
Tema 6. Programación Orientada a Eventos

- Introducción
- Ventanas. JFrame
- Distribución de paneles (layouts)
- Componentes
- Menu
- Diálogos
- Eventos
- WindowBuilder: Plugin en Eclipse para GUI

Introducción a la programación orientada a eventos

- En función del tipo de interacción con el usuario, los programas se clasifican en:
 - ❖ **Secuenciales:** el programador define cuál va a ser el flujo del programa
 - ❖ **Interactivos:** exigen la intervención del usuario en tiempo de ejecución, pero el programa realiza acciones independientemente de las órdenes del usuario
 - ❖ **Dirigidos por eventos:** el programa “espera” la llegada de un evento (orden del usuario, ...), cuando el evento se produce realiza la acción correspondiente y vuelve a esperar

Introducción a la programación orientada a eventos

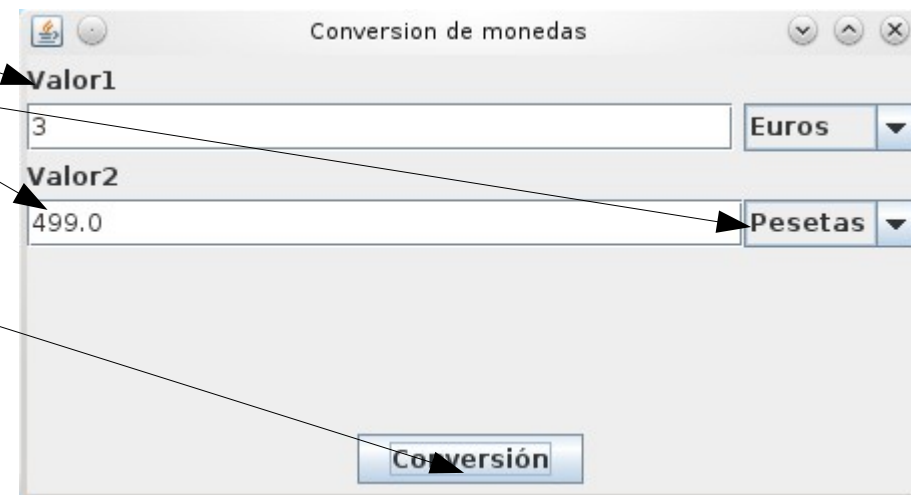
- Programación orientada a eventos se encuentra basada en la captura de todos los eventos producidos por el sistema operativo a través de mensajes y la ejecución de una acción motivada por dicho evento.
- Un ejemplo de programación orientada a eventos es la programación visual o la programación en tiempo real.
 - ❖ Por ejemplo cuando se mueve el ratón se produce un evento y acción es el movimiento del puntero.
 - ❖ Por ejemplo si se detecta un nivel alto en un sistema, se realiza una determinada acción.



Introducción a la programación orientada a eventos

➤ Componentes de esta aplicación:

- ❖ 2 etiquetas
- ❖ 2 campos de edición
- ❖ 2 Listas de desplegable
- ❖ 1 botón



➤ Características:

- ❖ Para cada elemento se han configurado sus propias características.

➤ Eventos:

- ❖ Al hacer click en el botón
- ❖ Al introducir un valor en los cuadros de texto
- ❖ Al cambiar el valor

➤ Elementos de la programación visual

- ❖ Se encuentra formada por una **ventana** donde se incluyen todos los componentes
 - ✓ **Componentes** como por ejemplo Botones, barra de desplazamiento, etiquetas,
- ❖ **Contenedores**: un componente que contiene otros componentes como por ejemplo ventana, panel, ...
- ❖ **Eventos**: En estos elementos se van a programar las acciones que queremos que se ejecuten cuando un usuario realiza un determinado evento con ellos.
- ❖ Estos elementos (formularios y componentes) poseen una serie de **características** que se pueden modificar tales como el color, tamaño, texto, puntero, etc. --> **Propiedades**.
- ❖ Todos los objetos del mismo tipo tendrán las mismas propiedades: Los botones siempre tienen las mismas características

