

Interfaces Gráficas de Usuario



Francisco J. Martínez
Lenguajes y Ciencias de la Computación
Universidad de Málaga

E.T.S. INGENIERÍA INFORMÁTICA

Índice

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

Construcción de GUIs en Java

- Tecnología Swing.
- JFC (Java Foundation Classes):
 - AWT, Java 2D, Accesibility, Drag and Drop, Swing.
 - Versión 1.1 disponía sólo de AWT.
 - Versión 1.2 en adelante incluyen JFC: Swing.
- Swing está apoyado en parte en AWT.
- AWT: Abstract Window Toolkit.
 - La biblioteca se encuentra en el paquete `java.awt`.

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

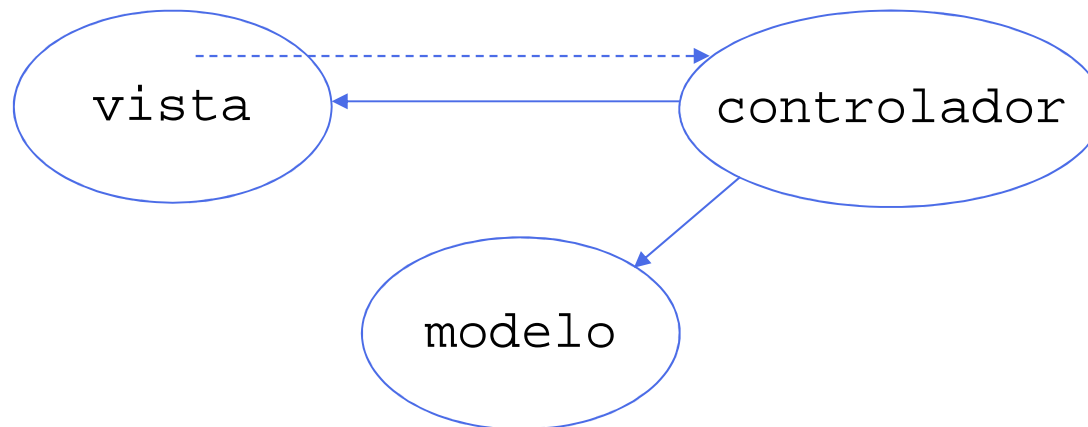
- Permiten la construcción de Interfaces Gráficas de Usuario (GUIs).
- SWING se basa en (y extiende) AWT.
- Se verán las características más importantes de SWING para construir GUIs básicos.

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`

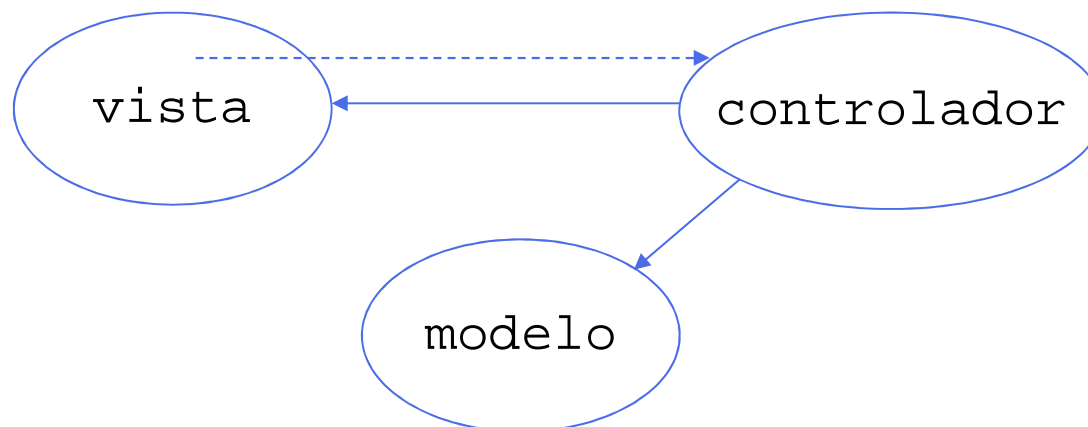
Diseño de interfaces de usuario

- 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.



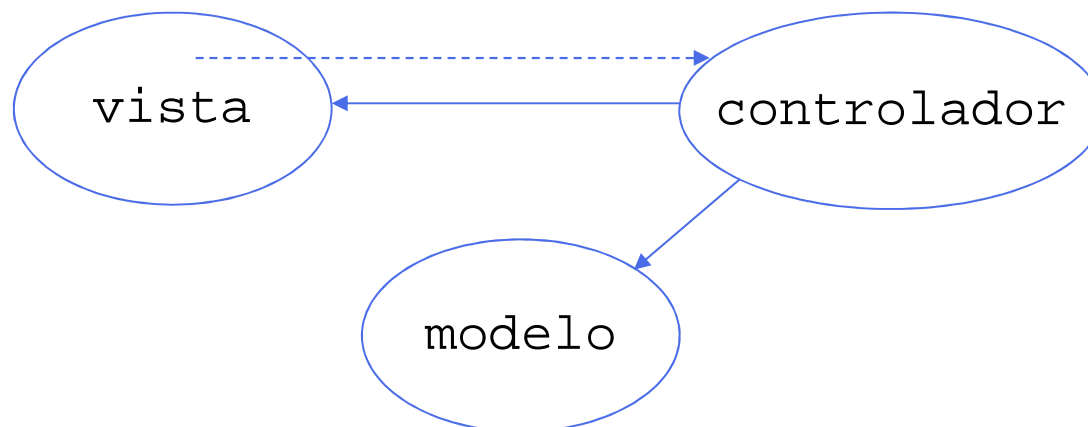
MVC: Modelo

- 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.



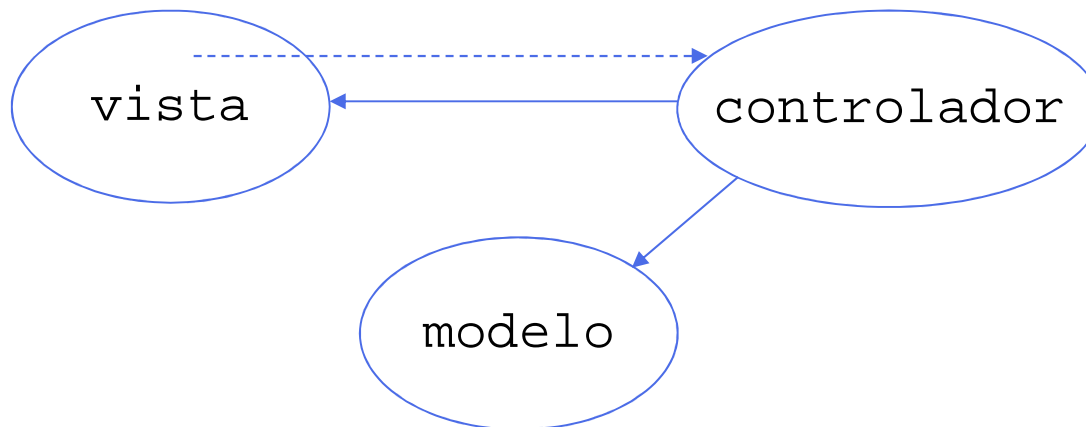
MVC: Vista

- Representación 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.



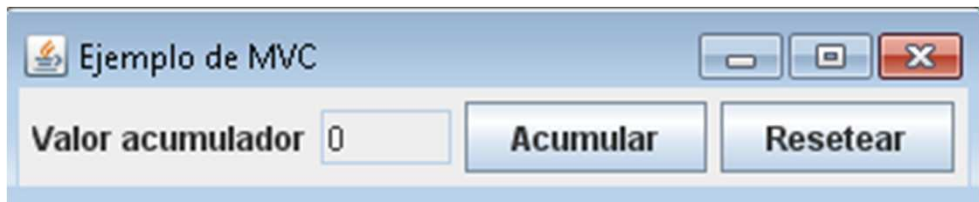
MVC: Controlador

- 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



controlador

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

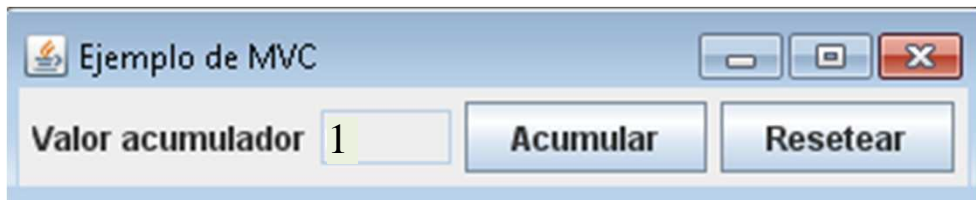


controlador

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

¿Cómo funciona el MVC?

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



1. Se ha pulsado “Acumular”

4. Actualiza el cuadro de texto

controlador

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

2. Envía el mensaje: acumular(1)

3. Envía el mensaje: obtenerAcumulador()

ejecutar

Ejemplo MVC Paso a Paso

- Crearemos:
 - El modelo: Una cuenta bancaria.
 - Clase **Cuenta.java**
 - La vista: Dos vistas distintas.
 - Interfaz **VistaCuenta.java**
 - Clases **PanelCuenta1.java** y **PanelCuenta2.java** que implementan la interfaz **VistaCuenta.java**
 - El controlador:
 - Clase **CtrCuenta.java**
 - Una aplicación **CuentaDemo.java** que crea el modelo, la vista y el controlador y establece las relaciones entre ellos.

