

Introducción

- *Swing* es una biblioteca de interfaces gráficas de usuario (GUI) para Java.
- Viene incluida con el entorno de desarrollo de Java (JDK).
- Extiende a otra librería gráfica más antigua llamada *AWT*.
- Paquetes:
 - *javax.swing*
 - *java.awt*
 - *java.awt.event*
- Alternativas: SWT (<http://www.eclipse.org/swt/>)



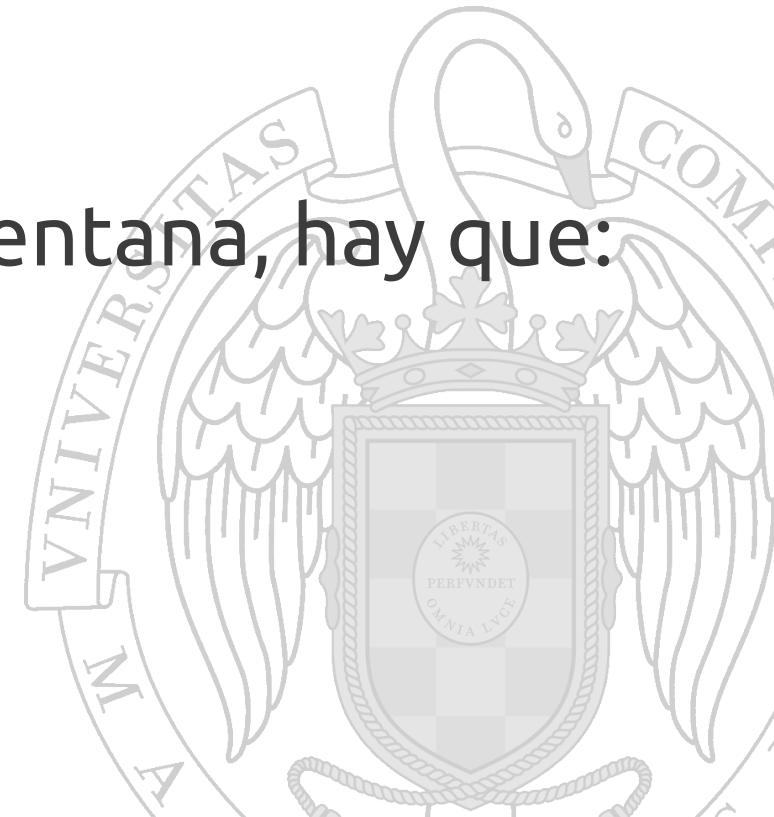
Contenidos

- Ventanas
- Componentes
- Layout Managers
- Manejo de eventos
- Cuadros de diálogo predefinidos
- Dibujo de gráficos
- Arquitectura MVC



Creación de ventanas

- La clase `JFrame` proporciona operaciones para manipular ventanas.
- Constructores:
 - `JFrame()`
 - `JFrame(String titulo)`
- Una vez creado el objeto de ventana, hay que:
 - Establecer su tamaño.
 - Establecer la acción de cierre.
 - Hacerla visible.

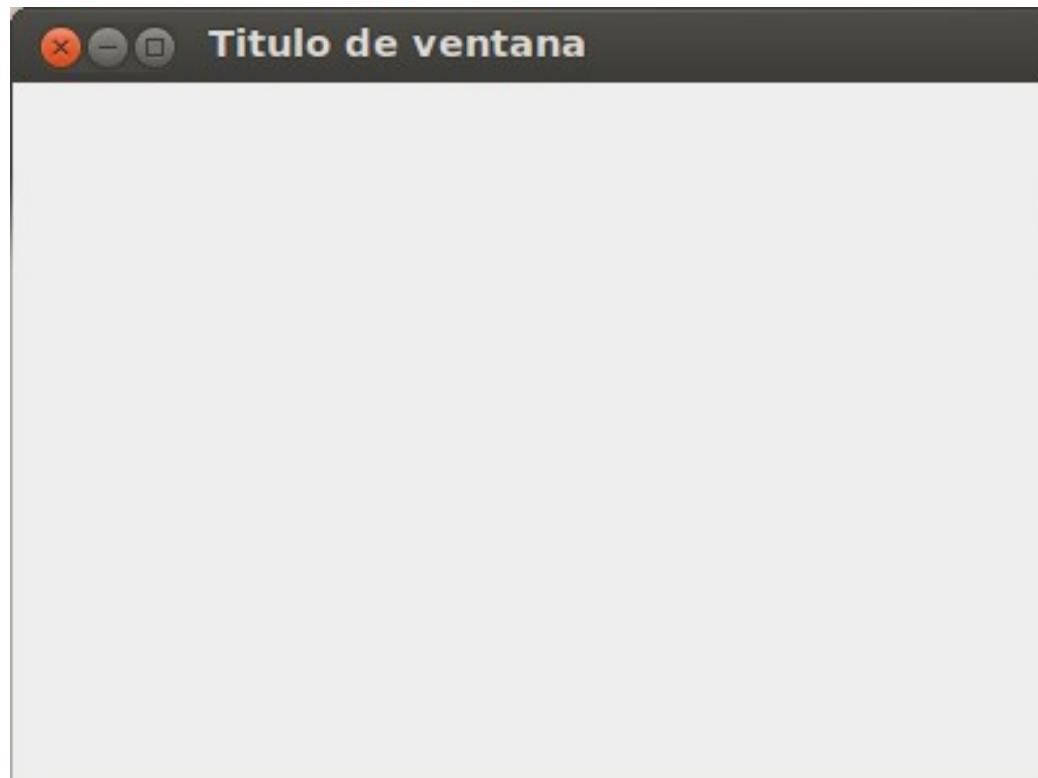


Creación de ventanas

```
import javax.swing.*;  
  
public class VentanaTest {  
    public static void main(String[] args) {  
        JFrame f = new JFrame("Titulo de ventana");  
        f.setSize(400, 300);  
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        f.setVisible(true);  
    }  
}
```

- Acciones de cierre:
 - `JFrame.EXIT_ON_CLOSE`: Abandona aplicación.
 - `JFrame.DISPOSE_ON_CLOSE`: Libera los recursos asociados a la ventana.
 - `JFrame.DO NOTHING_ON_CLOSE`: No hace nada.
 - `JFrame.HIDE_ON_CLOSE`: Cierra la ventana, sin liberar sus recursos.

Creación de ventanas



Creación de ventanas

- Es usual extender la clase JFrame, y realizar las operaciones de inicialización en su constructor.

```
public class MiVentana extends JFrame {  
    public MiVentana() {  
        super("Titulo de ventana");  
        setSize(400, 300);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}
```

```
public class VentanaTest {  
    public static void main(String[] args) {  
        MiVentana v = new MiVentana();  
        v.setVisible(true);  
    }  
}
```



Contenidos

- Ventanas
- Componentes
- Layout Managers
- Manejo de eventos
- Cuadros de diálogo predefinidos
- Dibujo de gráficos
- Arquitectura MVC



Componentes de una ventana



JButton

Etiqueta:

JLabel



JTextField

Caja de verificación

JCheckBox

Botones de radio

JRadioButton

- Tras crear uno de estos componentes con new, ha de añadirse al contentPane de la ventana correspondiente mediante su método add.

Añadir componentes

```
public class MiVentana extends JFrame {  
    public MiVentana() {  
        super("Titulo de ventana");  
        setSize(400, 300);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        Container cp = getContentPane();  
        cp.setLayout(new FlowLayout());  
        JLabel etiqueta = new JLabel("Nombre: ");  
        JTextField texto = new JTextField(20);  
        JButton boton = new JButton("Saludar");  
        cp.add(etiqueta);  
        cp.add(texto);  
        cp.add(boton);  
    }  
}
```



Añadir componentes



Contenidos

- Ventanas
- Componentes
- Layout Managers
- Manejo de eventos
- Cuadros de diálogo predefinidos
- Dibujo de gráficos
- Arquitectura MVC



Layout Managers

- En Java no es habitual indicar explícitamente la posición de los componentes de la interfaz dentro de la ventana.
- Los *layout managers* se encargan de colocar los componentes de la interfaz de usuario en la ventana contenedora.
- Especifican la **posición** y el **tamaño** de dichos componentes.
 - FlowLayout
 - GridLayout
 - BorderLayout
 - GridBagConstraints
 - ...



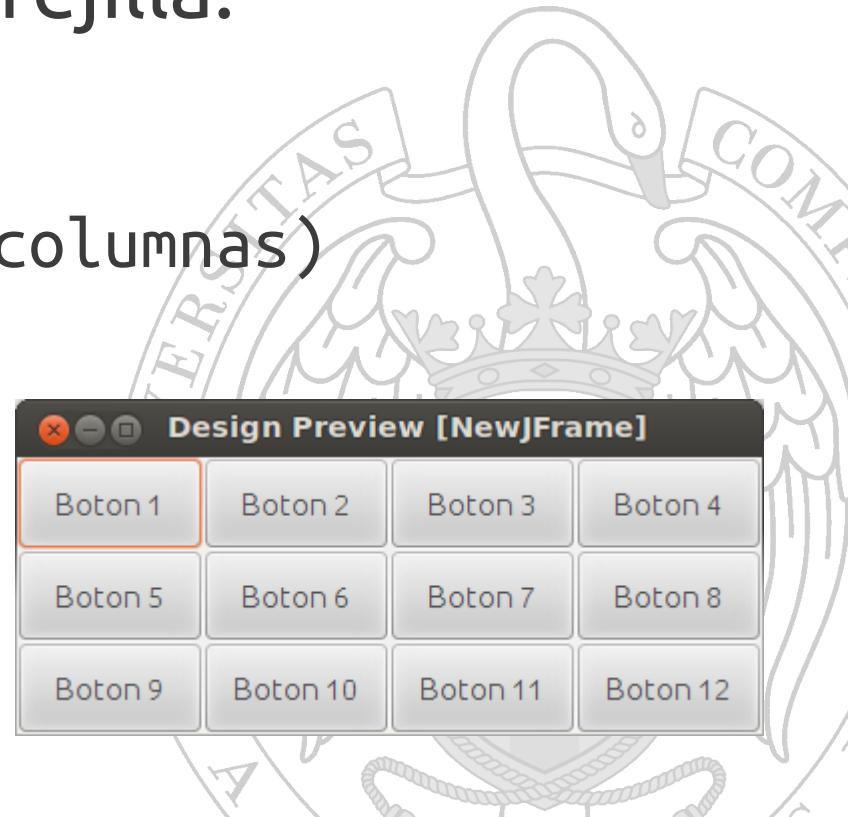
FlowLayout

- Coloca los elementos uno a continuación de otro, de manera similar a la colocación de palabras en un procesador de textos.
- Métodos:
 - `setAlignment(int alineacion)`
 - `setHgap(int separacion)`
 - `setVgap(int separacion)`



GridLayout

- Coloca los componentes de la interfaz en forma de rejilla.
- El orden en que se añadan los componentes determina su posición en la rejilla.
- Constructor:
 - `GridLayout(int filas, int columnas)`
- Métodos:
 - `setHgap(int separacion)`
 - `setVgap(int separacion)`

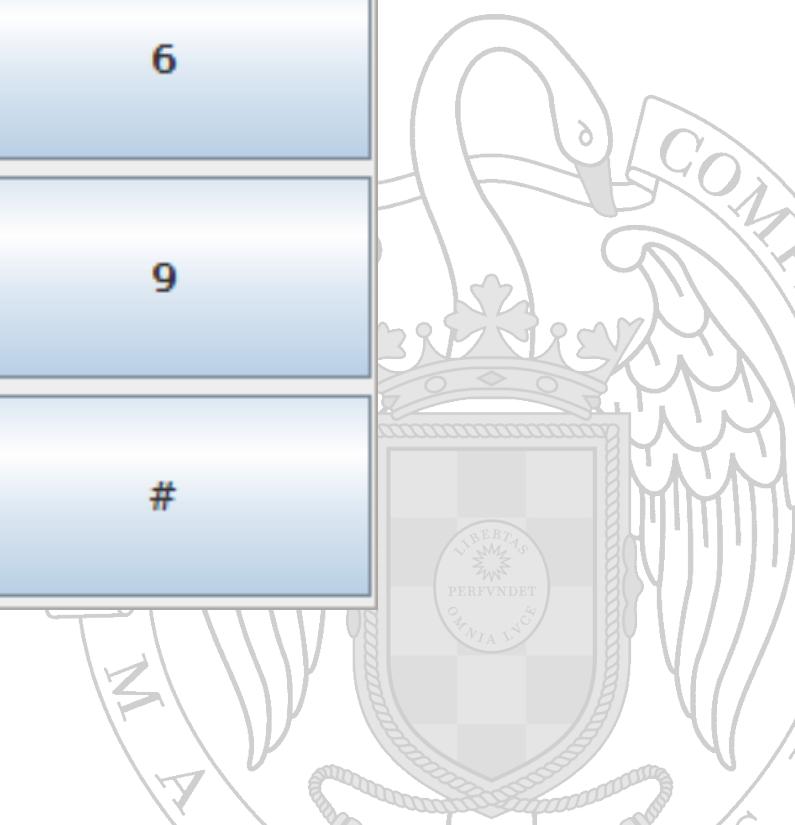


GridLayout

```
public class MiVentana2 extends JFrame {  
    public MiVentana2() {  
        super("Titulo de ventana");  
        setSize(400, 300);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        Container cp = getContentPane();  
        GridLayout gl = new GridLayout(4,3);  
        gl.setHgap(5); gl.setVgap(5);  
        cp.setLayout(gl);  
        for(int i = 1; i <= 9; i++) {  
            cp.add(new JButton(String.valueOf(i)));  
        }  
        cp.add(new JButton("*"));  
        cp.add(new JButton("0"));  
        cp.add(new JButton("#"));  
    }  
}
```