Introducción a la programación orientada a eventos

- En Java existen varias librerías para crear interfaces GUI (Graphical User Interfaces):
 - ❖ AWT: Abstract Window Toolkit
 - ❖ Swing: más moderna, basada en AWT
 - ❖ SWT: Standard Widget Toolkit (Eclipse)
- En esta asignatura utilizaremos Swing+AWT las cuales forman parte de las JFC (Java Foundation Classes)
- Paquetes
 - ❖ java.awt
 - ❖ javax.swing
(<http://download.oracle.com/javase/tutorial/uiswing/components/index.html>)

➤ Abstract Windowing Toolkit (AWT)

- ❖ “Look& Feel” dependiente de la plataforma

- ✓ - La apariencia de ventanas, menús, etc. es distinta en Windows, Mac, Motif, y otros sistemas

- ❖ Funcionalidad independiente de la plataforma

- ❖ Básico y experimental

- ❖ Estándar hasta la versión JDK 1.1.5

➤ Swing / Java Foundation Classes (desde JDK 1.1.5)

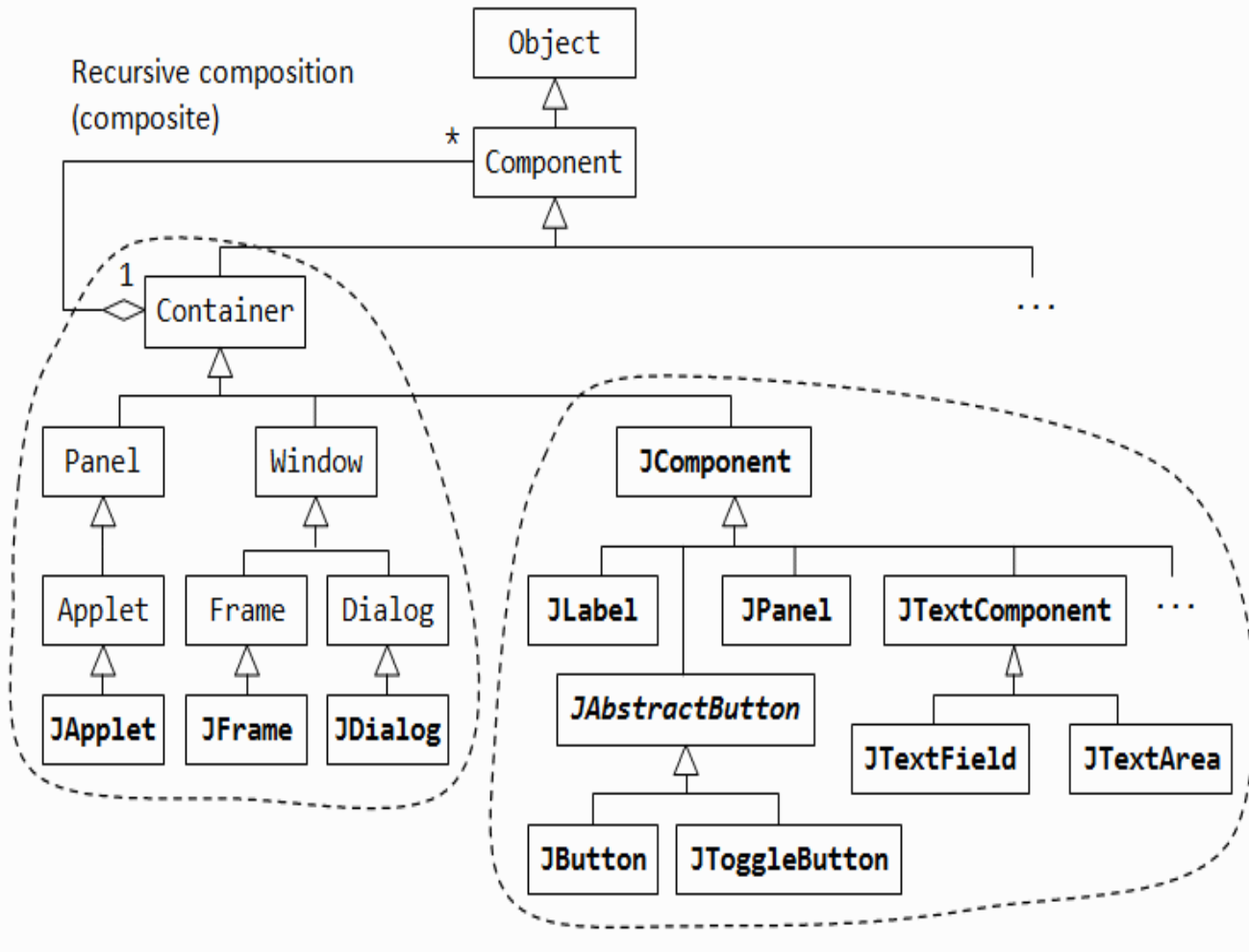
- ❖ “Look& Feel” y funcionalidad independiente de la plataforma (“Java Look& Feel”)