Ejemplo MVC: La clase Cuenta

- Utilizaremos un objeto de esta clase como **modelo**.
 - Permite manipular una cuenta bancaria.
 - Métodos para:
 - Ingresar en la cuenta
`void ingresa(double)`
 - Extraer de la cuenta
`double extrae(double)`
 - Devuelve la cantidad realmente extraída según el saldo.
 - Consultar el saldo
`double saldo()`

El Modelo: La clase Cuenta

```
public class Cuenta {
    private double saldo;

    public Cuenta(double si) { //Si el saldo inicial es negativo se guarda 0
        saldo = Math.max(0, si);
    }
    public void ingresa(double ing) {
        saldo += ing;
    }
    public double extrae(double extrae) {
        double realExtrae = extrae;
        if (saldo < extrae) {
            realExtrae = saldo;
            saldo = 0;
        } else {
            saldo -= realExtrae;
        }
        return realExtrae;
    }
    public double saldo() {
        return saldo;
    }
}
```

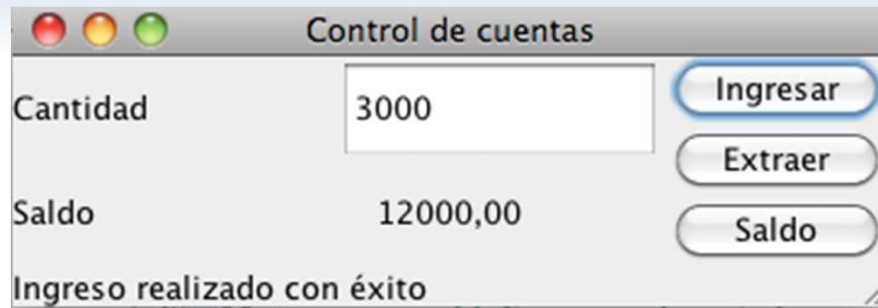
Sin GUI. Aplicación para la clase Cuenta

```
public class ApCuenta {  
    public static void main(String args[]) {  
        Cuenta cuenta = new Cuenta(Double.parseDouble(args[0]));  
        cuenta.ingresa(3000);  
        double realExt = cuenta.extrae(6000);  
  
        System.out.println("Saldo = " + cuenta.saldo());  
        System.out.println("Extraído = " + realExt);  
    }  
}
```

```
java ApCuenta 9000  
    Saldo          = 6000.0  
    Extraído       = 6000.0
```

```
java ApCuenta 2000  
    Saldo          = 0.0  
    Extraído       = 5000.0
```


Con GUI. Dos vistas para Cuenta



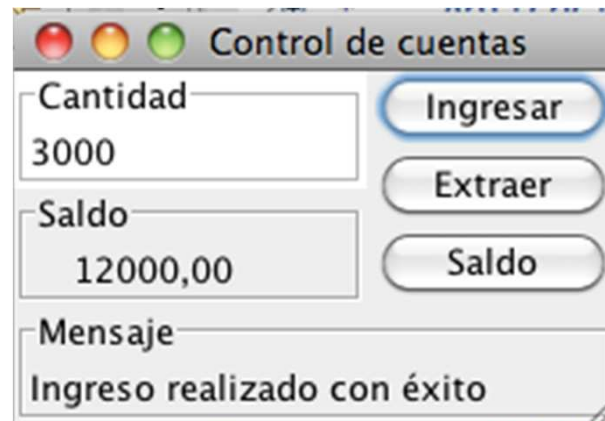
Controlador

CtrCuenta

Cuenta

Modelo

Vistas



Vistas: VistaCuenta

- Definimos una interfaz para las vistas:

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

```
public interface VistaCuenta {
```

```
    // ctes. Comandos
```

```
    // métodos para consultar/actualizar los elementos de la vista
```

```
    double obtenerCantidad();
```

```
    void saldo(double saldo);
```

```
    void mensaje(String msg);
```

```
    // método que registra el controlador en los elementos de la vista
```

```
    void controlador(ActionListener ctr);
```

```
}
```

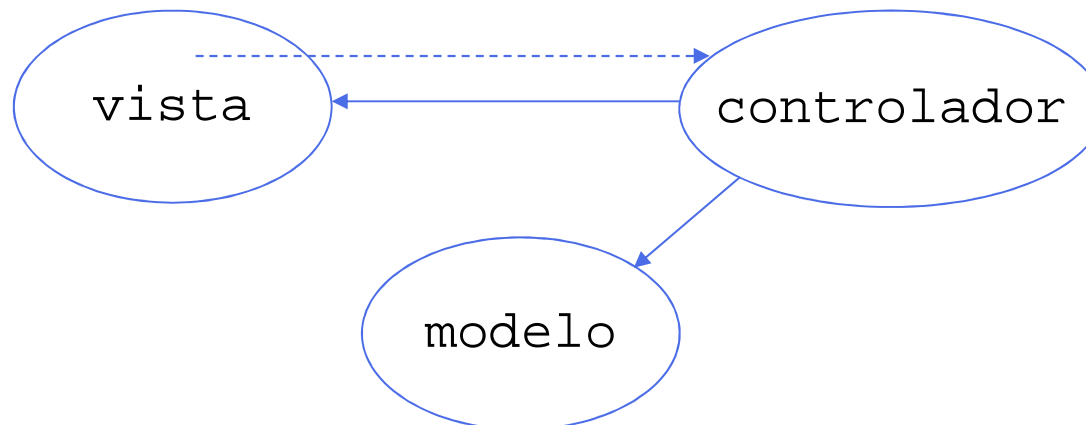
- El método **controlador(ActionListener ctr)**
 - Registra el controlador **ctr** en los componentes adecuados.
- Creamos dos vistas distintas sobre el mismo controlador
 - **PanelCuenta1** y **PanelCuenta2**

Lo vemos más adelante

Controlador: CtrCuenta

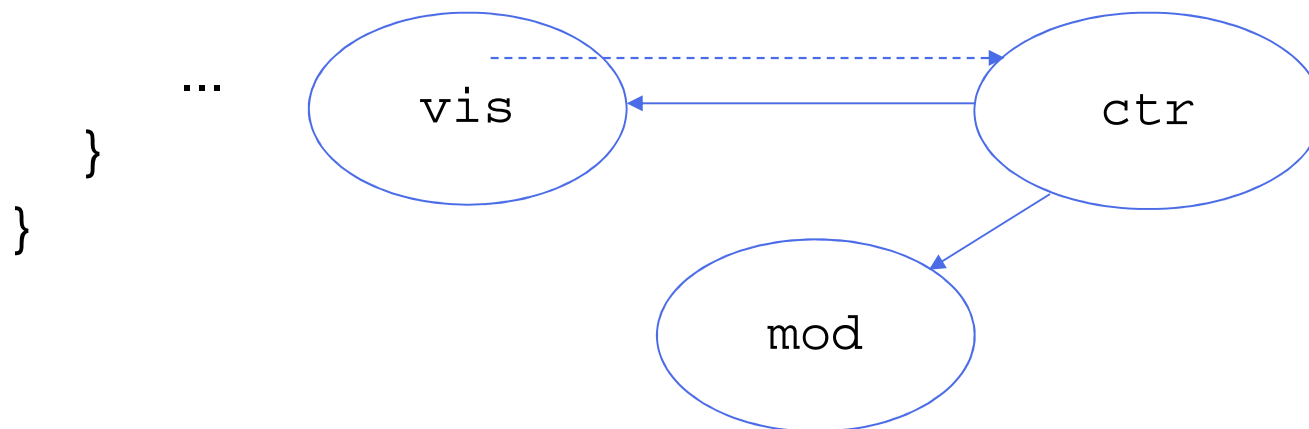
- Mantiene dos variables de instancia:
 - El modelo: **cuenta**
 - La vista : **vistaCuenta**

```
public CtrCuenta(VistaCuenta vc, Cuenta c ) {  
    vistaCuenta = vc;    // La vista  
    cuenta      = c;     // El modelo  
}
```



Aplicación con MVC

```
import javax.swing.*;  
public class CuentaDemo {  
    public static void main(String args[]) {  
        VistaCuenta vistaCuenta = new PanelCuenta2();  
        Préstamo cuenta = new Cuenta(3000);  
        CtrCuenta ctrCuenta = new CtrCuenta(vistaCuenta, cuenta);  
        vistaCuenta.controlador(ctrCuenta);  
    }  
}
```

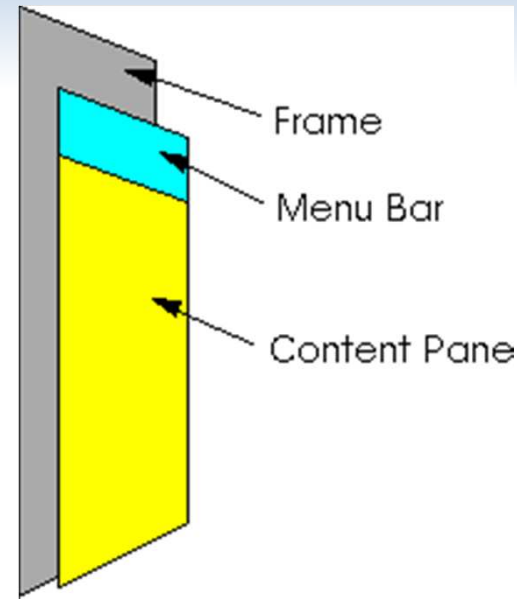
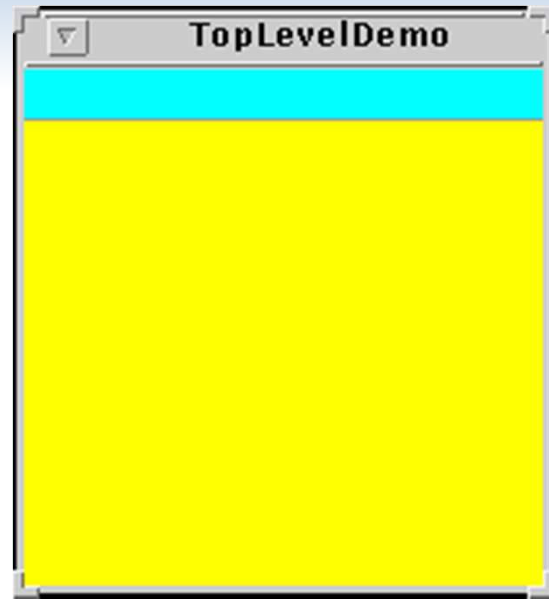


