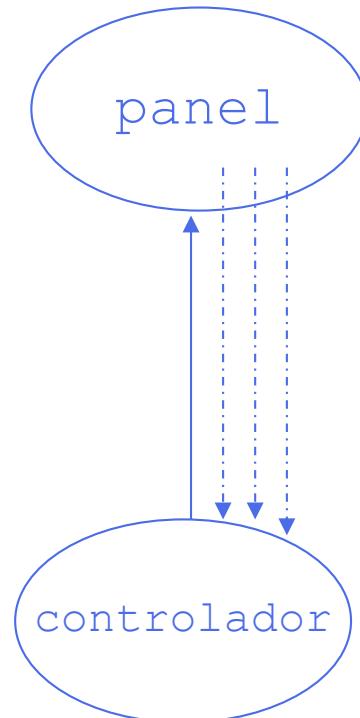
Relaciones Vista-Controlador

```
// Se crea la vista, es decir, el  
// panel que contiene la GUI  
MiPanel pan = new MiPanel();
```

```
// El controlador conoce la vista  
MiControlador crt = new MiControlador(pan);
```

```
// La vista debe disponer de un método  
// para asignar el controlador.  
// Aquí, ctr se debe registrar en cada  
// objeto que desee controlar  
pan.controlador(crt);
```

```
// Si tenemos varios oyentes no será  
// suficiente con un único método controlador
```



Interfaces en `java.awt.event` I

Interfaces para controladores

ActionListener	<code>actionPerformed(ActionEvent)</code>
AdjustmentListener	<code>adjustmentValueChanged(AdjustementEvent)</code>
ComponentListener	<code>componentHidden(ComponentEvent)</code> <code>componentMoved(ComponentEvent)</code> <code>componentResized(ComponentEvent)</code> <code>componentShown(ComponentEvent)</code>
ContainerListener	<code>componentAdded(ContainerEvent)</code> <code>componentRemoved(ContainerEvent)</code>
FocusListener	<code>focusGained(FocusEvent)</code> <code>focusLost(FocusEvent)</code>
ItemListener	<code>itemStateChanged(ItemEvent)</code>
KeyListener	<code>keyPressed(KeyEvent)</code> <code>keyReleased(KeyEvent)</code> <code>keyTyped(KeyEvent)</code>

Interfaces en `java.awt.event` II

Interfaces para controladores

MouseListener

`mouseClicked(MouseEvent)`
`mouseEntered(MouseEvent)`
`mouseExited(MouseEvent)`
`mousePressed(MouseEvent)`
`mouseReleased(MouseEvent)`

MouseMotionListener

`mouseDragged(MouseEvent)`
`mouseMoved(MouseEvent)`

TextListener

`textValueChanged(TextEvent)`

WindowListener

`windowActivated(WindowEvent)`
`windowClosed(WindowEvent)`
`windowClosing(WindowEvent)`
`windowDeactivated(WindowEvent)`
`windowDeiconified(WindowEvent)`
`windowIconified(WindowEvent)`
`windowOpened(WindowEvent)`

ActionListener I

- Se lanza si:
 - Se pulsa un botón de cualquier tipo.
 - Se selecciona una opción de menú.
 - Se pulsa retorno de carro en un campo de texto.

```
void actionPerformed(ActionEvent)
```

- Si un controlador está pendiente de varios objetos, puede:

- Preguntar quién lo ha activado
 - `Object getSource()`

- Consultar sobre una acción
 - `String getActionCommand()`

previamente indicada junto al registro

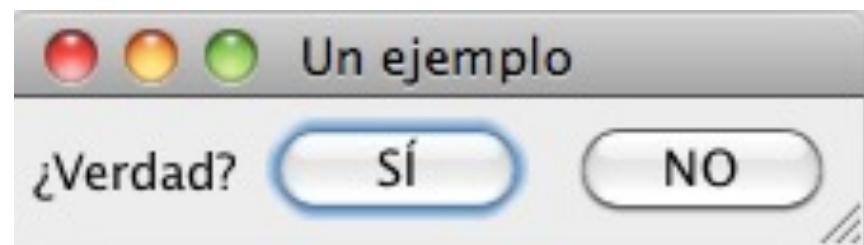
- `setActionCommand(String)`

Un ActionListener para PanelVentana

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

public interface PanelVentanaCtrExterno {
    public static final String SI = "SI";
    public static final String NO = "NO";

    public void controlador(ActionListener ctr);
    public void cambiaTexto(String s);
}
```



Un ActionListener para PanelVentana

```
import java.awt.*; import java.awt.event.*; import javax.swing.*;
public class PanelVentanaCtrExternoPanel extends JPanel implements PanelVentanaCtrExterno {
    private JButton bSi, bNo;
    private JLabel l;

    public PanelVentanaCtrExternoPanel() {
        setLayout(new FlowLayout());
        bSi = new JButton("SÍ");
        bNo = new JButton("NO");
        l = new JLabel("¿Verdad?");
        add(l);
        add(bSi);
        add(bNo);
    }

    public void controlador(ActionListener ctr) {
        bSi.addActionListener(ctr);
        bSi.setActionCommand(SI);
        bNo.addActionListener(ctr);
        bNo.setActionCommand(NO);
    }

    public void cambiaTexto(String s) {
        l.setText(s);
    }
}
```



El controlador

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

public class PanelVentanaCtrExternoCtr implements ActionListener {
    PanelVentanaCtrExterno ven;

    public PanelVentanaCtrExternoCtr(PanelVentanaCtrExterno v) {
        ven = v;
    }

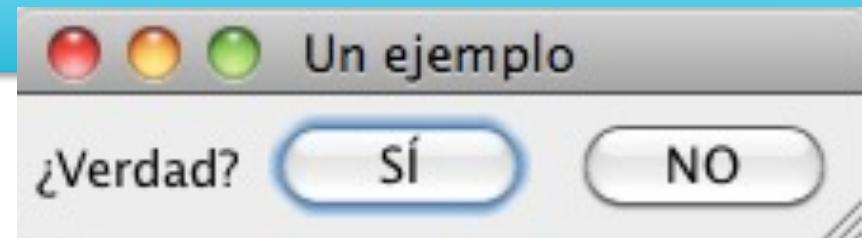
    public void actionPerformed(ActionEvent e) {
        String comando = e.getActionCommand();
        if (comando == PanelVentanaCtrExterno.SI)
            ven.cambiaTexto("Sí pulsado");
        else if (comando == PanelVentanaCtrExterno.NO)
            ven.cambiaTexto("No pulsado");
    }
}
```

La aplicación

```
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class PanelVentanaCtrExternoDemo {
    public static void main(String[] args) {
        PanelVentanaCtrExterno panel = new PanelVentanaCtrExternoPanel();
        ActionListener bt = new PanelVentanaCtrExternoCtr(panel);
        panel.controlador(bt);

        JFrame ventana = new JFrame("Un ejemplo con control");
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.getContentPane((JPanel) panel);
        ventana.pack();
        ventana.setVisible(true);
    }
}
```



Escenario posible