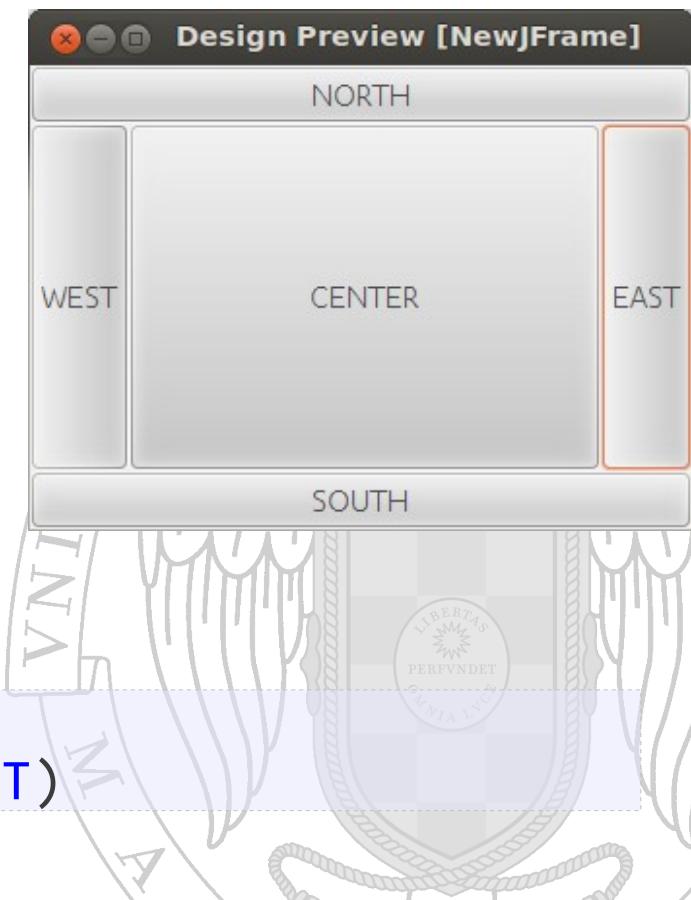
GridLayout



BorderLayout

- Coloca y cambia de tamaño sus componentes para que se ajusten a los bordes y parte central de la ventana.
- Métodos:
 - `setHgap(int separacion)`
 - `setVgap(int separacion)`
- Al añadir un elemento a la ventana, hay que especificar su colocación:

```
JButton b = new JButton(...);  
getContentPane().add(b, BorderLayout.EAST)
```

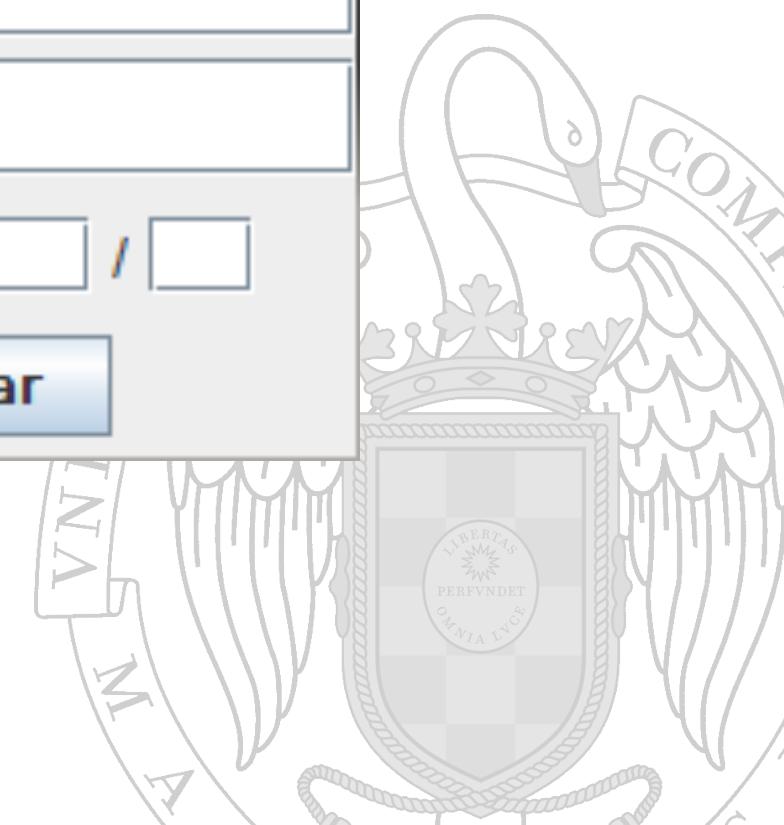
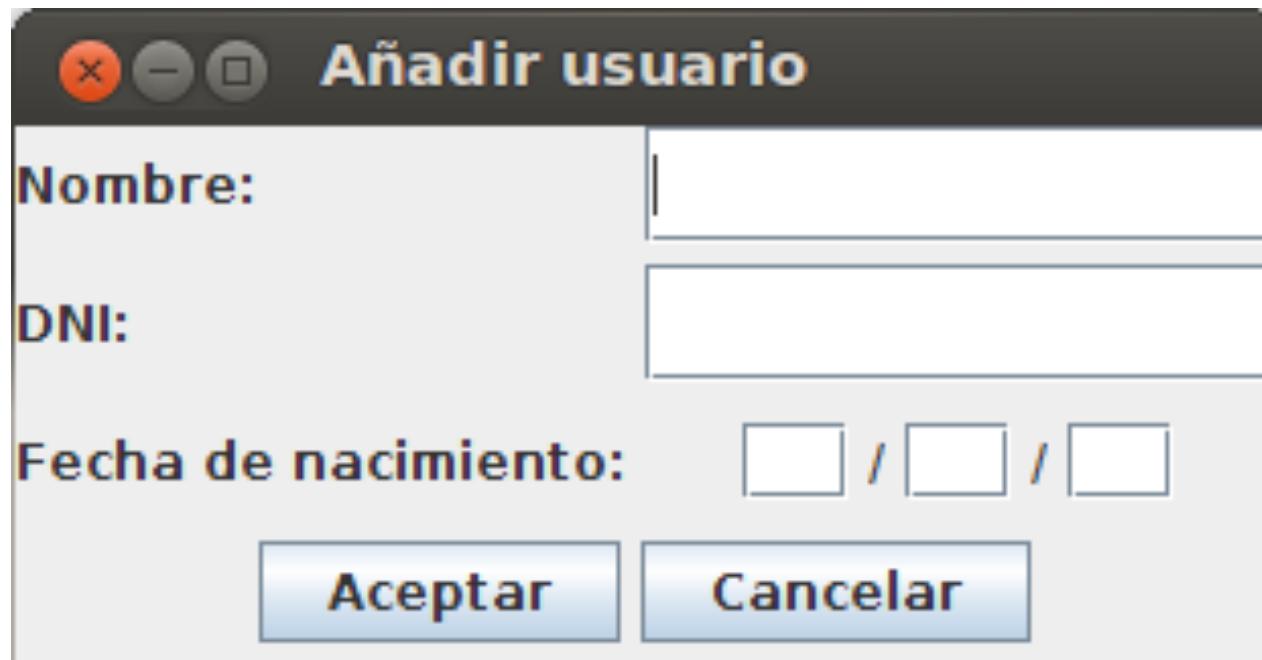


Interfaces complejas: JPanel

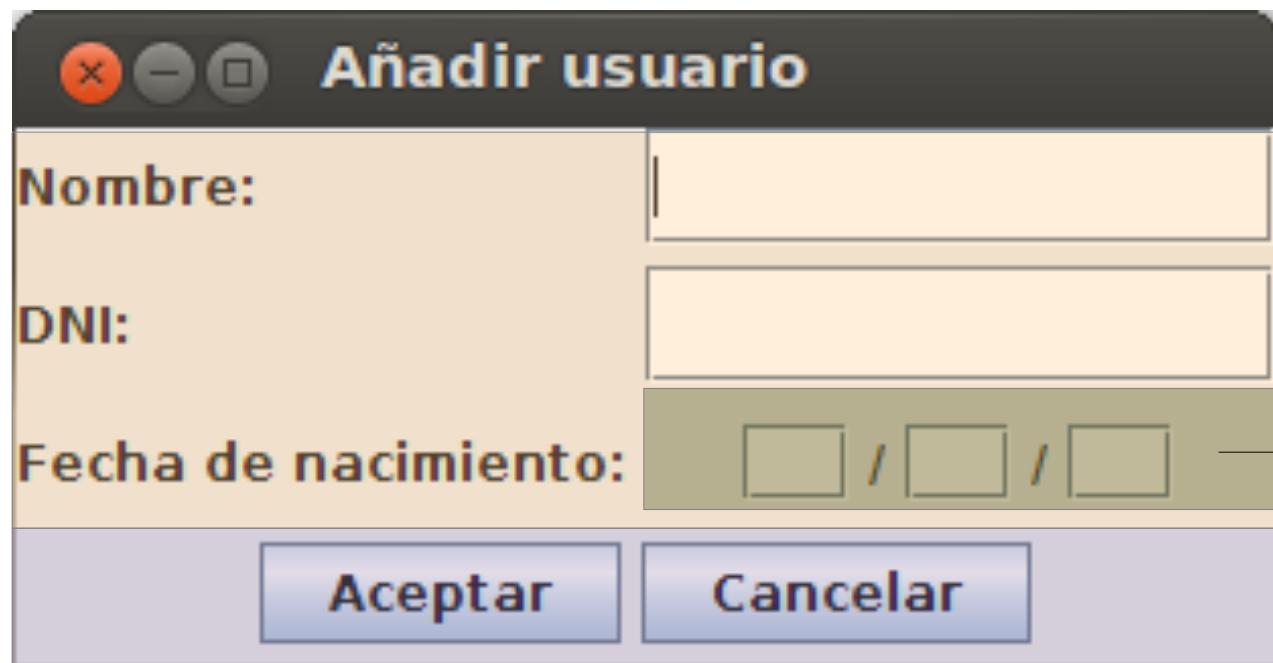
- Un panel es un componente con un *layout manager* propio, y que puede contener varios componentes en su interior.
- Constructor:
 - JPanel()
- Métodos:
 - void `setLayout(LayoutManager lm)`
 - void `add(JComponent componente)`
 - ...