- ❖ Los menús y controles son como los de las aplicaciones “nativas”

- ❖ Las aplicaciones se les puede dar una apariencia en función de la plataforma específica

- ❖ Nuevas funcionalidades

Introducción a la programación orientada a eventos



➤ Pasos para GUI en Java:

- ❖ Contenedor alto nivel: Window (JFrame o JDialog) o Applet
- ❖ Crear contenedores nivel intermedio: Paneles
- ❖ Incluir componentes en los contenedores
- ❖ Manejar los eventos

➤ Los dos tipos de ventanas principales son JFrame y JDialog.

❖ JFrame

- ✓ Es la ventana principal de nuestra aplicación y sólo debe haber una.
- ✓ Compuesta por barra de título y marco

❖ JDialog

- ✓ Es una ventana secundaria de nuestra aplicación principal.
- ✓ Por el motivo anterior, admite otra ventana (JFrame o JDialog) como padre en el constructor. JFrame no admite padres.

❖ Otras diferencias:

- ✓ JFrame tiene un método setIconImage() para cambiar el icono por defecto
- ✓ JDialog puede ser modal, un JFrame no.

❖ Ventana modal:

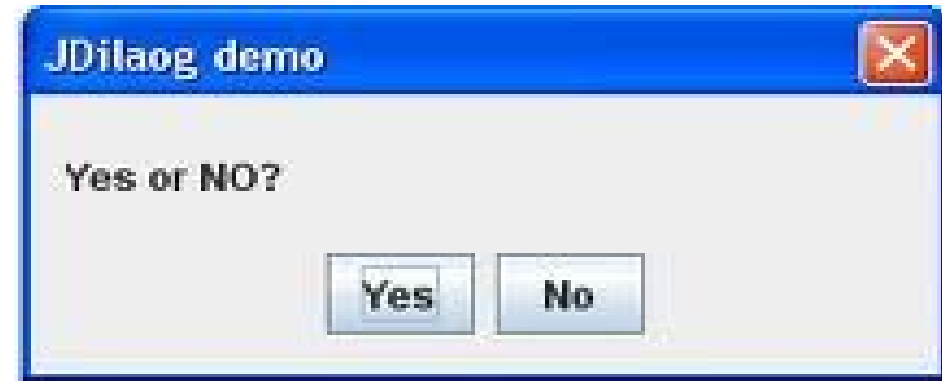
- ✓ Si se hace un JDialog modal, todas las demás ventanas se deshabilitarán hasta que el usuario cierre el JDialog (al pulsar intro, al cerrar, al cancelar, etc).