VISTAS

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

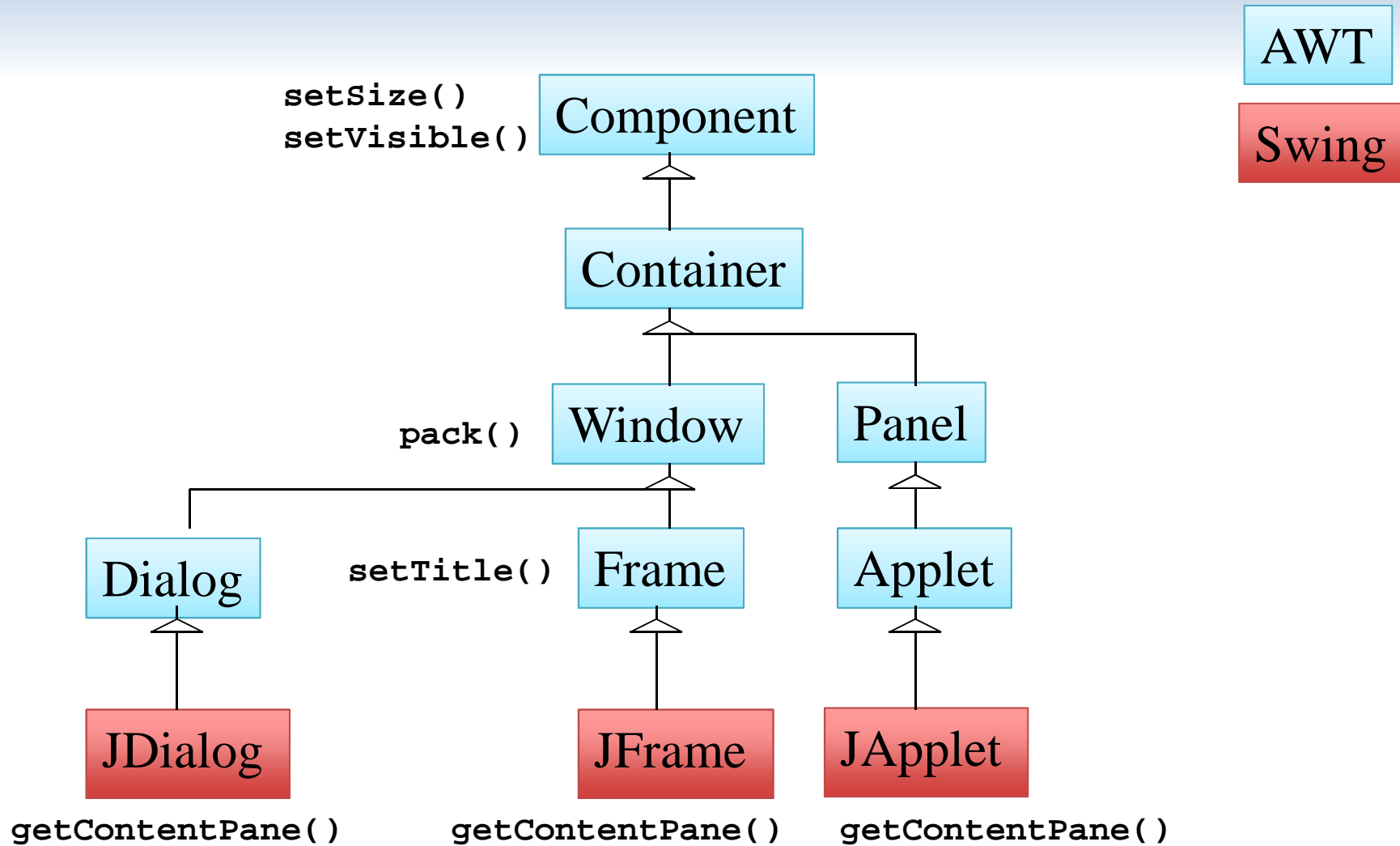


[ejecutar](#)

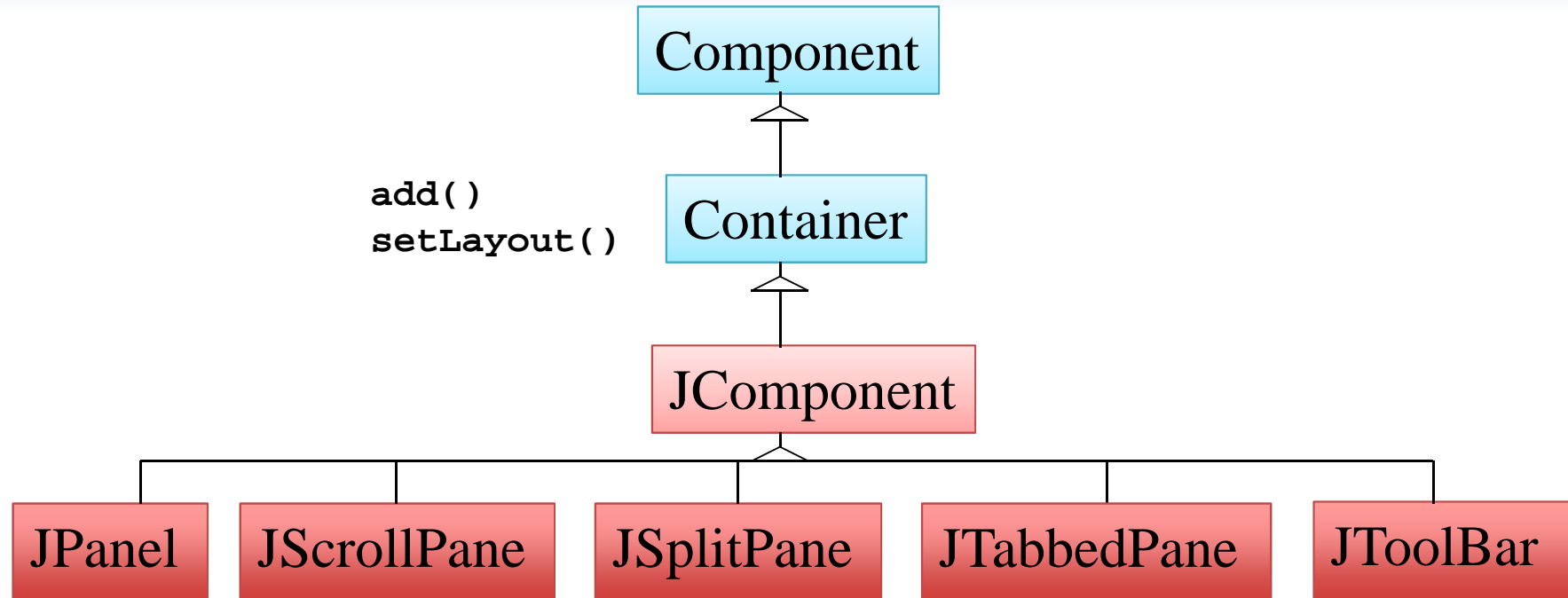
- Disponen de un panel de contenidos (`contentPane`) donde generalmente colocaremos la vista (este es el contenedor intermedio)
- Pueden opcionalmente disponer de un menú.

```
Container cpane = unFrame.getContentPane();  
unFrame.setContentPane(unPanel);  
unFrame.setJMenuBar(unMenuBar);
```

Contenedores superiores II



Contenedores intermedios



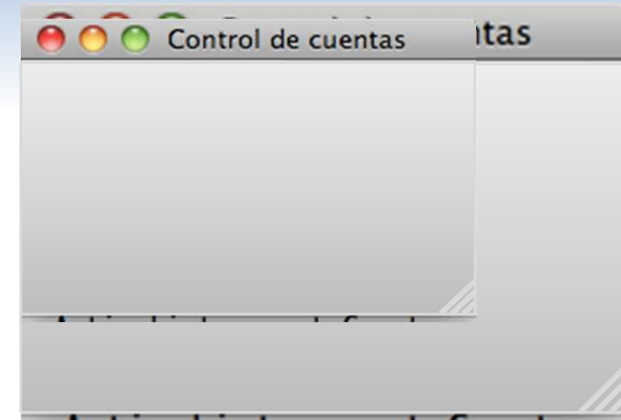
- El contenedor más utilizado es **JPanel**.

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.
- 2'. Crear una clase heredera de **JApplet**.
 1. Redefinir el método **void init()** y dentro de él cambiar su panel de contenidos por el objeto vista creado en el paso 1.

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.

1.1. Gestor de esquemas para contenedores intermedios

- Determinan cómo se distribuyen los componentes dentro de los contenedores.
 - Los gestores (clases) existentes son:
 - `FlowLayout`, `BorderLayout`, `GridLayout`, `GridBagLayout`, `BoxLayout`, ...
 - Cada contenedor tiene un gestor propio:
 - Por defecto `JPanel` tiene `FlowLayout`.
 - El método `void setLayout(LayoutManager)` permite cambiar de gestor.
 - Puede no utilizarse ningún gestor y colocar los componentes por medio del método de componentes `void setPosition(int, int)`
 - Esta opción no es recomendable.
- Ejemplo: para asignar un gestor de esquemas `FlowLayout`:
`setLayout(new FlowLayout())`
- La asignación del gestor de esquemas suele hacerse en el constructor del panel.

1.2. 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("¿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).

1.3. Agregar componentes al contenedor

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

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

public class PanelVentana 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("¿Verdad?"); // Componente declarado como variable 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**, **JDialog** y **JApplet**.
 - **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  
...
```


2.1. 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 PanelVentana());
```

- También disponen del método:

```
Container getContentPane()
```

para obtener su panel de contenidos por defecto.

- Esto proporciona otra alternativa para crear GUIs

2.2. 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)`

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

- Ésta es la forma recomendada para ajustar el tamaño.