Interfaces complejas: JPanel



Interfaces complejas: JPanel



→ GridLayout

→ FlowLayout

→ FlowLayout

Interfaces complejas: JPanel

```
public MiVentana3() {  
    super("Añadir usuario");  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    // Panel de fecha  
    JPanel panelFecha = new JPanel();  
    panelFecha.setLayout(new FlowLayout());  
    panelFecha.add(new JTextField(2));  
    panelFecha.add(new JLabel("/"));  
    panelFecha.add(new JTextField(2));  
    panelFecha.add(new JLabel("/"));  
    panelFecha.add(new JTextField(2));  
    // Panel de datos  
    JPanel panelDatos = new JPanel();  
    GridLayout gl = new GridLayout(3,2,0,5);  
    panelDatos.setLayout(gl);  
    panelDatos.add(new JLabel("Nombre:"));  
    panelDatos.add(new JTextField(10));  
    panelDatos.add(new JLabel("DNI:"));  
    panelDatos.add(new JTextField(10));  
    panelDatos.add(new JLabel("Fecha de nacimiento: "));  
    panelDatos.add(panelFecha);  
    ...  
}
```



Interfaces complejas: JPanel

```
...
// Panel de botones
JPanel panelBotones = new JPanel();
panelBotones.setLayout(new FlowLayout());
panelBotones.add(new JButton("Aceptar"));
panelBotones.add(new JButton("Cancelar"));

Container cp = getContentPane();
cp.add(panelDatos, BorderLayout.CENTER);
cp.add(panelBotones, BorderLayout.SOUTH);
}
```



Interfaces complejas: GridBagLayout

- Más flexible que GridLayout
- Cada componente ha de tener asociado un objeto de la clase GridBagConstraints. La asociación se producirá en el método add.



```
JButton b = new JButton("Aceptar");
GridBagConstraints gbc = new GridBagConstraints(...);
getContentPane().add(b, gbc);
```

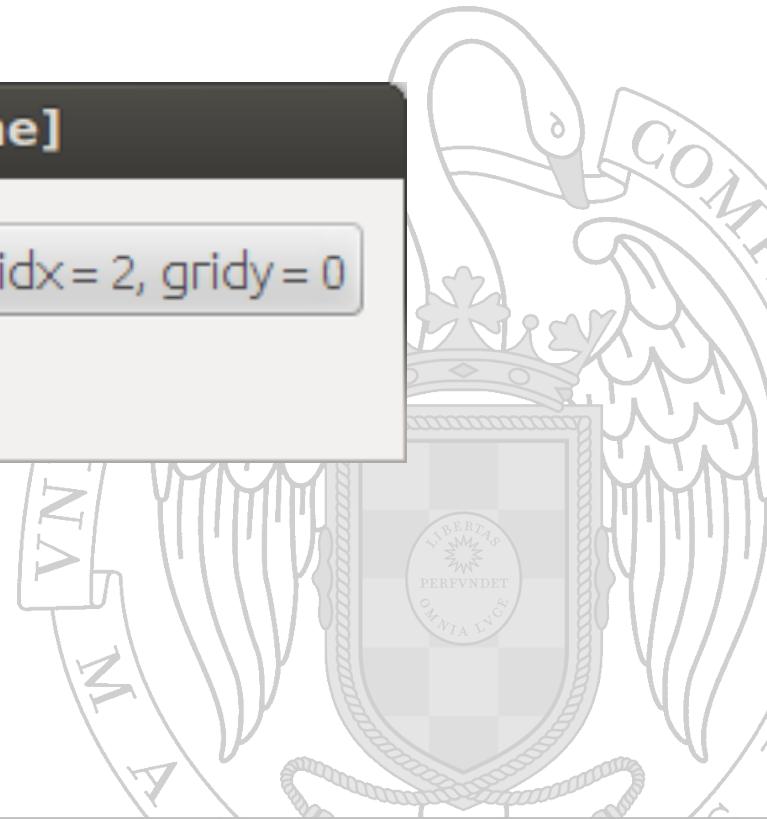
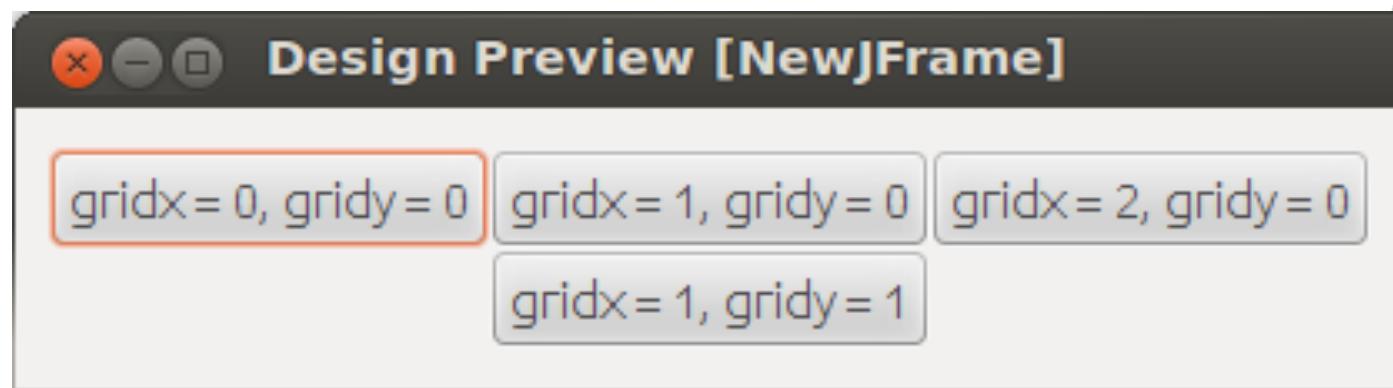
GridBagConstraints

- Atributos públicos:
 - int.gridx,.gridy
 - int.gridwidth, gridheight
 - double weightx, weighty
 - int.fill
 - int.anchor
 - Insets insets
 - int.ipadx, ipady
- Pueden ser inicializados en el constructor.



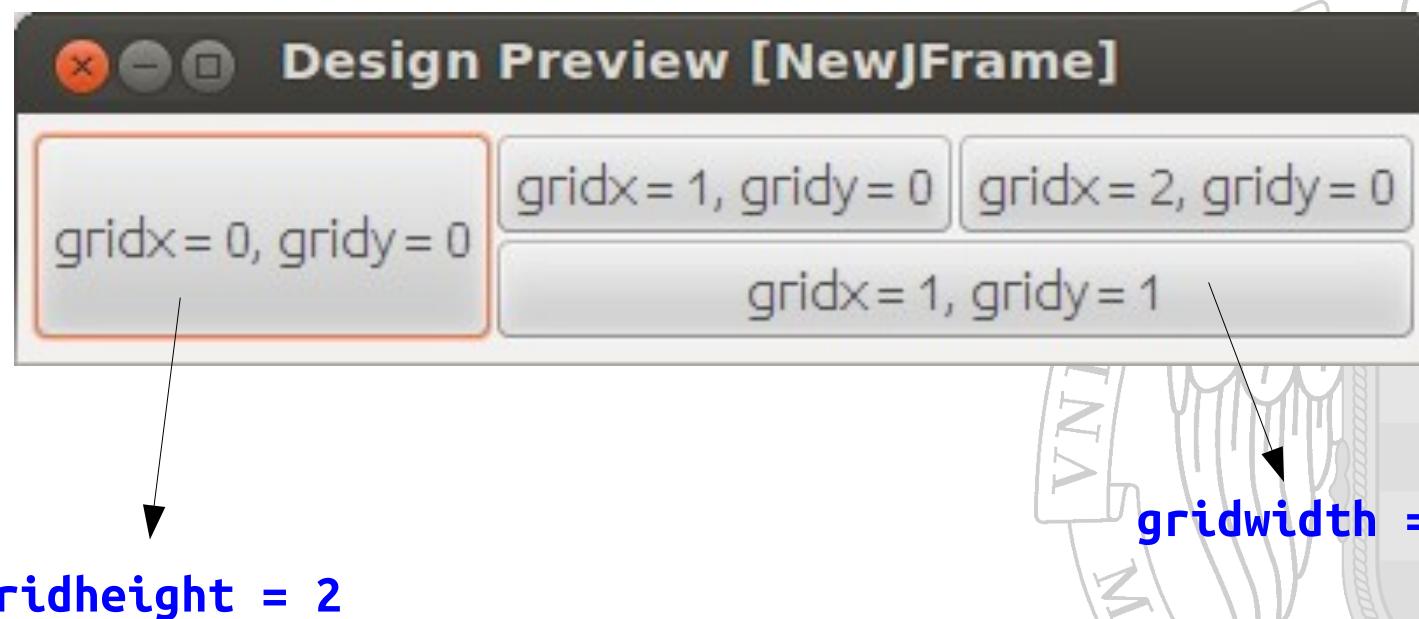
GridBagConstraints

- Atributos públicos:
 - int.gridx,.gridy



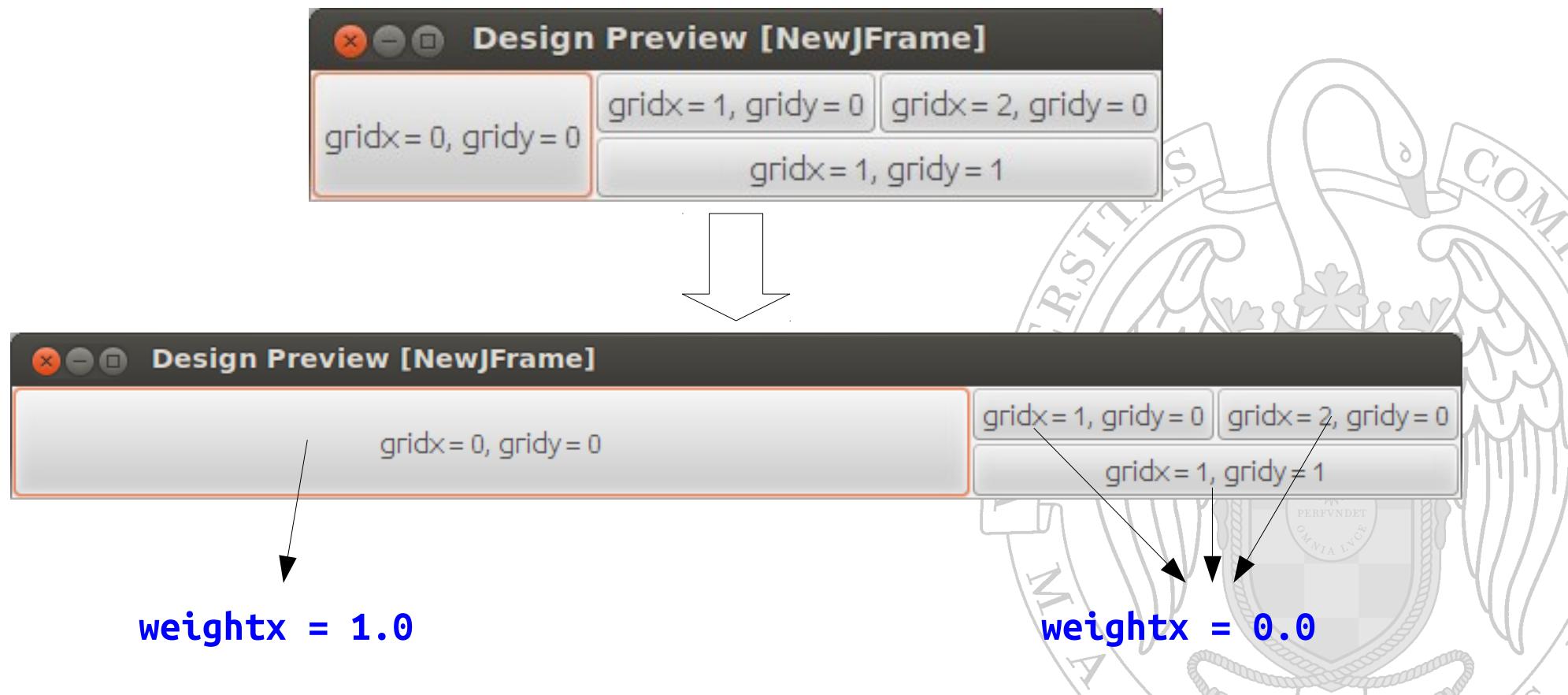
GridBagConstraints

- Atributos públicos:
 - int gridwidth, gridheight



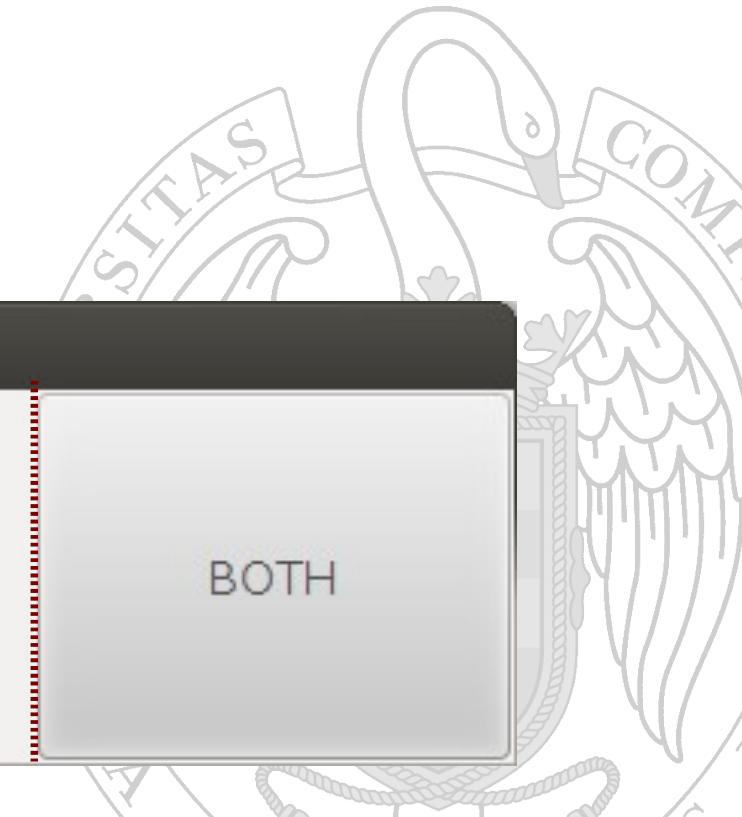
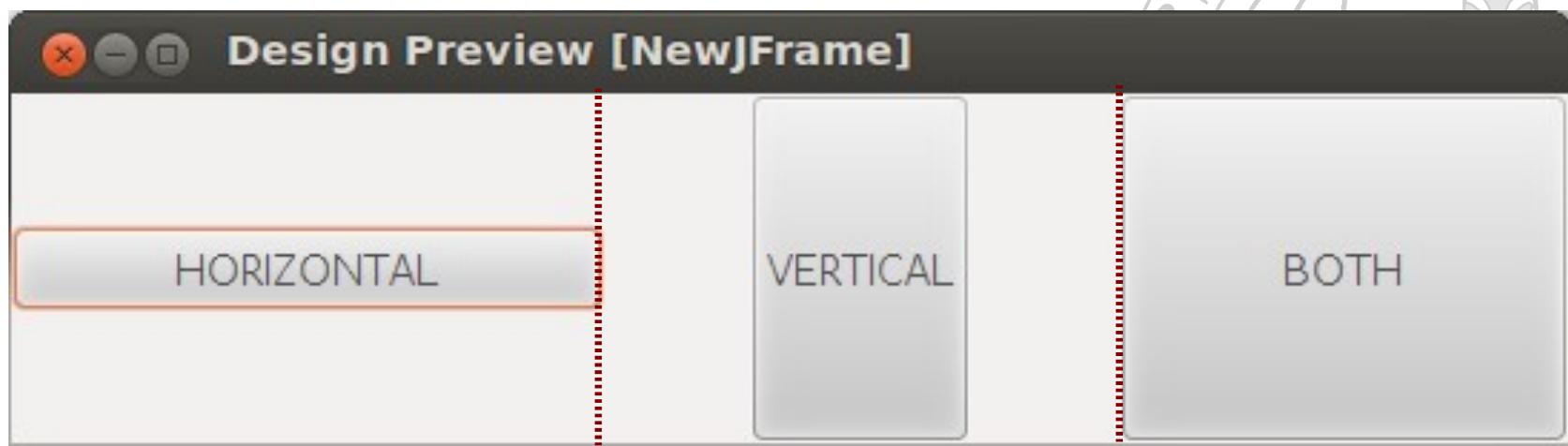
GridBagConstraints