JFrame: <http://download.oracle.com/javase/6/docs/api/javax/swing/JFrame.html>

JDialog: <http://download.oracle.com/javase/6/docs/api/javax/swing/JDialog.html>



JFrame



JDialog

➤ JFrame. Métodos más importantes:

❖ void setLocation(int x, int y)

- ✓ Posiciona la ventana en la pantalla. (x, y) son las coordenadas de su vértice superior izquierdo

❖ void setSize(int ancho, int alto)

- ✓ Cambia el tamaño de la ventana

❖ void setVisible(boolean b)

- ✓ Visualiza o no la ventana dependiendo del valor booleano.

❖ void pack()

- ✓ Reduce la ventana alrededor de sus componentes

❖ void setDefaultCloseOperation (int acción);

- ✓ Acción al cerrar la ventana (Jframe.EXIT_ON_CLOSE)

❖ void setTitle(String title):

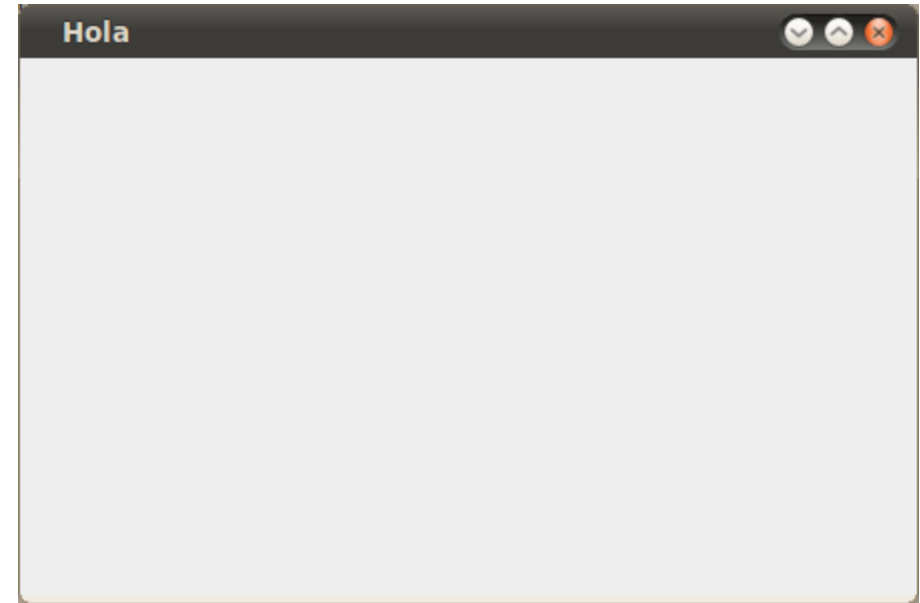
- ✓ Cambia el nombre de la barra de título de la ventana

❖ void setBounds(int x, int y, int width, int height)

- ✓ Posiciona la ventana y establece su ancho

➤ JFrame mediante composición (Prueba2.java)

```
public class Prueba2 {  
    private JFrame frame;  
    public static void main(String[] args) {  
        EventQueue.invokeLater(new Runnable() {  
            public void run() {  
                try {  
                    Prueba2 window = new Prueba2();  
                    window.frame.setVisible(true);  
                } catch (Exception e) {  
                    e.printStackTrace();  
                }  
            }  
        });  
    }  
    /**  
     * Create the application.  
     */  
    public Prueba2() {  
        initialize();  
    }  
    /**  
     * Initialize the contents of the frame.  
     */  
    private void initialize() {  
        frame = new JFrame("Hola");  
        frame.setBounds(100, 100, 450, 300);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}
```