2.3. 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);  
    }  
}
```

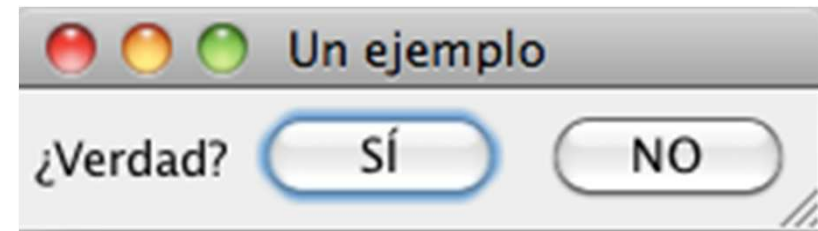
```
class PanelVentanaDemo {  
    public static void main(String[] args) {  
        JFrame ventana = new JFrame("Un ejemplo");  
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        ventana.setContentPane(new PanelVentana());  
        ventana.pack();  
        ventana.setVisible(true);  
    }  
}
```

```
// Creamos el contenedor superior
```

```
// Cambiamos el panel de contenidos
```

```
// Empaquetamos
```

```
// La hacemos visible
```



PanelVentana y PanelVentanaDemo

- 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);
```



2'. Crear un contenedor superior

- Hay tres clases de contenedores superiores:
 - **JFrame**, **JDialog** y **JApplet**.
 - **JFrame** -> Aplicaciones.
 - Ventana de nivel superior con bordes y título.
 - `setTitle(String)`, `getTitle()`, `setIconImage(Image)`.
 - **JApplet** -> Applets. Aplicaciones que corren en un navegador.
 - **JDialog** -> Diálogos. Tienen otro contenedor superior del que dependen.
 - Hay que heredar de **JApplet** y redefinir el método `void init()` para que cambie el panel de contenidos por la vista.

– Métodos de instancia ...

```
Container getContentPane() // Obtiene el panel de contenidos
void setContentPane(Container) // Cambia el panel de contenidos
void setJMenuBar(Menu) // Coloca un menú
...
```

2'.1. Heredar de JApplet

```
import javax.swing.JApplet;
import javax.swing.JPanel;

public class PanelVentanaApplet extends JApplet {
    public void init() {
        setContentPane(new PanelVentana());
    }
}
```

Se llama desde una página HTML: (**PanelVentana.html**)

```
<H1> Fichero de Pruebas </H1>
```

```
<H2> Lanzamiento de PanelVentana </H2>
```

```
<applet codebase=. code=PanelVentanaApplet width=300 height=100>
```

```
</applet>
```

Una forma alternativa

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

```
public class PanelVentanaDemo2 {  
    public static void main(String[] args) {
```

```
        JFrame ventana = new JFrame("Un ejemplo Alternativo 1");  
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        // No se utiliza otro panel intermedio
```

```
        Container cpane = ventana.getContentPane();
```

```
        cpane.setLayout(new FlowLayout());  
        JButton bSí = new JButton("Sí");  
        JButton bNo = new JButton("NO");  
        JLabel l = new JLabel("¿Verdad?");
```

```
        cpane.add(l);  
        cpane.add(bSí);  
        cpane.add(bNo);
```

```
        ventana.pack();  
        ventana.setVisible(true);
```

```
    }  
}
```

Obtenemos el panel de contenidos ...

... y creamos la vista en él

Realizar toda la construcción en el método **main()**:

- Se crea el contenedor superior **JFrame**
- Se obtiene su panel de contenidos
- Se construye la representación en este panel de contenidos.
- Se distribuye con **pack()**.
- Se muestra con **setVisible(true)**.

Otra forma alternativa

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

public class ApVentana extends JFrame {
    private JButton bSí;
    private JButton bNo;
    private JLabel l;

    public ApVentana(String s) {
        super(s);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container contP = getContentPane();
        contP.setLayout(new FlowLayout());
        bSí = new JButton("SÍ");
        bNo = new JButton("NO");
        l = new JLabel("¿Verdad?");
        contP.add(l);
        contP.add(bSí);
        contP.add(bNo);
    }
}

class ApVentanaDemo {
    public static void main(String[] args) {
        ApVentana ventana = new ApVentana("Un ejemplo Alt 2");
        ventana.pack();
        ventana.setVisible(true);
    }
}
```

Creamos un contenedor superior heredero de JFrame

... y por tanto dispone de un panel de contenidos donde crear la vista

- Realizar la ventana como subclase de **JFrame**:
 - El constructor llama a **super (String)** .
 - Toda la interfaz gráfica se crea en el constructor.
- Luego, se crea un método estático **main** que :
 - Crea un objeto de la clase **ApVentana**.
 - Lo distribuye con **pack()** .
 - Lo muestra con **setVisible(true)** .

GUI en Swing

- Queda por conocer:
 - Controlar el aspecto de la aplicación:
 - *Look and Feel*.
 - No lo veremos.
 - Usar adecuadamente los gestores de esquemas.
 - Únicamente los fundamentales.
 - Estudiar en detalle los componentes.
 - Únicamente los más utilizados.
 - Asociar acciones a los componentes.
 - El modelo general y los eventos más básicos.

Iconos

- En algunos constructores y métodos aparece un argumento **Icon** que representa un icono.
 - **Icon** es una interfaz.
 - La clase **ImageIcon** implementa esa interfaz.
- Para cargar un icono desde un fichero:

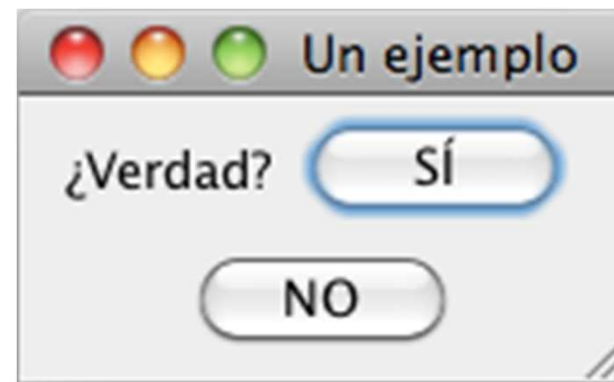
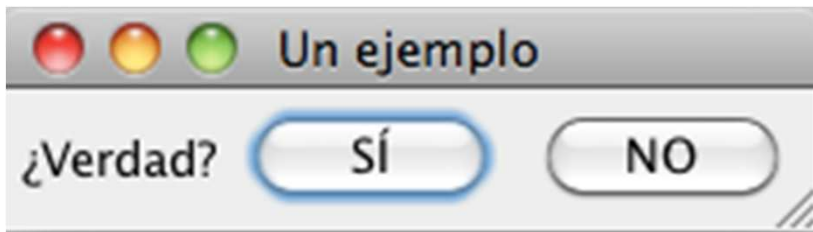
```
Icon i = new ImageIcon("/miDir/misIconos/bruja.gif")  
Icon j = new ImageIcon("miDir/misIconos/bruja.gif")
```

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.

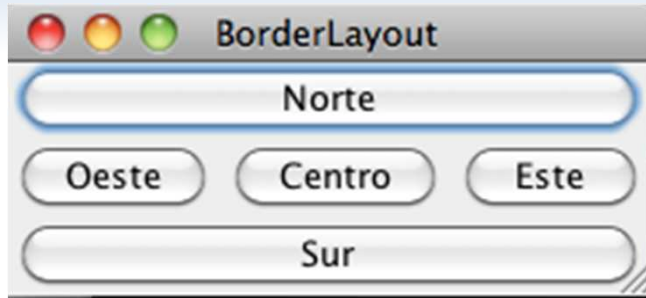


[ejecutar](#)

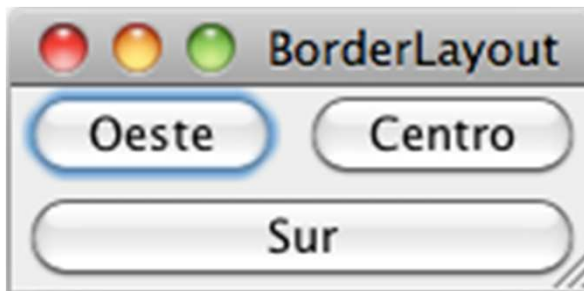
BorderLayout

- Divide el contenedor en 5 partes:
 - **NORTH, SOUTH, EAST, WEST y CENTER.**
 - Los componentes se ajustan hasta rellenar 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



[ejecutar](#)

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).

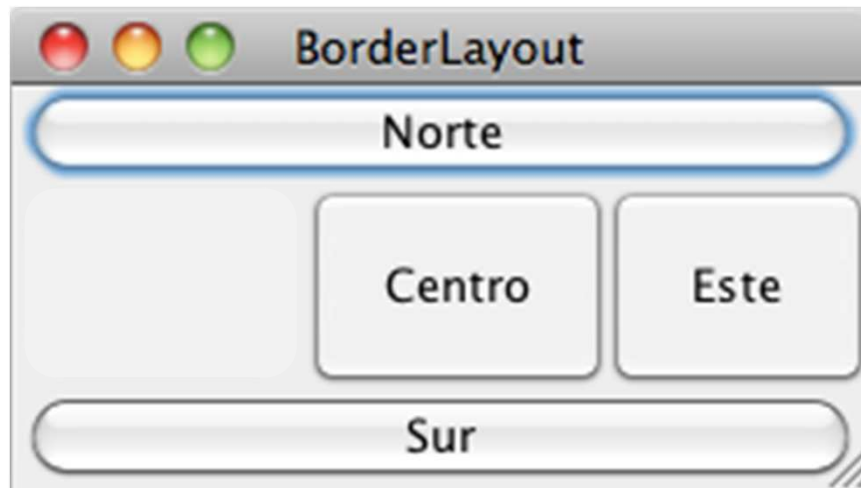
```
cpane.setLayout(new GridLayout(2, 3))
```

[ejecutar](#)

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());
```

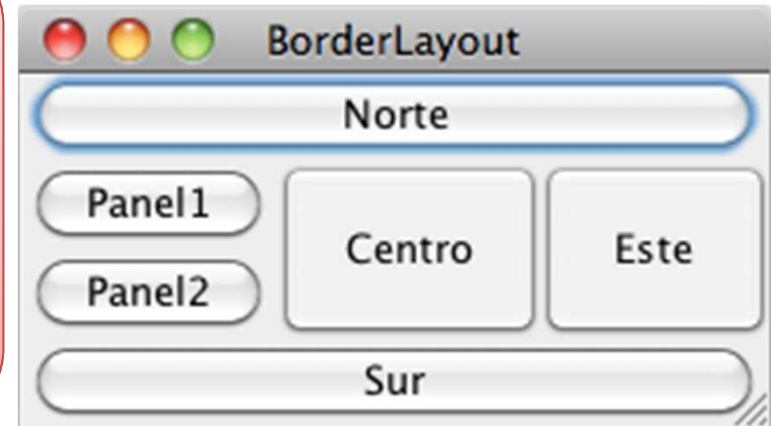
```
JPanel p = new JPanel();  
JButton bp1 = new JButton("Panel1");  
JButton bp2 = new JButton("Panel2");
```

```
p.setLayout(new GridLayout(2, 1));  
p.add(bp1);  
p.add(bp2);
```

...

```
add(p, BorderLayout.WEST);
```

...