- Atributos públicos:
 - double weightx, weighty



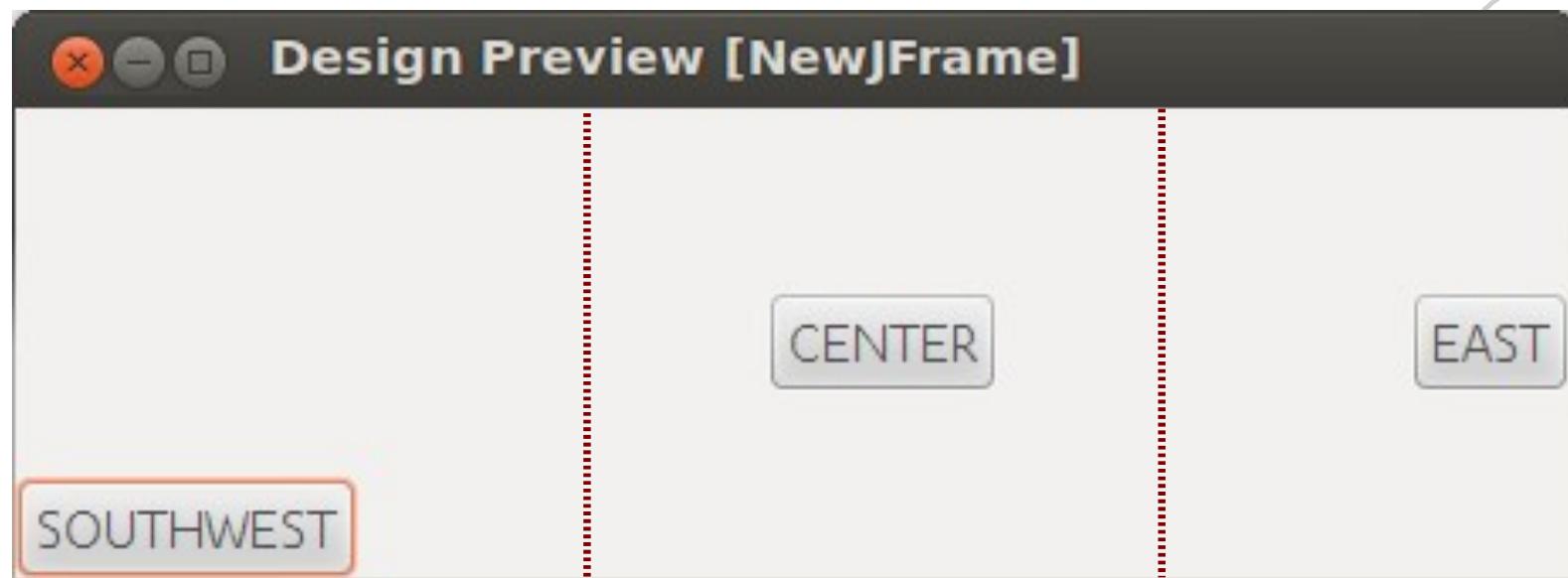
GridBagConstraints

- Atributos públicos:
 - int fill
- Puede ser:
 - GridBagConstraints.HORIZONTAL
 - GridBagConstraints.VERTICAL
 - GridBagConstraints.BOTH



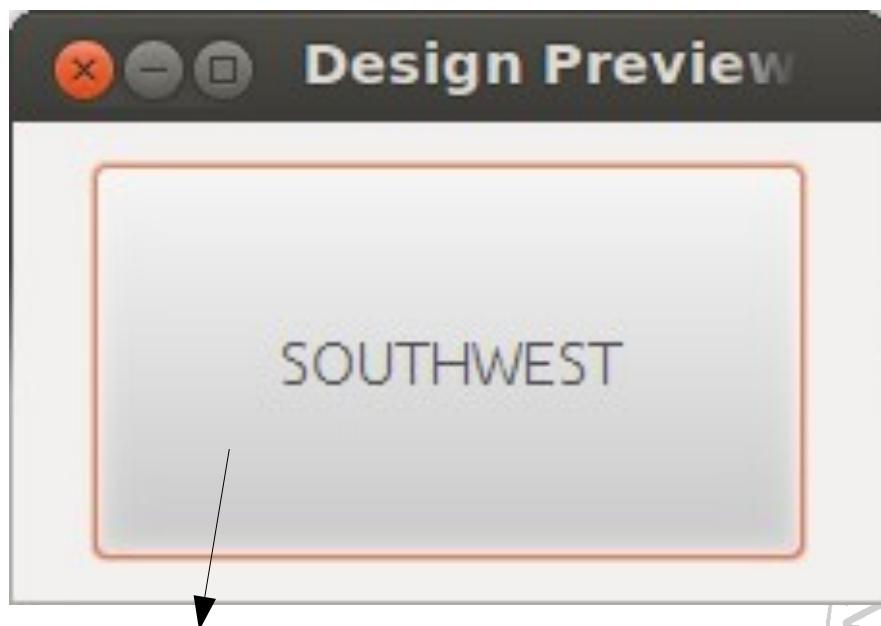
GridBagConstraints

- Atributos públicos:
 - int anchor

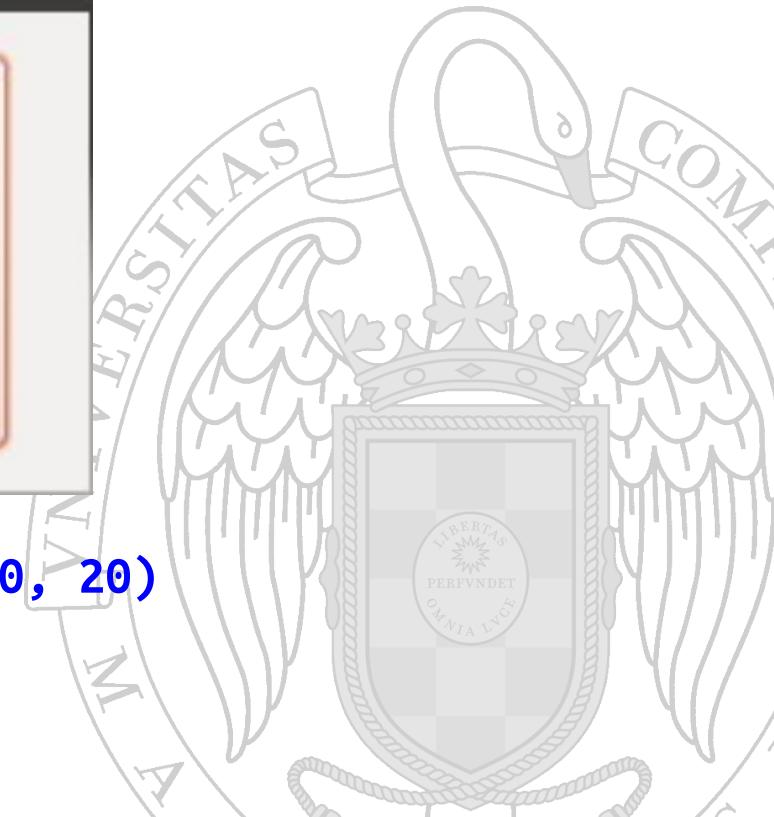


GridBagConstraints

- Atributos públicos:
 - Insets insets



```
insets = new Insets(10, 20, 10, 20)
```



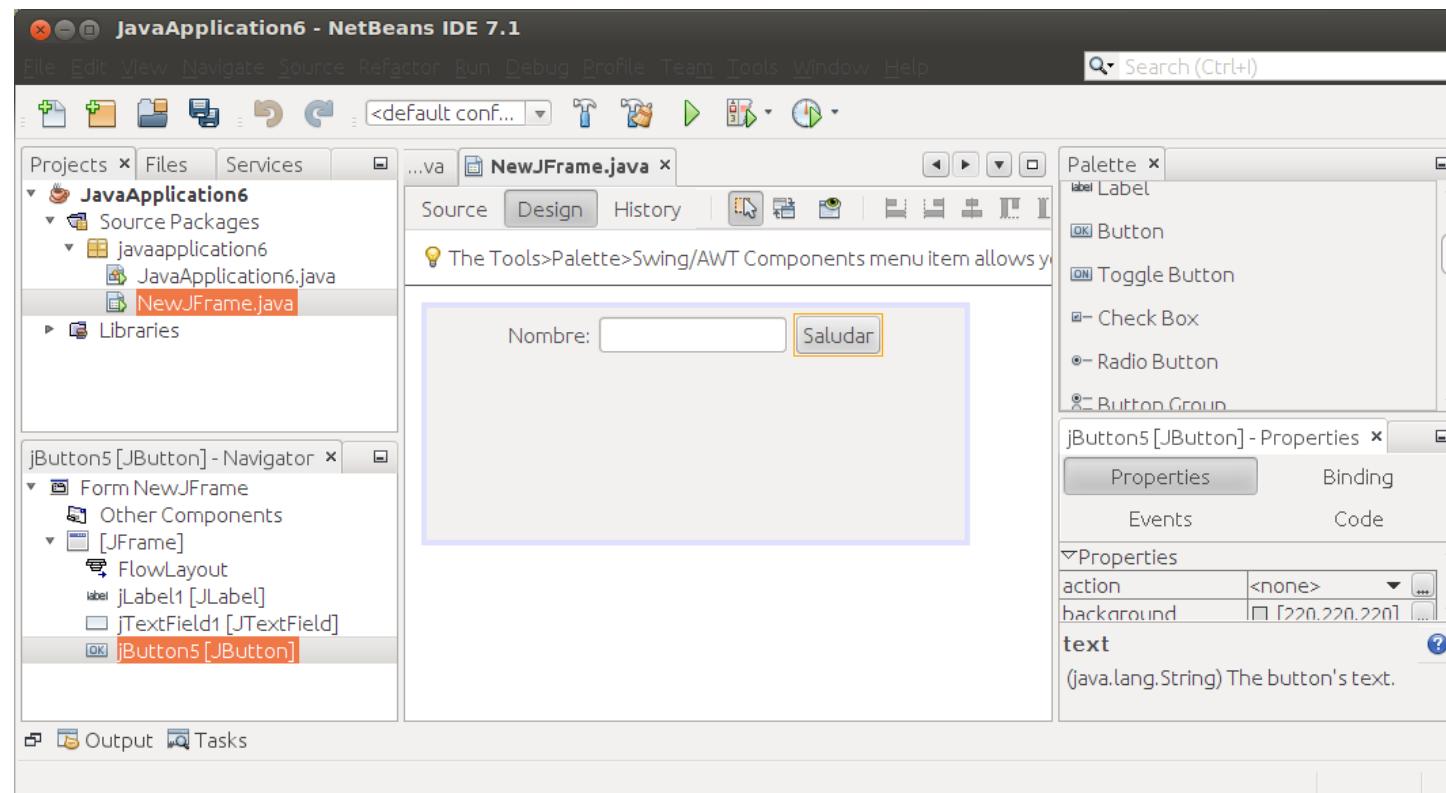
GridBagConstraints

- Atributos públicos:
 - int padx, int pady
- Especifican cuánto espacio añadir a la anchura/altura mínima de los componentes.