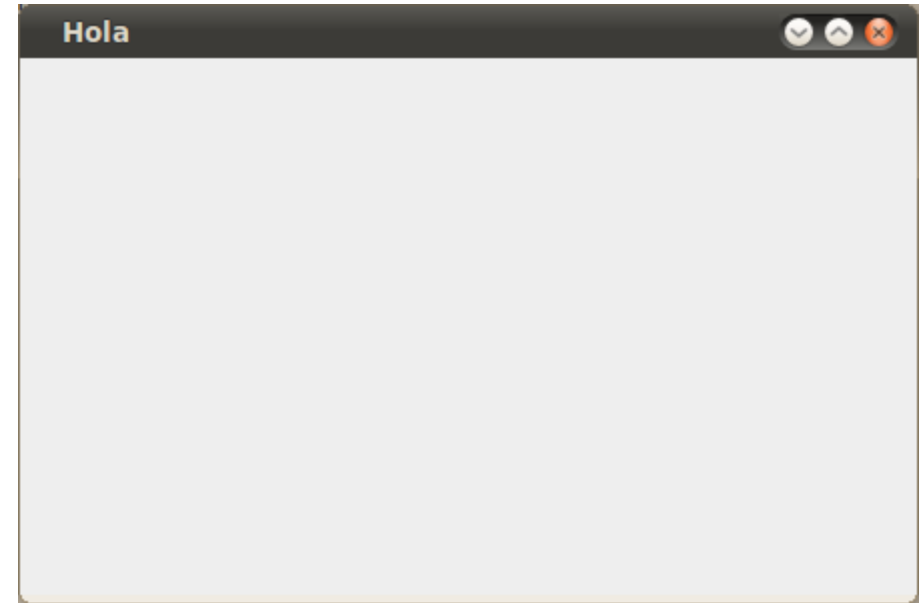


➤ JFrame mediante herencia (Prueba.java)

```
public class Prueba extends JFrame {
```

```
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Prueba frame = new Prueba();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Prueba() {
        setTitle("Hola"); //hereda
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //hereda
        setBounds(100, 100, 450, 300); //Hereda
    }
}
```



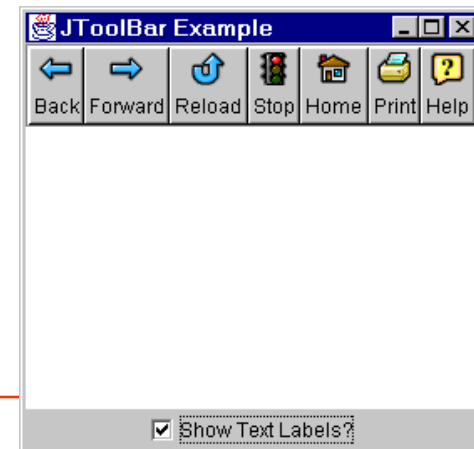
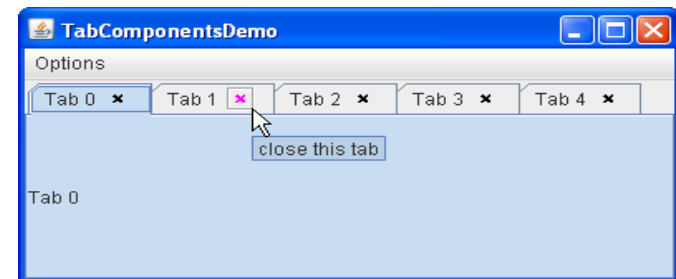
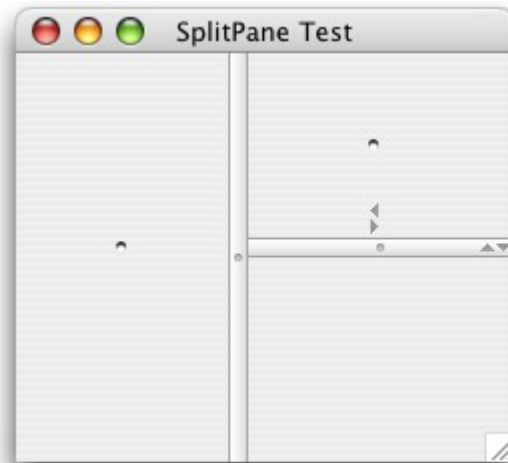
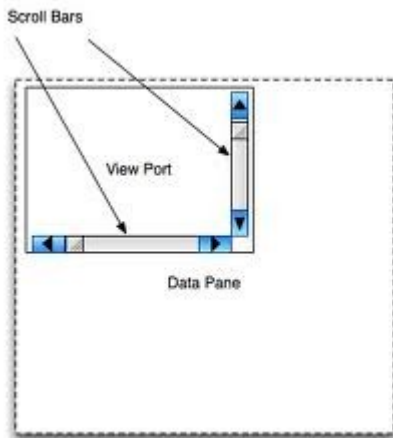
Paneles (JFrame)