[ejecutar](#)

JScrollPane I

- Permite hacer *scroll* sobre un componente (que puede ser otro contenedor intermedio)

- Constructores:

```
JScrollPane();
```

```
JScrollPane(Component);
```

```
JScrollPane(Component, int, int);
```

- Constantes desde la interfaz `javax.swing.ScrollConstants` para control del *scroll*:

(2º argumento)

```
VERTICAL_SCROLLBAR_AS_NEEDED
```

```
VERTICAL_SCROLLBAR_ALWAYS
```

```
VERTICAL_SCROLLBAR_NEVER
```

(3º argumento)

```
HORIZONTAL_SCROLLBAR_AS_NEEDED
```

```
HORIZONTAL_SCROLLBAR_ALWAYS
```

```
HORIZONTAL_SCROLLBAR_NEVER
```

- Métodos para introducir el componente dentro de un panel JScrollPane y para decidir cuando deben aparecer las barras de desplazamiento:

```
setViewportView(Component)
```

```
setHorizontalScrollBarPolicy(int)
```

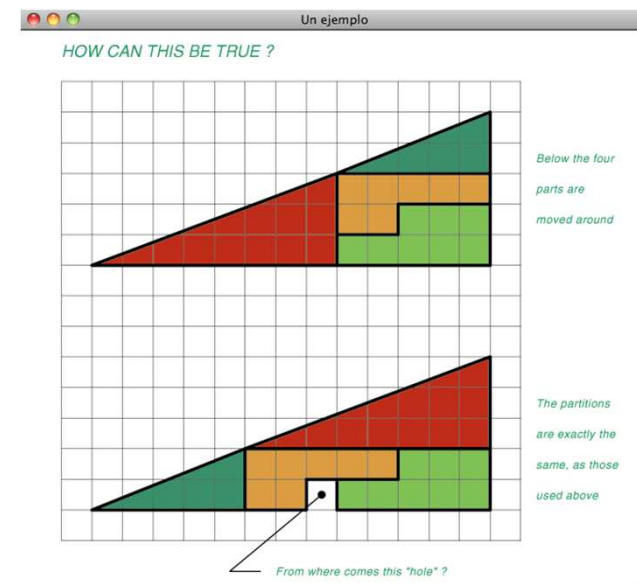
```
setVerticalScrollBarPolicy(int)
```

JScrollPane II

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

```
class ScrollDemo {  
    public static void main(String[] args) {  
        JFrame ventana = new JFrame("Un ejemplo");  
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        Icon ii = new ImageIcon("imagenes/triang.gif");  
        JLabel label = new JLabel(ii);  
        ventana.setContentPane(new JScrollPane(label));  
        ventana.setSize(670, 620);  
        ventana.setVisible(true);  
    }  
}
```

[ejecutar](#)



JSplitPane I

- Divide una ventana en dos:
 - Vertical u horizontal.
 - Podemos hacer que la redimensión sea visible o no.

- Constructores (entre otros):

```
JSplitPane()  
JSplitPane(int, Component, Component)  
JSplitPane(int, boolean, Component, Component)
```

- Constantes (1^{er} argumento):

```
HORIZONTAL_SPLIT                  VERTICAL_SPLIT
```

- Métodos de instancia:

```
setOneTouchExpandable(boolean) //forma de redimensionar  
setContinuousLayout(boolean) //repintar al mover el divisor  
setDividerLocation(int) //localización del divisor. Si negativo  
// intenta usar tamaño preferido de componentes
```

- Para introducir los componentes

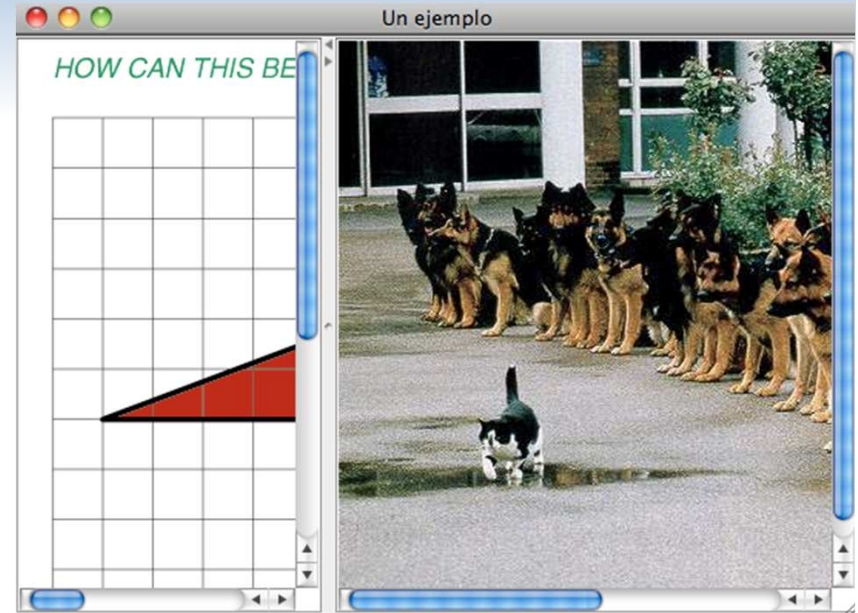
```
setLeftComponent(Component)                  setTopComponent(Component)  
setRightComponent(Component)                setBottomComponent(Component)
```

JSplitPane II

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

```
public class PanelSplit extends JSplitPane {  
    public PanelSplit(JLabel l1, JLabel l2) {  
        super(JSplitPane.HORIZONTAL_SPLIT,  
            new JScrollPane(l1),  
            new JScrollPane(l2));  
        setOneTouchExpandable(true);  
        setContinuousLayout(true);  
        setDividerLocation(200);  
    }  
}
```

```
class PaneSplitDemo {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("Un ejemplo");  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        Icon i1 = new ImageIcon("imagenes/triang.gif");  
        Icon i2 = new ImageIcon("imagenes/valor.jpg");  
        JLabel label1 = new JLabel(i1);  
        JLabel label2 = new JLabel(i2);  
        frame.setContentPane(new PanelSplit(label1, label2));  
        frame.setSize(550, 400);  
        frame.setVisible(true);  
    }  
}
```



[ejecutar](#)

JTabbedPane I

- Permite simular carpetas sobre la ventana.

- Constructores (entre otros):

```
JTabbedPane( )
```

```
JTabbedPane(int)
```

- Constantes desde la interfaz `javax.swing.SwingConstants` (1^{er} argum.):

```
TOP    BOTTOM    LEFT    RIGHT
```

- Métodos de instancia:

```
//1er parámetro: nombre de la carpeta
```

```
addTab(String, Component);
```

```
addTab(String, Icon, Component);
```

```
//4º parámetro: ayuda mostrada al pasar por el nombre carpeta
```

```
addTab(String, Icon, Component, String);
```

```
setSelectedIndex(int);
```

JTabbedPane II

Icon icon =

```
new ImageIcon("imagenes/online.gif");
```

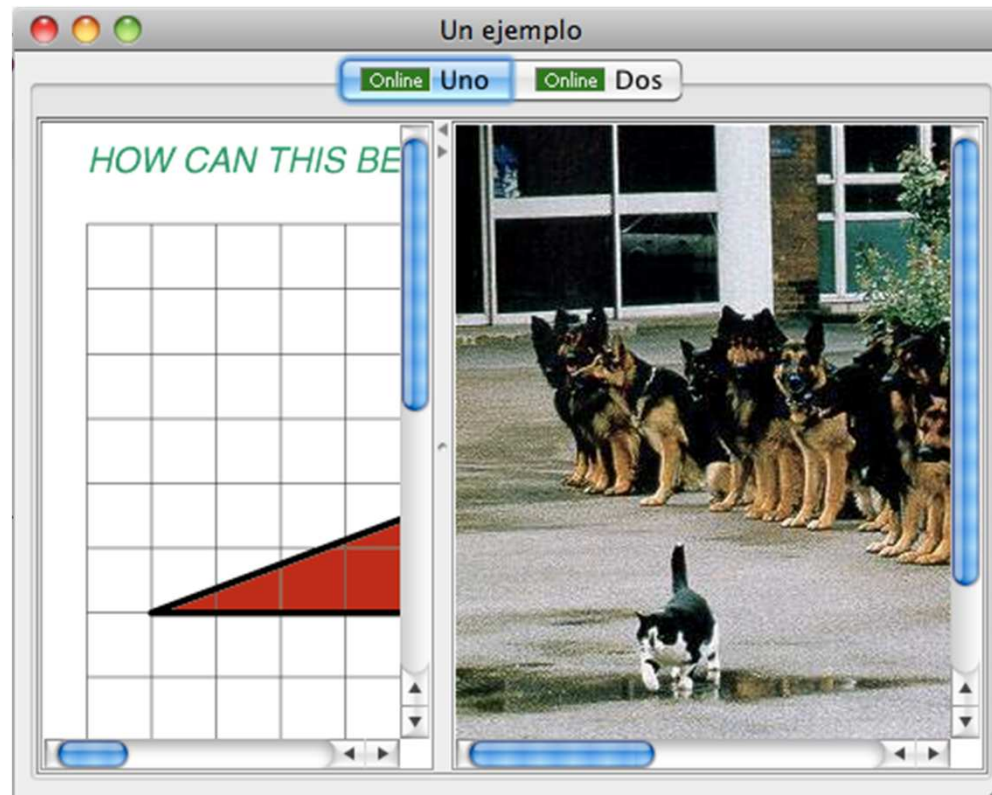
...

```
addTab("Uno", icon, splitPane, "Primer panel");
```