Constructores de interfaces

- Permiten construir interfaces de usuario interactivamente.
- Ejemplo: *Netbeans* (netbeans.org)



Contenidos

- Ventanas
- Componentes
- Layout Managers
- Manejo de eventos
- Cuadros de diálogo predefinidos
- Dibujo de gráficos
- Arquitectura MVC



Manejo de eventos

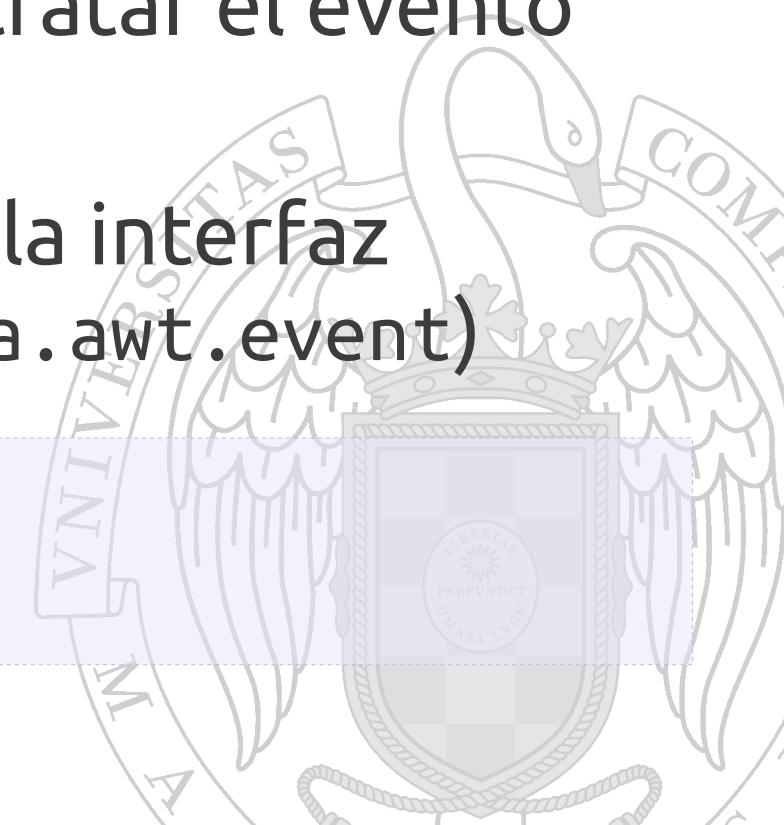
- Un evento es un suceso que ocurre como consecuencia de la interacción del usuario con la interfaz gráfica.
 - Pulsación de un botón.
 - Cambio del contenido en un cuadro de texto.
 - Deslizamiento de una barra.
 - Activación de un JCheckBox.
 - Movimiento de la ventana.



Pulsación de un botón

- La clase JButton tiene un método:
 - void addActionListener(ActionListener l)
- Que especifica el objeto (manejador de evento) que se encargará de tratar el evento de pulsación del botón.
- Este objeto ha de interpretar la interfaz ActionListener (paquete java.awt.event)

```
public interface ActionListener {  
    void actionPerformed(ActionEvent e)  
}
```



Pulsación de un botón

- Cuando el usuario pulse el botón, se llamará al método actionPerformed de todos los manejadores de eventos que se hayan registrado.

```
public class Manejador implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        ...  
    }  
}
```

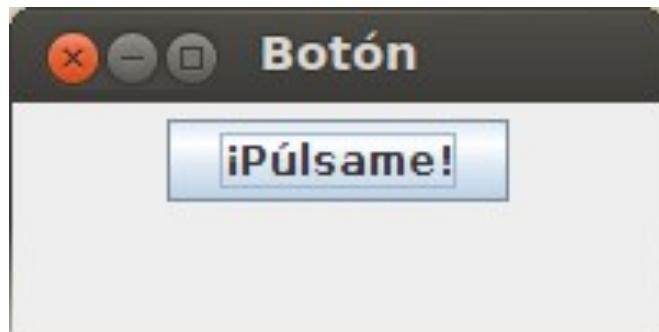
↑
**Información sobre
el evento**

- Métodos de ActionEvent:
 - `public Object getSource()`
 - `public int getModifiers()`



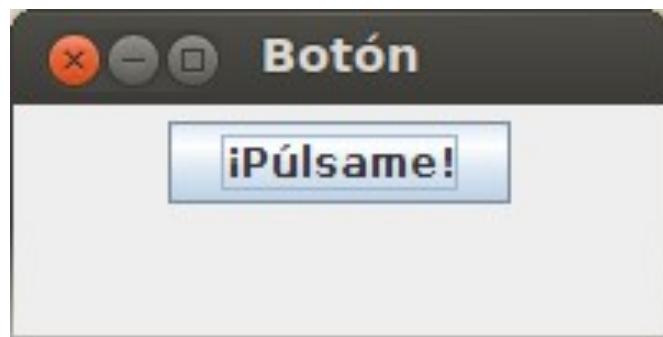
Ejemplo

```
public class BotonVentana extends JFrame {  
    public BotonVentana() {  
        super("Botón");  
        setSize(200,100);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        Container cp = getContentPane();  
        cp.setLayout(new FlowLayout());  
        JButton boton = new JButton("¡Púlsame!");  
        boton.addActionListener(new EventoBotonPulsado());  
        cp.add(boton);  
    }  
}
```



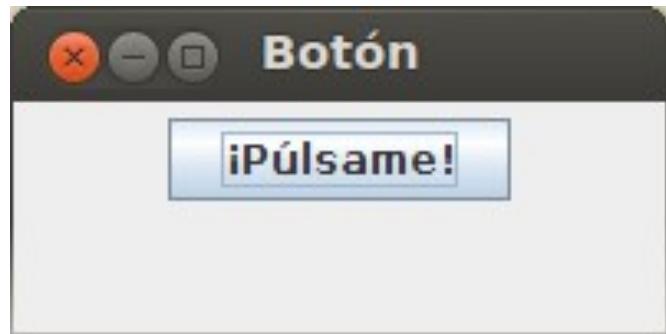
Ejemplo

```
public class EventoBotonPulsado implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("¡Gracias!");  
    }  
}
```

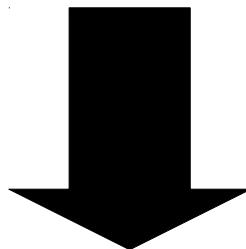
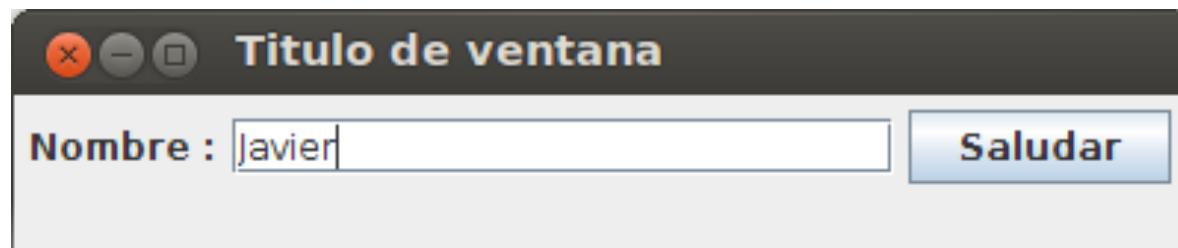


Ejemplo

```
public class EventoBotonPulsado implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        JButton boton = (JButton) e.getSource();  
        boton.setText("¡Gracias!");  
    }  
}
```

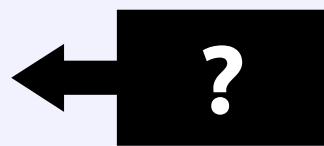


Acceso a componentes de la interfaz



Acceso a componentes de la interfaz

```
public class EventoSaludo implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        }  
    }
```



- ¿Cómo accedo al objeto **JTextField**?



Posibilidad 1

```
public class EventoSaludo implements ActionListener {  
    private JTextField cuadroTexto;  
  
    public EventoSaludo(JTextField cuadroTexto) {  
        this.cuadroTexto = cuadroTexto;  
    }  
  
    public void actionPerformed(ActionEvent e) {  
        JOptionPane.showMessageDialog(null, "¡Hola, " +  
            cuadroTexto.getText() + "!");  
    }  
}
```



Posibilidad 1

```
public class MiVentana extends JFrame {  
  
    public MiVentana() {  
        super("Titulo de ventana");  
        setSize(400, 300);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(new FlowLayout());  
        Container cp = getContentPane();  
        cp.add(new JLabel("Nombre :"));  
        JTextField texto = new JTextField(20);  
        cp.add(texto);  
        JButton botonSaludo = new JButton("Saludar");  
        cp.add(botonSaludo);  
        botonSaludo.addActionListener(new EventoSaludo(texto));  
    }  
}
```

Posibilidad 2

```
public class MiVentana extends JFrame {  
    private JTextField cuadroTexto;  
  
    class EventoSaludo implements ActionListener {  
        public void actionPerformed(ActionEvent e) {  
            JOptionPane.showMessageDialog(null, "¡Hola, " +  
                cuadroTexto.getText() + "!");  
        }  
    }  
  
    public MiVentana() {  
        ...  
        cuadroTexto = new JTextField(20);  
        cp.add(cuadroTexto);  
        ...  
        botonSaludo.addActionListener(new EventoSaludo());  
    }  
}
```



Posibilidad 3

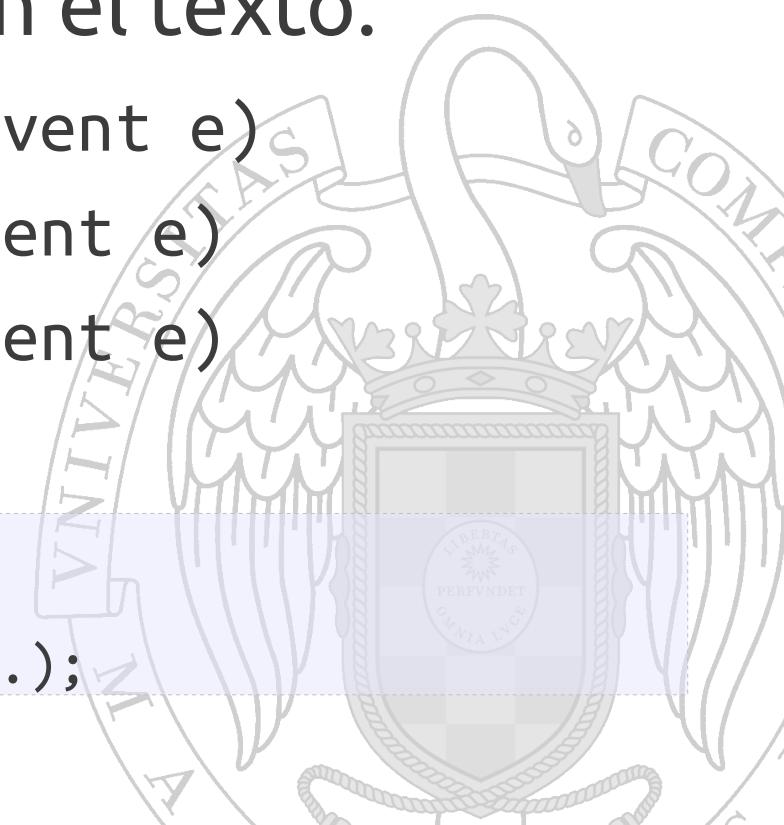
```
public class MiVentana extends JFrame implements ActionListener {  
    private JTextField cuadroTexto;  
  
    public void actionPerformed(ActionEvent e) {  
        JOptionPane.showMessageDialog(null, "¡Hola, " +  
            cuadroTexto.getText() + "!");  
    }  
  
    public MiVentana() {  
        ...  
        cuadroTexto = new JTextField(20);  
        cp.add(cuadroTexto);  
        ...  
        botonSaludo.addActionListener(this);  
    }  
}
```



Eventos en un JTextField

- **CaretListener:** Cambios en la posición del CURSOR.
 - void `caretUpdate(CaretEvent e)`
- **DocumentListener:** Cambios en el texto.
 - void `changedUpdate(DocumentEvent e)`
 - void `insertUpdate(DocumentEvent e)`
 - void `removeUpdate(DocumentEvent e)`

```
JTextField text = ...;  
text.addCaretListener(...);  
text.getDocument().addDocumentListener(...);
```