- Cada ventana tiene un panel asociado: “content pane”
 - ❖ Sobre él se establecen los componentes o contenedores que van a formar parte de la aplicación.
 - ❖ Se debe establecer la distribución del panel y el tipo
 - ❖ Se utiliza el método `setContentPane` para establecer el panel de `JFrame`

```
public class Prueba extends JFrame {  
  
    private JPanel contentPane;  
  
    public static void main(String[] args) {  
        .....  
    }  
  
    public Prueba() {  
        setTitle("Hola"); //hereda  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //hereda  
        setBounds(100, 100, 450, 300); //Hereda  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5)); //Tamaño border  
        contentPane.setLayout(new BorderLayout(0, 0)); //Tipo de layout  
        setContentPane(contentPane);  
    }  
}
```

Paneles (JFrame)

- Swing tiene varios tipos de paneles contenedores:
 - ❖ JPanel – Contenedor
 - ❖ JScrollPane – Contenedor con barras de desplazamiento
 - ❖ JSplitPane – Contenedor dividido en dos partes
 - ❖ JTabbedPane – Contenedor con pestañas
 - ❖ JDesktopPane – Contenedor para incluir ventanas dentro
 - ❖ JToolBar – Barra de herramientas



Paneles (JFrame)

- Cada uno de los paneles anteriores, se componen de componentes (botones, etiquetas, ...) → Se añaden mediante el método **add(Component comp)**.

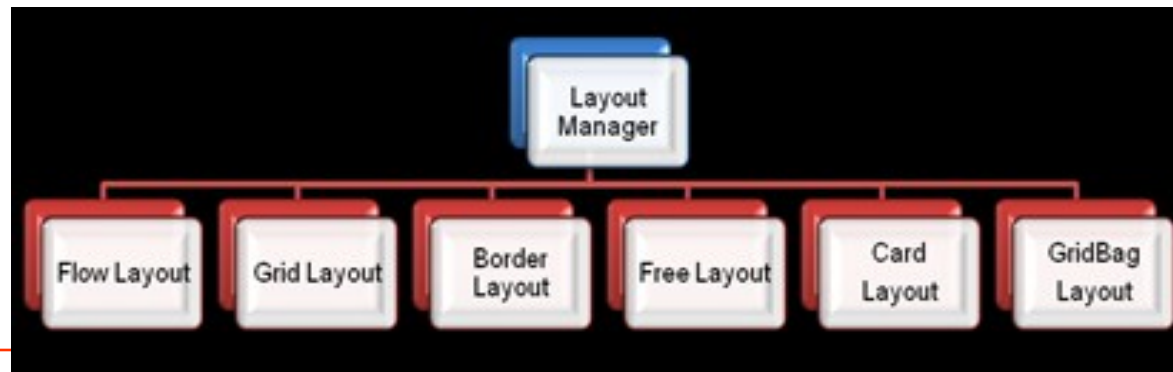
```
JFrame j = new JFrame("Un ejemplo");
Container cont = j.getContentPane();
cont.setLayout(new FlowLayout()); //Distribución de los
componentes
JButton bAceptar = new JButton("Aceptar");
JButton bCancelar = new JButton("Cancelar");
JLabel lab = new JLabel("¿Qué decides?");
cont.add(lab);
cont.add(bAceptar);
cont.add(bCancelar);
```

- También se pueden eliminar componentes:
void remove(Component comp)

Estos componentes se distribuyen
a partir de distintos layouts o gestores de distribución