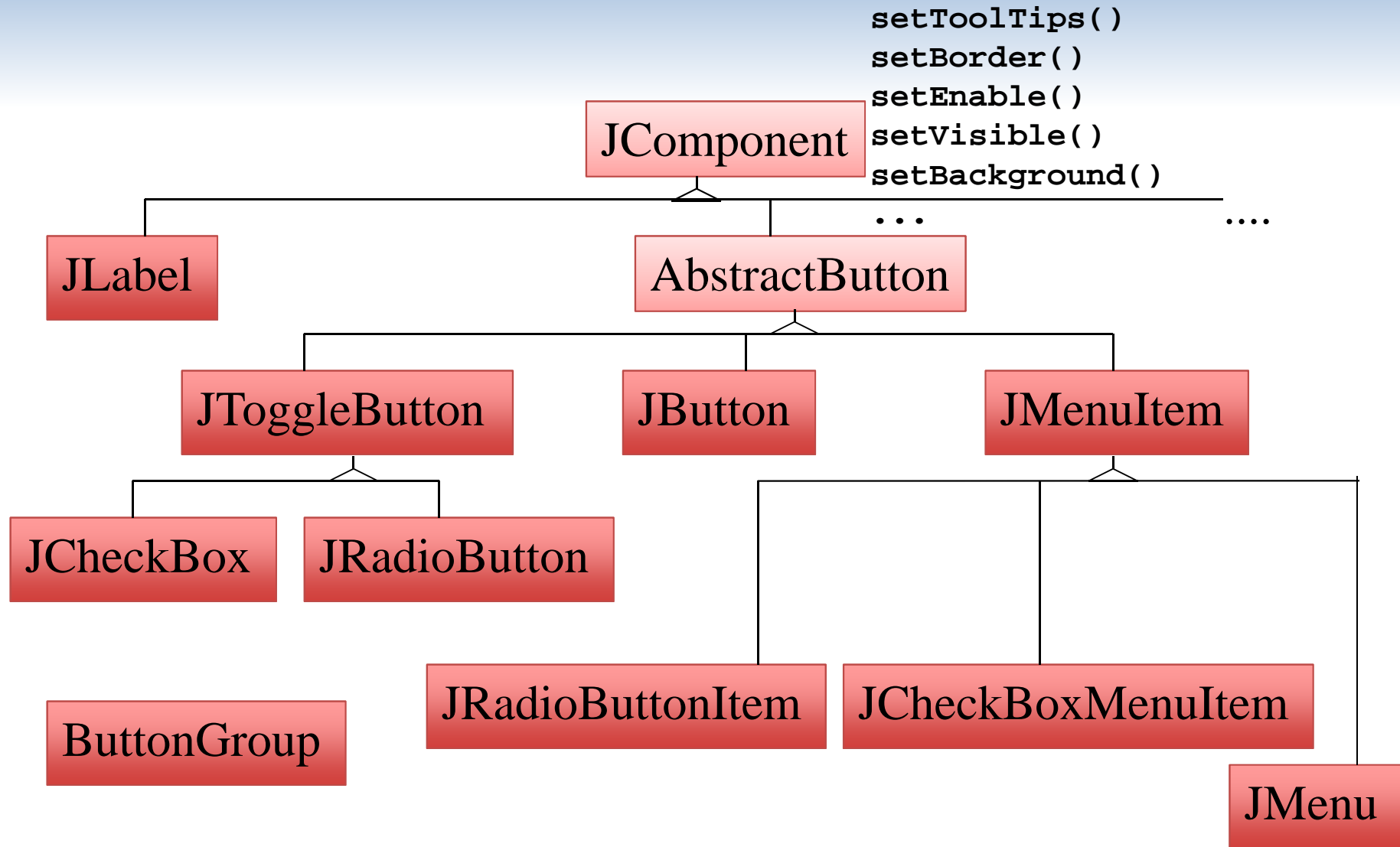
...

```
addTab("Dos", icon, scrollPane, "Segundo panel");
```

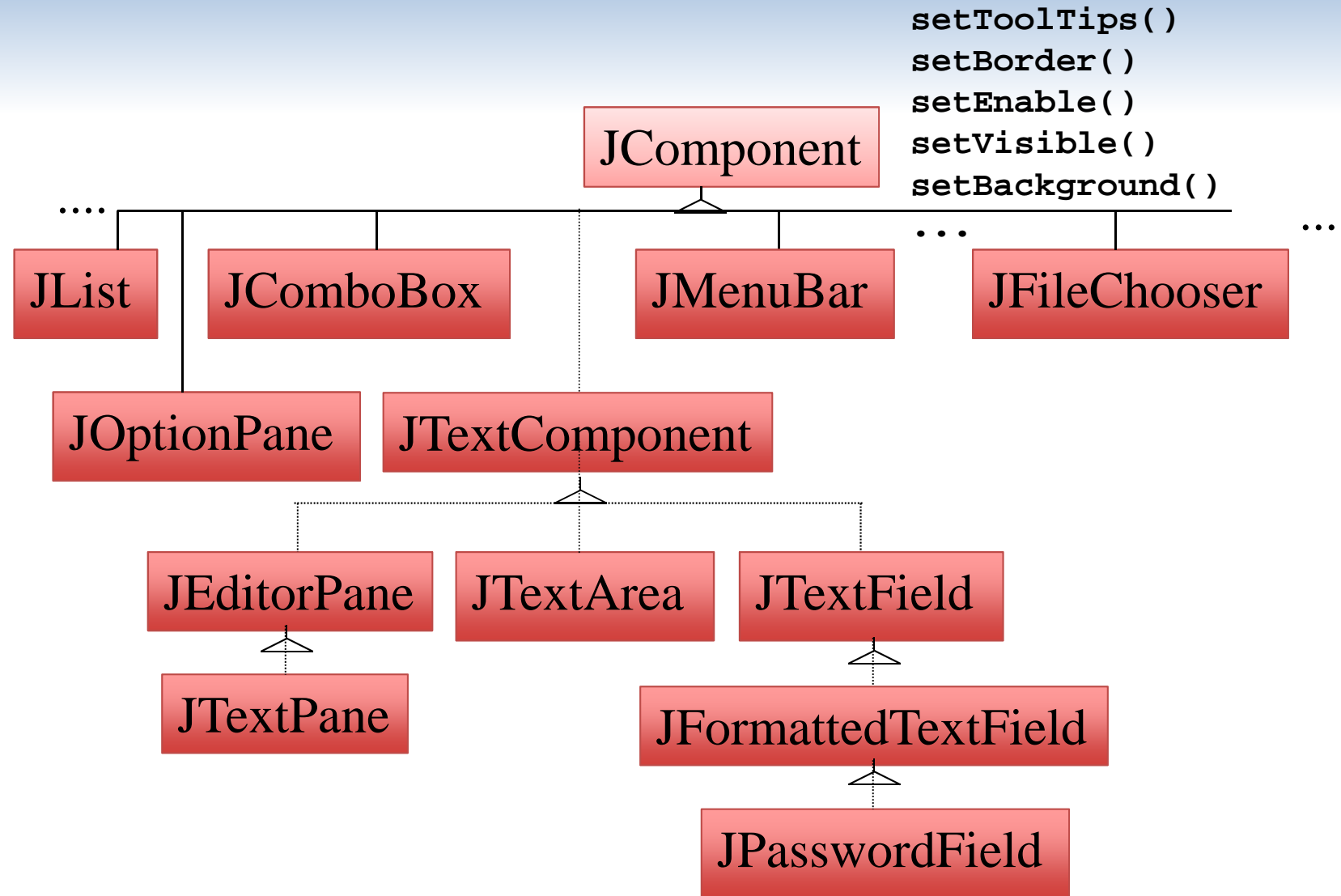
[ejecutar](#)



Componentes I



Componentes II



Componentes III

- Métodos heredados de **JComponent** (y sus superclases):
 - `Color` `getBackground()`
 - `void` `setBackground(Color)`
 - `Graphics` `getGraphics()`
 - `String` `getName()`
 - `Toolkit` `getToolkit()`
 - `void` `setEnabled(boolean)`
 - `void` `setVisible(boolean)`
 - `void` `paint(Graphics g)`
 - `void` `paintComponent(Graphics g)`
 - `void` `repaint()`
 - `void` `setBorder()`
 - `void` `setAlignmentX(float)`
 - Component: `LEFT_ALIGNMENT`, `CENTER_ALIGNMENT`, `RIGHT_ALIGNMENT`
 - `void` `setAlignmentY(float)`
 - Component: `TOP_ALIGNMENT`, `CENTER_ALIGNMENT`, `BOTTOM_ALIGNMENT`

Bordes I

- En el paquete `javax.swing.borders` existen clases que permiten definir un borde a un componente:
 - `AbstractBorder`
 - `BevelBorder`
 - `CompoundBorder`
 - `EmptyBorder`
 - `EtchedBorder`
 - `LineBorder`
 - `MatteBorder`
 - `SoftBevelBorder`
 - `TitleBorder`