Eventos en una ventana

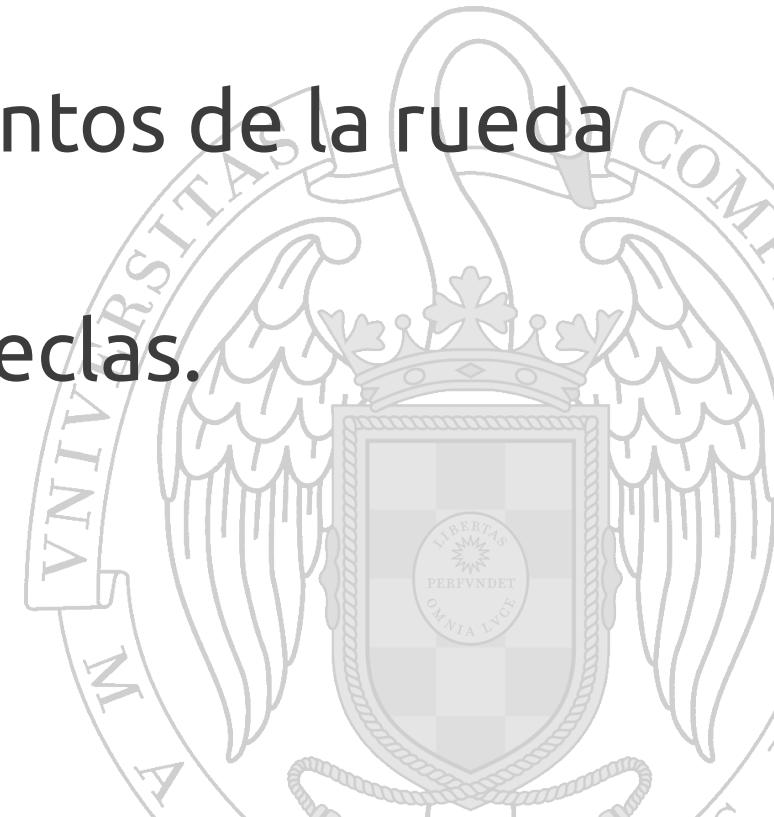
- **WindowListener**

- void **windowActivated**(WindowEvent e)
- void **windowClosed**(WindowEvent e)
- void **windowClosing**(WindowEvent e)
- void **windowDeactivated**(WindowEvent e)
- void **windowDeiconified**(WindowEvent e)
- void **windowIconified**(WindowEvent e)
- void **windowOpened**(WindowEvent e)



Eventos de ratón y de teclado

- **MouseListener**: Pulsaciones de botón, entradas y salidas del puntero en un componente.
- **MouseMotionListener**: Movimientos del ratón dentro de un componente.
- **MouseWheelListener**: Movimientos de la rueda central de un ratón.
- **KeyListener**: Pulsaciones de teclas.



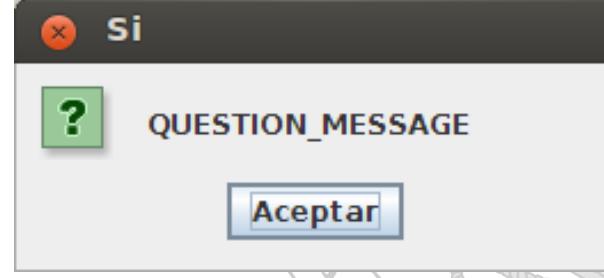
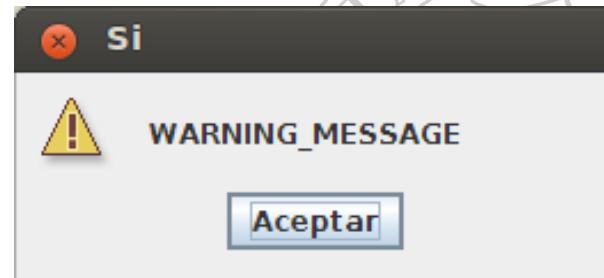
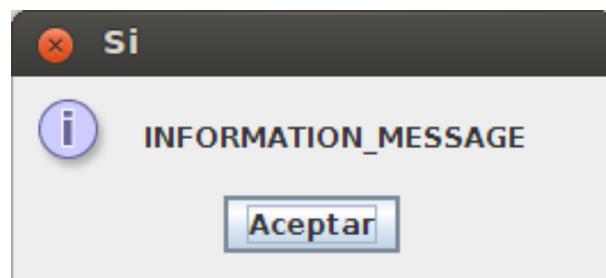
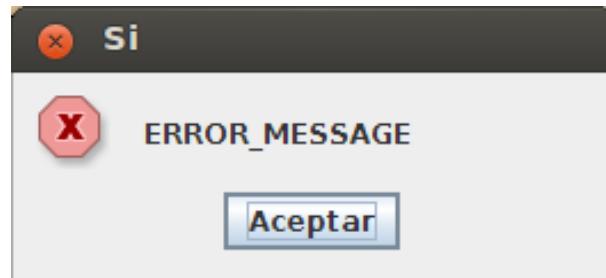
Contenidos

- Ventanas
- Componentes
- Layout Managers
- Manejo de eventos
- Cuadros de diálogo predefinidos
- Dibujo de gráficos
- Arquitectura MVC



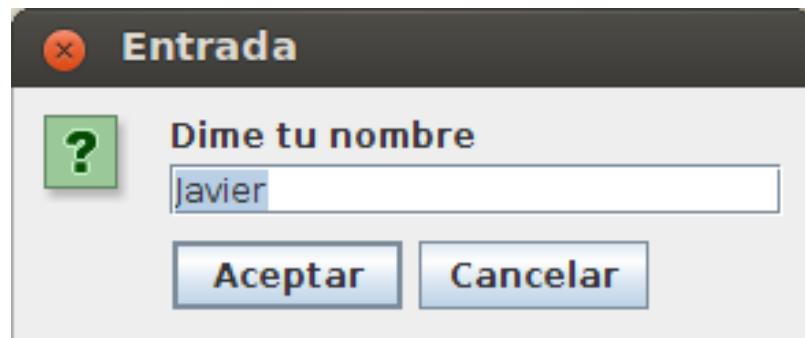
Cuadros de diálogo predefinidos

- La clase JOptionPane proporciona métodos de utilidad para mostrar ventanas de aviso y de confirmación estándar.
 - `void showMessageDialog(Component padre, Object mensaje, String tituloVentana, int tipoMensaje)`



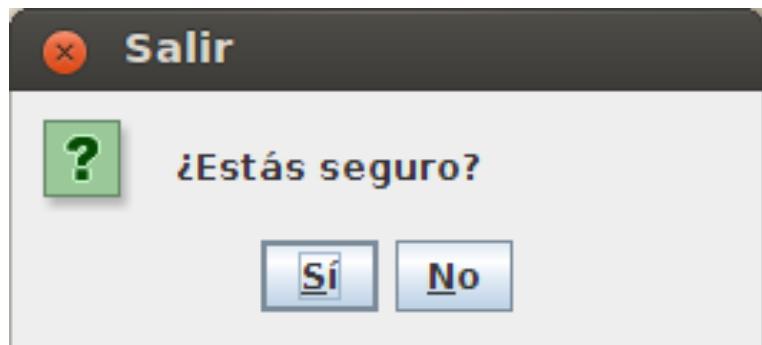
Cuadros de diálogo predefinidos

- La clase JOptionPane proporciona métodos de utilidad para mostrar ventanas de aviso y de confirmación estándar.
 - `String showInputDialog(Component padre, Object mensaje, Object valorDefecto)`



Cuadros de diálogo predefinidos

- La clase JOptionPane proporciona métodos de utilidad para mostrar ventanas de aviso y de confirmación estándar.
 - `int showConfirmDialog(Component padre, Object mensaje, String titulo, int tipoOpciones, int tipoMensaje)`



Contenidos

- Ventanas
- Componentes
- Layout Managers
- Manejo de eventos
- Cuadros de diálogo predefinidos
- Dibujo de gráficos
- Arquitectura MVC



Dibujar gráficos

- Cada componente tiene un método llamado `paintComponent`, que se encarga de pintarlo en pantalla.
- Para realizar un dibujo definido por el programador, basta con heredar de un componente (normalmente un JPanel), y sobreescribir su método `paintComponent`.
 - `void paintComponent(Graphics g)`



Dibujar gráficos

- Métodos de Graphics:

- void **drawPolygon**(int[] x, int[] y, int puntos)
- void **drawRect**(int x, int y, int ancho, int alto)
- void **fillRect**(int x, int y, int ancho, int alto)
- void **drawOval**(int x, int y, int ancho, int alto)
- void **fillOval**(int x, int y, int ancho, int alto)
- void **drawString**(String cad, int x, int y)
- void **setColor**(Color c)
- void **setFont**(Font f)



Ejemplo

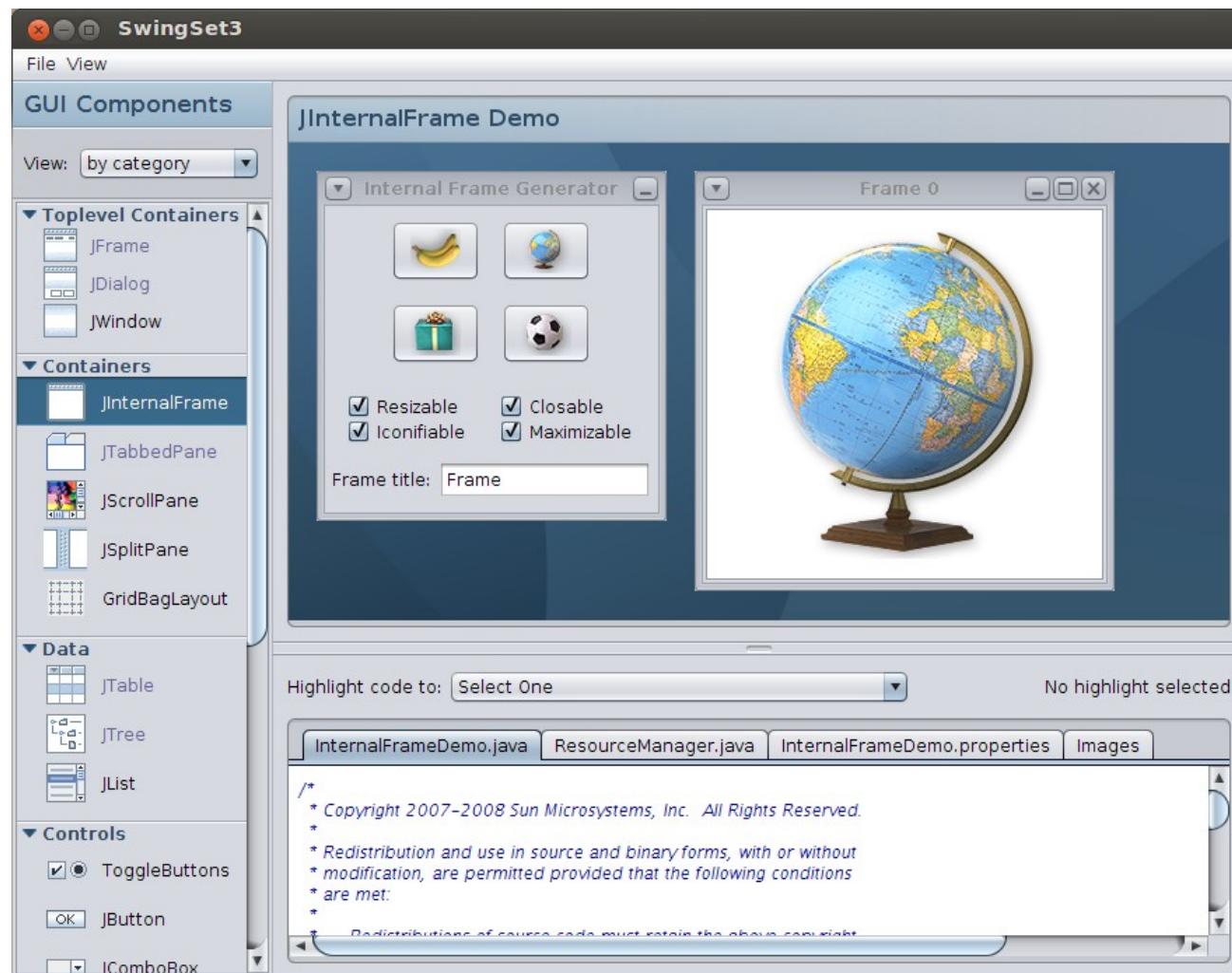
```
public class PanelSol extends JPanel {  
    public void paintComponent(Graphics g) {  
        g.setColor(Color.ORANGE);  
        g.fillOval(100,100,200,200);  
        for (double d = 0; d < 2*Math.PI; d += 0.1) {  
            int xEnd = (int) (200+150*Math.cos(d));  
            int yEnd = (int) (200+150*Math.sin(d));  
            g.drawLine(200, 200, xEnd, yEnd);  
        }  
        g.setColor(Color.BLACK);  
        g.drawArc(150, 150, 100, 100, 230, 80);  
        g.fillOval(150, 150, 20, 20);  
        g.fillOval(230, 150, 20, 20);  
    }  
}
```

Ejemplo

```
public class VentanaSol extends JFrame {  
    public VentanaSol() {  
        ...  
        Container cp = getContentPane();  
        cp.add(new PanelSol());  
    }  
}
```

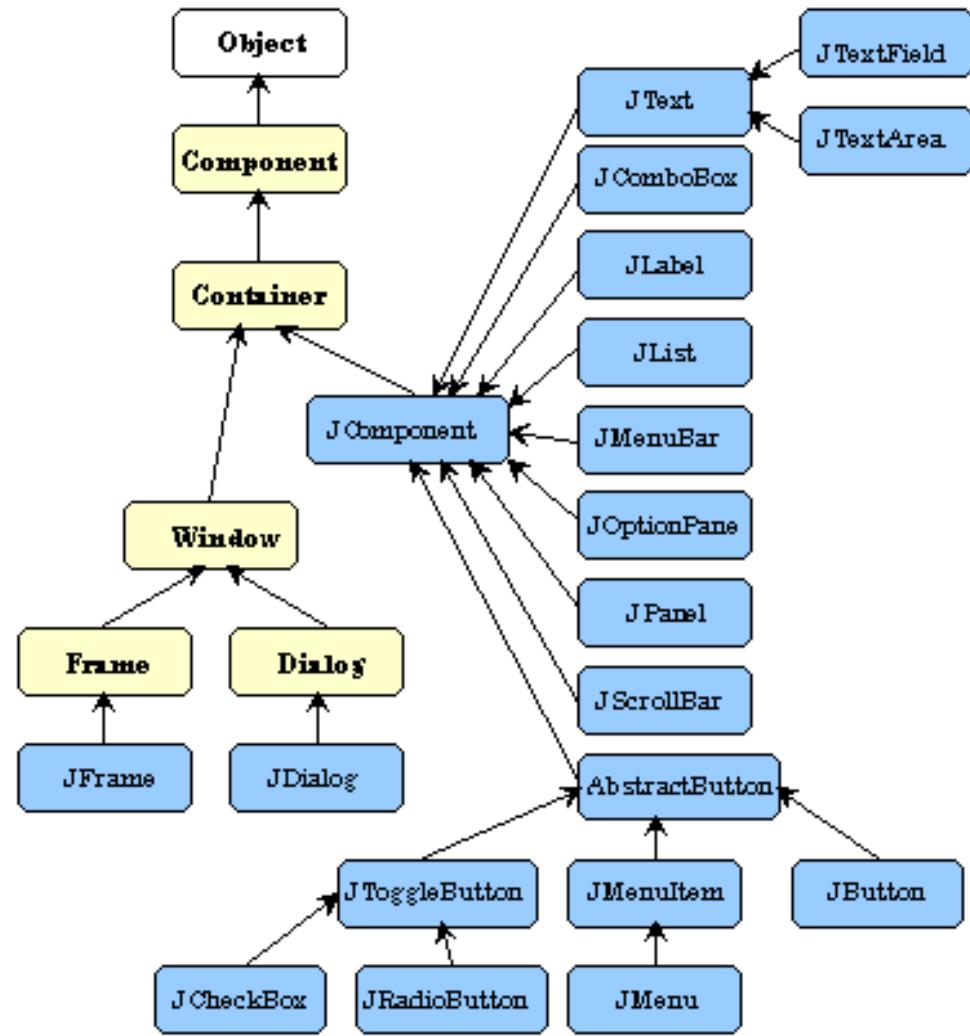


Aún hay más



<http://download.java.net/javadesktop/swingset3/SwingSet3.jnlp>

Aún hay más



<http://www.particle.kth.se/~fmi/kurs/PhysicsSimulation/Lectures/07A/swingDesign.html>

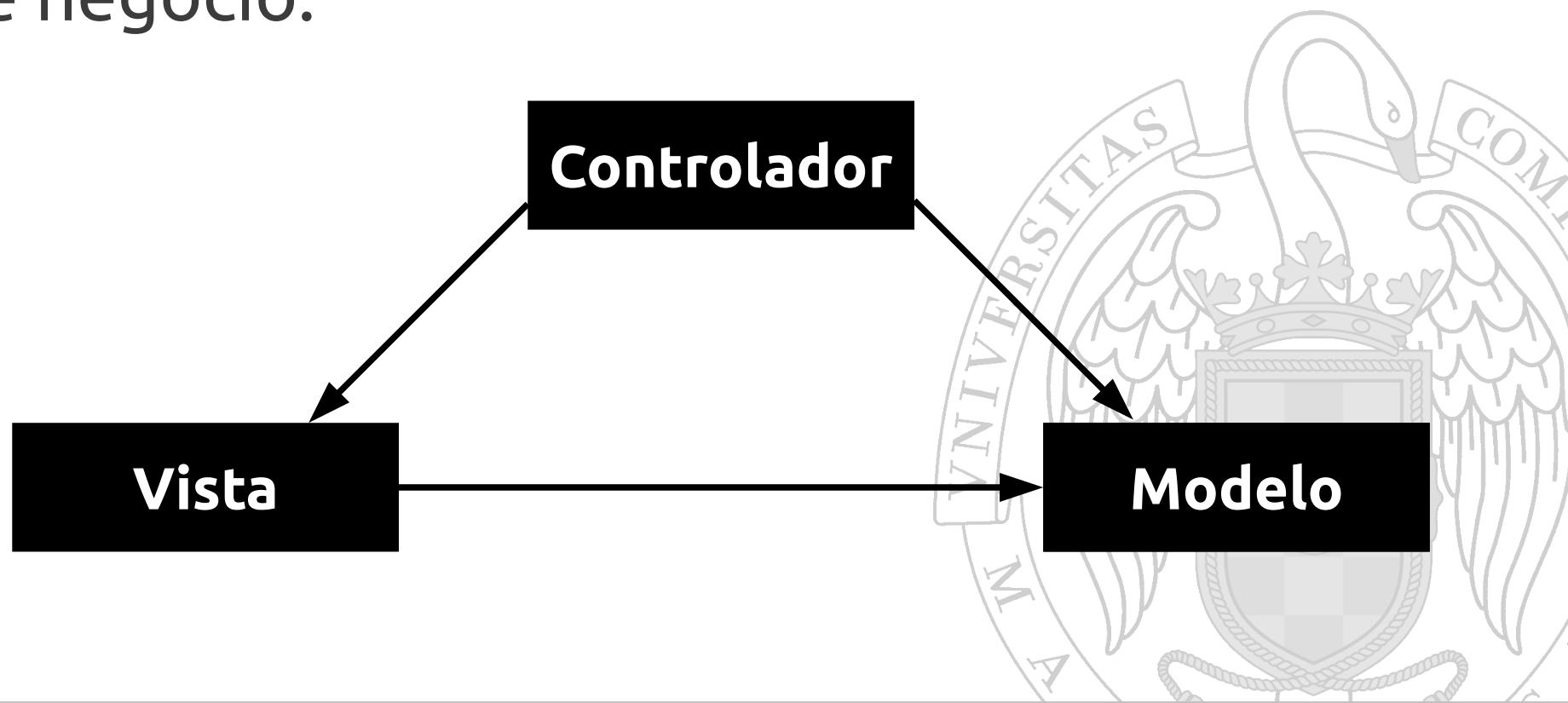
Contenidos

- Ventanas
- Componentes
- Layout Managers
- Manejo de eventos
- Cuadros de diálogo predefinidos
- Dibujo de gráficos
- Arquitectura MVC



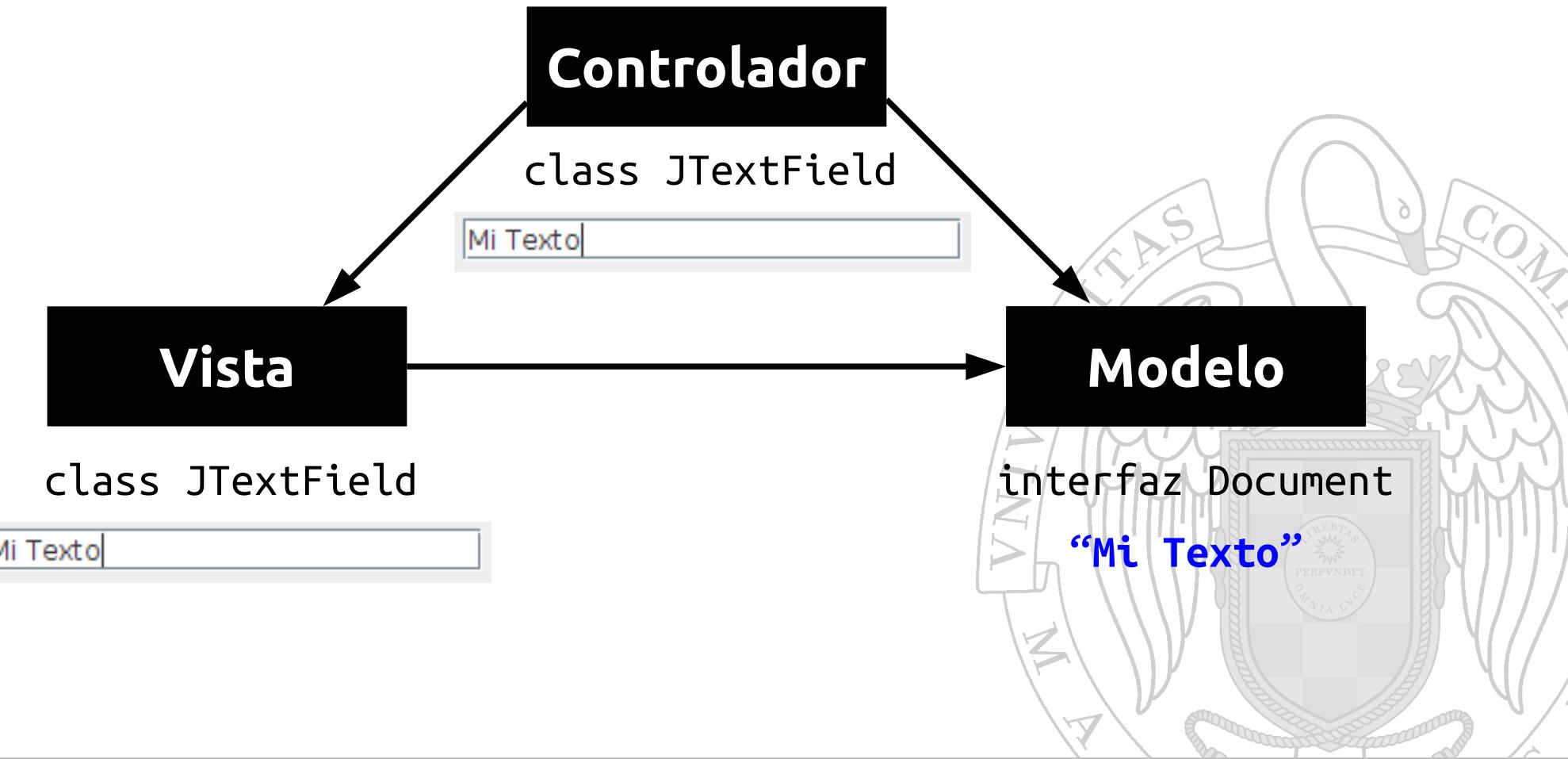
Arquitectura MVC

- MVC = Modelo – Vista – Controlador
- Es un patrón de diseño que separa los datos de la aplicación, la interfaz de usuario, y la lógica de negocio.