Bordes II

- Para cambiar el borde de un componente:

```
public void setBorder(Border)
```

```
JTextField jtf= new JTextField(15);  
jtf.setBorder(new TitledBorder("Nombre"));
```

- La clase `javax.swing.BorderFactory` dispone de métodos de clase (métodos factoría) para crear bordes:

```
JTextField jtf= new JTextField(15);  
jtf.setBorder(BorderFactory.createTitleBorder("Nombre"));
```

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)
```

```
...
```


JRadioButton y ButtonGroup

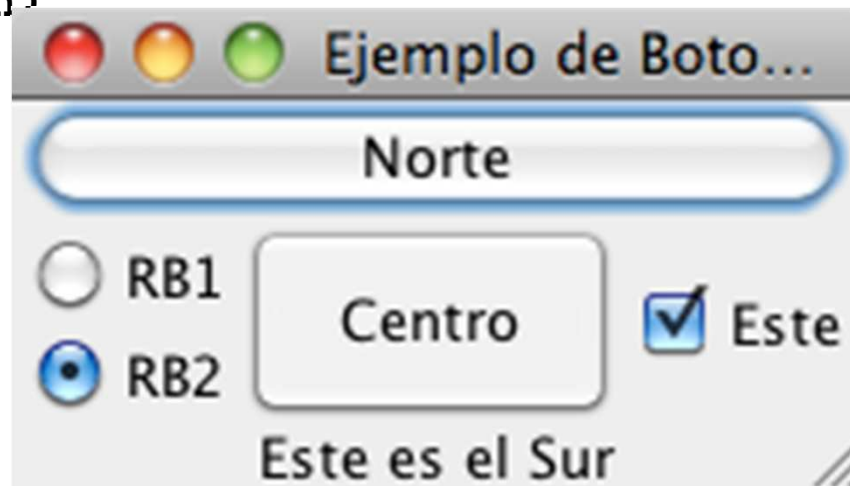
- Botones circulares (utilizados para selección alternativa).
- Se agrupan de manera que sólo uno esté pulsado.
- Constructores:

```
JRadioButton([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.*;
```

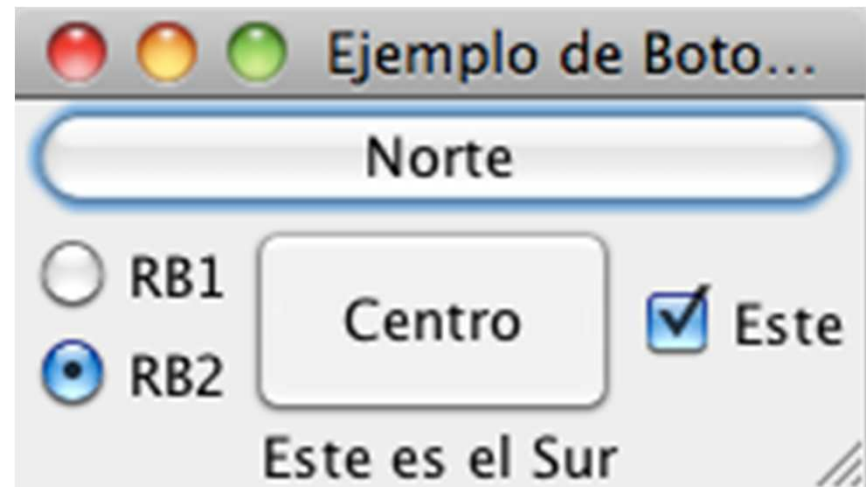
```
public class PanelBotones extends JPanel {  
    public PanelBotones() {  
        JButton bNorte = new JButton("Norte");  
        JLabel lSur = new JLabel("Éste es el Sur", JLabel.CENTER);  
        JCheckBox cEste = new JCheckBox("Este", true);  
        JButton bCentro = new JButton("Centro");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

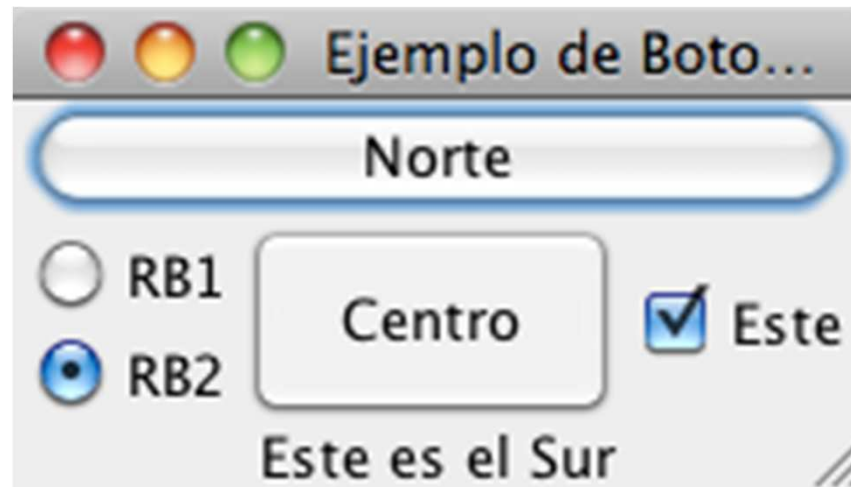
```
    }  
}
```



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.setContentPane(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);  
...
```

JList

- Muestra una lista de elementos para su selección.

- Constructores:

```
JList()           JList(Object[])  
JList(Vector<?>) JList(ListModel)
```

- Métodos de instancia:

```
int  getSelectedIndex() // -1 si no hay  
int[] getSelectedIndices()  
Object getSelectedValue()  
Object[] getSelectedValues()  
boolean isSelectedIndex(int)  
boolean isEmptySelection()  
void setListData(Object[])  
void setListData(Vector<?>)  
void setSelectionMode(int)  
int  getSelectionMode()
```

...

- Constantes de la interfaz `ListSelectionModel`:

```
SINGLE_SELECTION  
SINGLE_INTERVAL_SELECTION  
MULTIPLE_INTERVAL_SELECTION
```

JComboBox

- Permite la selección de un ítem de entre varios.
- No está desplegado como **Jlist**.

– Constructores:

```
JComboBox()           JComboBox(Object[])  
JComboBox(Vector<?>) JComboBox(ComboBoxModel)
```

– Métodos de instancia:

```
int getSelectedIndex()  
Object getSelectedItem()  
void setSelectedIndex(int)  
boolean isEditable()  
void setEditable(boolean)
```

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:

addXxxxxListener (XxxxxListener)

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

addActionListener(ctr)

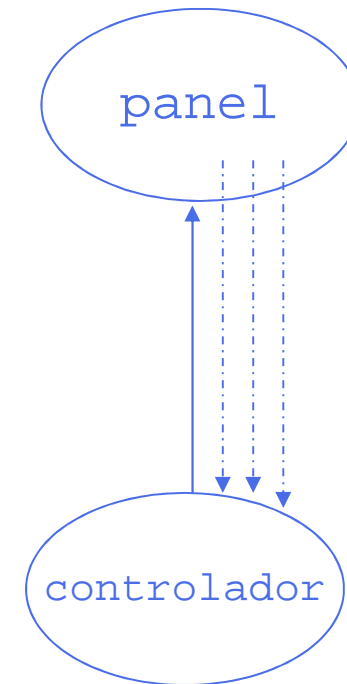
Relaciones Vista-Controlador

```
// 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(ctr);
```

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



Interfaces en `java.awt.event` I

Interfaces para controladores

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

Interfaces en `java.awt.event` II

Intefaces para controladores

MouseListener	<code>mouseClicked(MouseEvent)</code> <code>mouseEntered(MouseEvent)</code> <code>mouseExited(MouseEvent)</code> <code>mousePressed(MouseEvent)</code> <code>mouseReleased(MouseEvent)</code>
MouseMotionListener	<code>mouseDragged(MouseEvent)</code> <code>mouseMoved(MouseEvent)</code>
TextListener	<code>textValueChanged(TextEvent)</code>
WindowListener	<code>windowActivated(WindowEvent)</code> <code>windowClosed(WindowEvent)</code> <code>windowClosing(WindowEvent)</code> <code>windowDeactivated(WindowEvent)</code> <code>windowDeiconified(WindowEvent)</code> <code>windowIconified(WindowEvent)</code> <code>windowOpened(WindowEvent)</code>

Eventos

- Los eventos se implementan como subclasses de **java.util.EventObject**
- Método de instancia para conocer quién provoca el evento:
Object getSource()
- Los eventos se encuentran en los paquetes **java.awt.event** y **javax.swing.event**
 - Evento **XxxxxEvent**
 - Interfaces **XxxxxListener**
 - Para registrar un controlador se utiliza:
addXxxxxListener(XxxxxListener)
 - Todos los métodos de la interfaz tendrán como argumento:
XxxxxEvent

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 String SI = "SI";  
    public static 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;

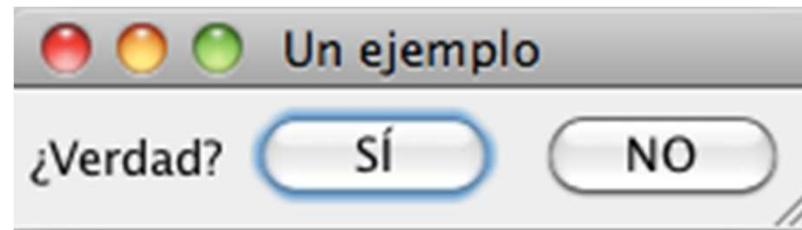
    public PanelVentanaCtrExternoCtr(PanelVentanaCtrExterno v) {
        ven = v;
    }

    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");
    }
}
```

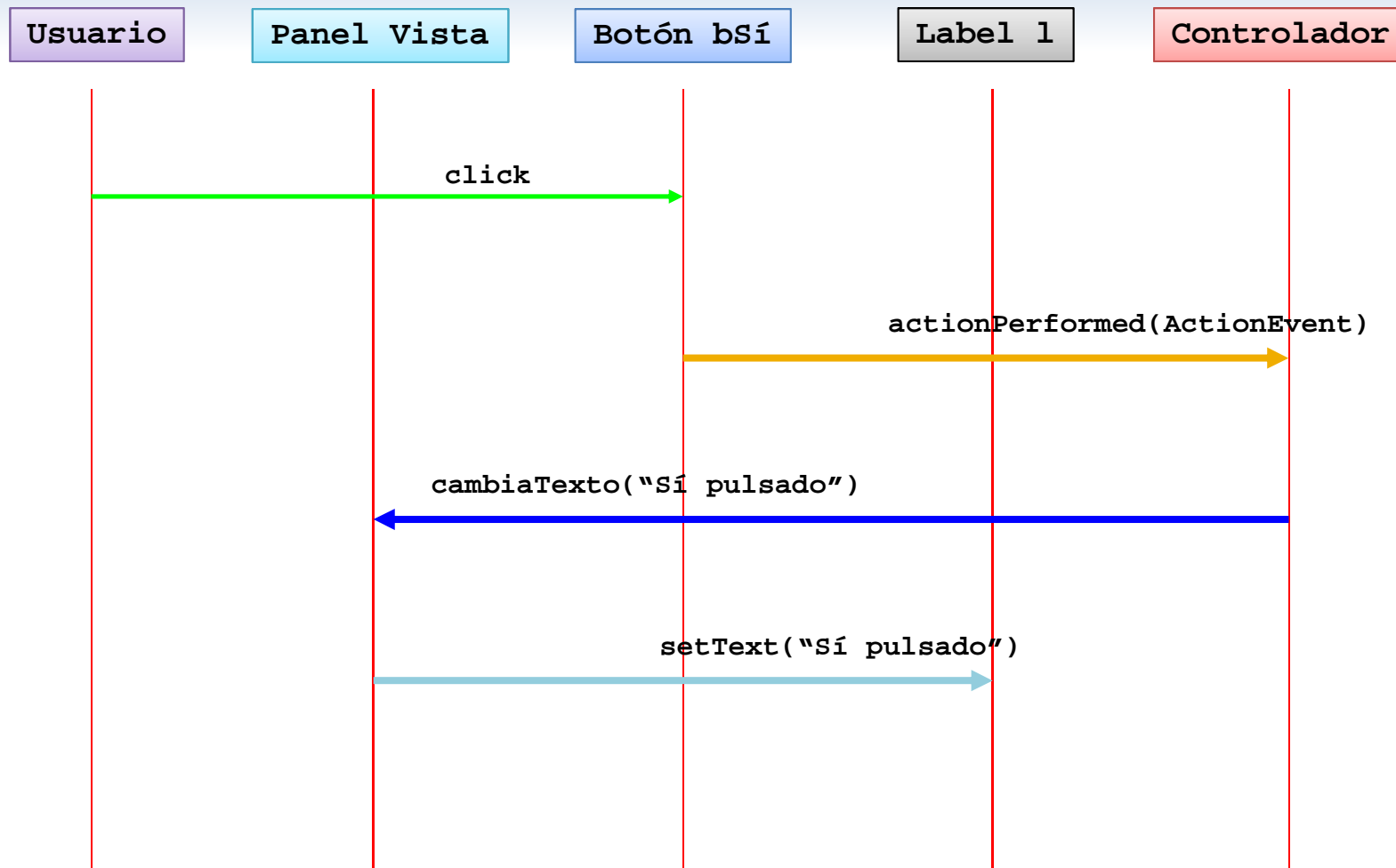
La aplicación

```
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.setContentPane((JPanel) panel);
        ventana.pack();
        ventana.setVisible(true);
    }
}
```



Escenario posible



Control alternativo I: El controlador es la propia vista

```
public class PanelVentanaCtrInterno extends JPanel
implements ActionListener {
    private JButton bSí, bNo;
    private JLabel l;

    public PanelVentanaCtrInterno() {
        setLayout(new FlowLayout());
        bSí = new JButton("SÍ");
        bNo = new JButton("NO");
        l = new JLabel("¿Verdad?");
        bSí.addActionListener(this);
        bSí.setActionCommand("SÍ");
        bNo.addActionListener(this);
        bNo.setActionCommand("NO");
        add(l);
        add(bSí);
        add(bNo);
    }

    public void actionPerformed(ActionEvent e) {
        String comando = e.getActionCommand();
        if (comando.equals("SÍ"))
            l.setText("SÍ pulsado");
        else if (comando.equals("NO"))
            l.setText("No pulsado");
    }
}
```

Control alternativo II: El controlador tiene visibilidad

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

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

    public PanelVentanaCtrAnonimo() {
        setLayout(new FlowLayout());
        bSí = new JButton("Sí");
        bNo = new JButton("NO");
        l = new JLabel("Pulsaciones");
        BotonControl bc = new BotonControl();
        bSí.addActionListener(bc);
        bSí.setActionCommand("Sí");
        bNo.addActionListener(bc);
        bNo.setActionCommand("NO");
        add(l);
        add(bSí);
        add(bNo);
    }

    class BotonControl implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            String comando = e.getActionCommand();
            if (comando.equals("Sí"))
                l.setText("Si pulsado");
            else if (comando.equals("NO"))
                l.setText("No pulsado");
        }
    }
}
```



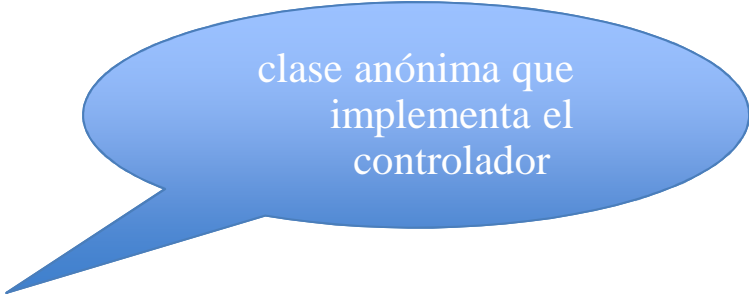
Clase anidada no
estática

Control alternativo III: Controladores con clases anónimas

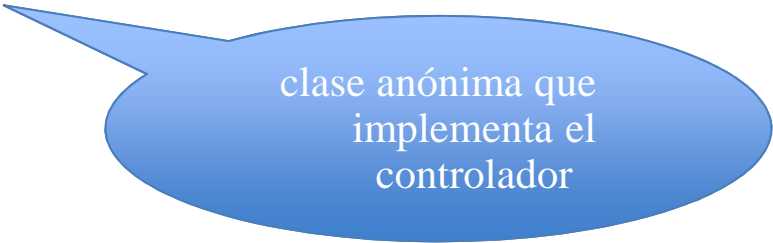
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

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

    public PanelVentanaCtrThis() {
        setLayout(new FlowLayout());
        bSí = new JButton("Sí");
        bNo = new JButton("NO");
        l = new JLabel("Pulsaciones");
        bSí.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                l.setText("Sí pulsado");
            }
        });
        bNo.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                l.setText("No pulsado");
            }
        });
        add(l);
        add(bSí);
        add(bNo);
    }
}
```



clase anónima que
implementa el
controlador



clase anónima que
implementa el
controlador

Pintar con Swing I

- Todo componente dispone del método:
public void paint(Graphics g)
donde **g** es el contexto gráfico sobre el que se puede pintar. Este contexto está restringido al área que ocupa el componente.
- El método **paint** llama a los tres métodos siguientes:
 - **protected void paintComponent(g)**
 - Pinta el componente en sí.
 - **protected void paintBorder(g)**
 - Pinta los bordes del componente.
 - **protected void paintChildren(g)**
 - Pinta los componentes contenidos en él si es que era un contenedor.
 - Es decir, llama a **paint(gg)** para cada componente contenido, donde **gg** es el contexto gráfico de dicho componente.

Pintar con Swing II

- Para pintar sobre un componente se debe redefinir `public void paintComponent(Graphics)` que pinta en el área rectangular del componente.
 - Se debe llamar al método redefinido.
- El contexto gráfico contiene entre otras cosas información del área a pintar.
- Dispone de métodos para pintar:

```
void drawString(String,int,int)
void drawLine(int,int,int,int)
void drawRect(int,int,int,int)    void fillRect(int,int,int,int)
void drawArc(int,int,int,int,int,int)  void fillArc(int,int,int,int,int,int)
void draw3DRect(int,int,int,int,boolean)  void fill3DRect(int,int,int,int,boolean)
void drawOval(int,int,int,int)    void fillOval(int,int,int,int)
void drawPolygon(int[],int[],int)    void fillPolygon(int[],int[],int)
...                                  ...
```

Pintar con Swing III

- *Nunca* se debe llamar a los métodos **paint(Graphics)** y **paintComponent(Graphics)** de un componente.
- El método **paint(Graphics)** del componente será llamado:
 - Al mostrarse la ventana que lo contiene.
 - Cada vez que se oculte por otra ventana y luego se haga visible.
 - Al moverse o cambiar de tamaño la ventana que lo contiene.
 - Al maximizar o restaurar la ventana que lo contiene.
 - Cuando se llama al método **repaint()**.
- Llamadas sucesivas a **repaint()** generan una única llamada a **paint(Graphics)**.

Pintar con Swing IV. Ejemplo

```
import java.awt.*;
import javax.swing.*;
public class SóloCírculo extends JPanel {
    public SóloCírculo() {
        setBackground(Color.GREEN);
        setForeground(Color.WHITE);
        setPreferredSize(new Dimension(200, 200));
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.fillOval(75, 75, 50, 50);
    }
}

class SóloCírculoDemo {
    public static void main(String[] args) {
        JFrame ventana = new JFrame("Un círculo");
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setContentPane(new SóloCírculo());
        ventana.pack();
        ventana.setVisible(true);
    }
}
```