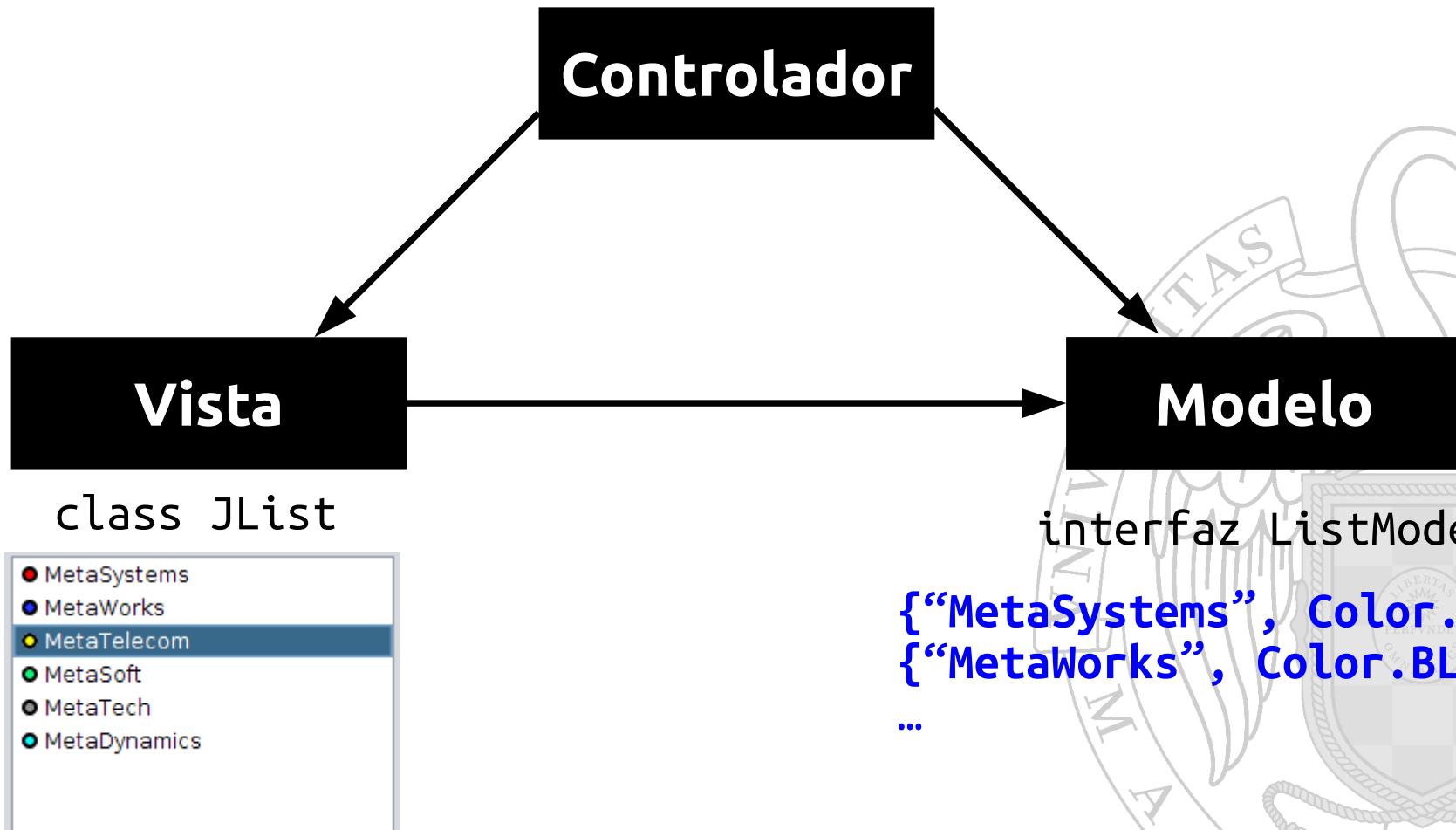
Arquitectura MVC

- Ejemplo: Edición de textos.



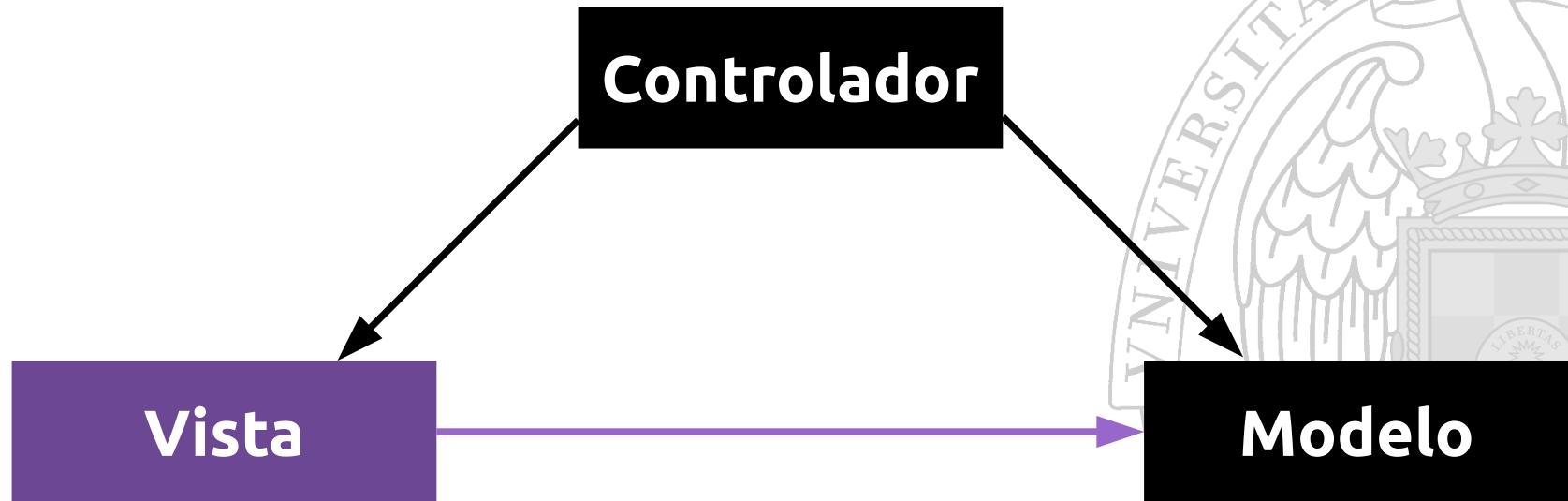
Arquitectura MVC

- Ejemplo: Listas



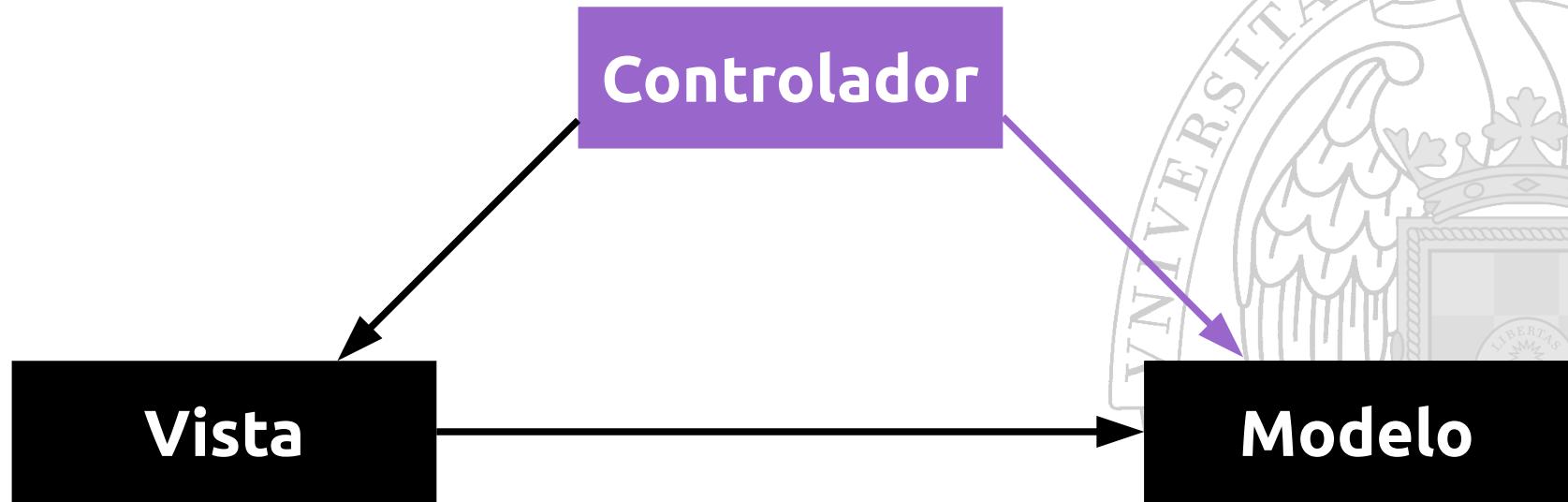
Arquitectura MVC

- **Preparación:** El modelo tiene una lista de vistas. Cada vez que haya un cambio de modelo, éste avisará a todas las vistas para que se actualicen.
- Cuando se crea una vista, ésta avisa al modelo para que la añada a su lista.



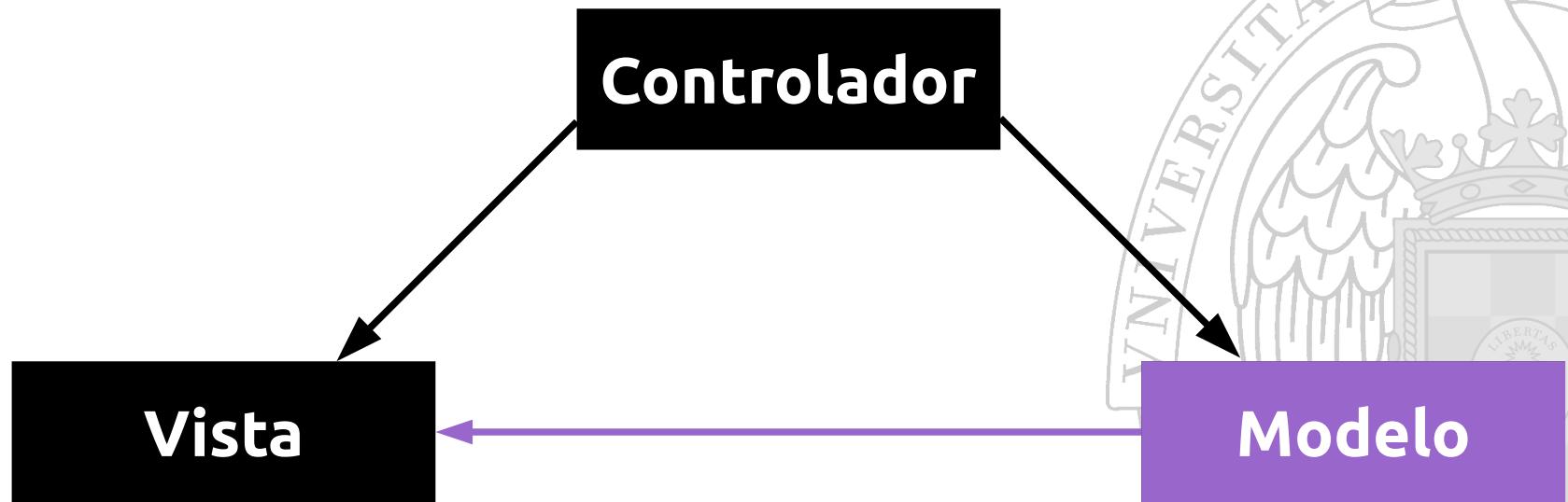
Arquitectura MVC

- **Paso 1:** Cuando el controlador recibe la acción del usuario, envía un mensaje al modelo para modificarlo.



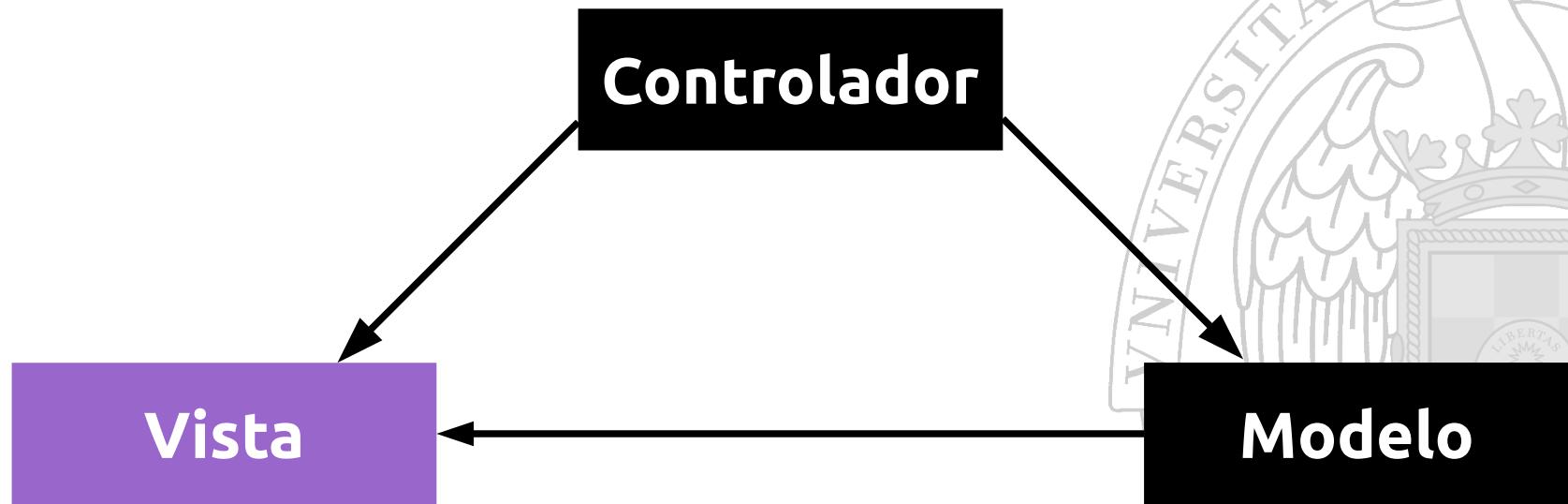
Arquitectura MVC

- **Paso 2:** Cuando el modelo se modifica, envía un mensaje a todas las vistas que tenga registradas.



Arquitectura MVC

- **Paso 3:** La vista se actualiza con los cambios realizados en el modelo



Referencias

- P. Deitel, H. Deitel
Java. How to Program (9th Edition)
Caps. 14, 15 y 25
- B. Eckel
Thinking in Java (3rd Edition)
Cap. 14
- J. Zukowski
The definitive guide to Java Swing