Paneles (Jframe). Layout

- Layout: Permiten determinar la forma en que se distribuyen los componentes dentro de un contenedor
 - ❖ FlowLayout (por defecto)
 - ❖ AbsoluteLayout
 - ❖ BoxLayout
 - ❖ GridLayout
 - ❖ BorderLayout
 - ❖ CardLayout
 - ❖ SpringLayout
 - ❖ GroupLayout



➤ AbsoluteLayout (Sin layout):

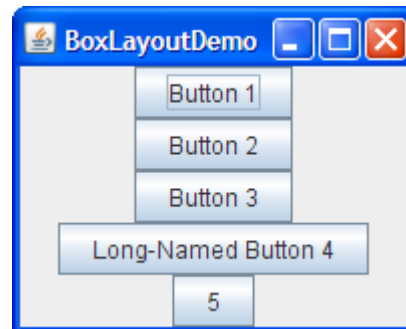
- ❖ En código se determina el tamaño y posición de cada componente: botón, cuadro de texto, etc
- ❖ No se recomienda su uso, pues cuando se redimensiona el efecto obtenido puede ser bastante negativo.

```
contenedor.setLayout(null); // Eliminamos el layout
contenedor.add (boton); // Añadimos el botón
boton.setBounds (10,10,40,20); // Botón en posicion 10,10 con ancho 40 pixels y alto 20
```

Paneles (Jframe). Layout

➤ BorderLayout (BoxLayoutDemo.java)

- ❖ BorderLayout es como un FlowLayout con la excepción que permite colocar los elementos en horizontal o vertical.
- ❖ Suele utilizarse cuando los componentes van a los lados de la aplicación (en vertical).
- ❖ El constructor del BorderLayout es más complejo que el del FlowLayout. Para crear una clase BorderLayout, necesitamos 2 argumentos:
 - ✓ el objeto contenedor,
 - ✓ y la clase que indica la forma de como ordenara los componentes,.



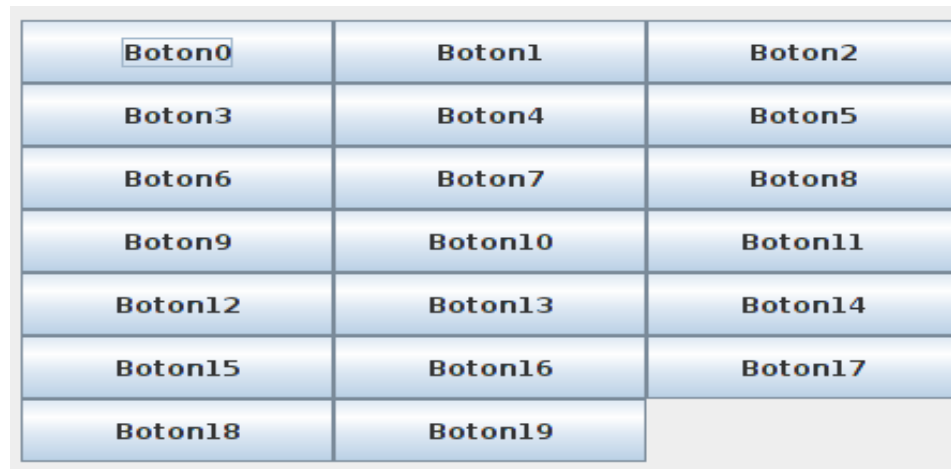
➤ FlowLayout (FlowLayoutDemo.java)

- ❖ coloca los componentes en fila adaptándose para que todos los componentes cojan en la ventana (si el tamaño de la ventana lo permite).
- ❖ Es adecuado para barras de herramientas, filas de botones, etc.
- ❖ Si en el constructor de FlowLayout no ponemos nada, los botones irán en fila horizontal centrados en el panel.
- ❖ Se puede establecer su alineación :FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER, FlowLayout.LEADING y FlowLayout.Trailing.
- ❖ También se puede indicar qué espacio se quiere arriba, abajo, izquierda y derecha e incluso de separación entre botones.
 - ✓ `new FlowLayout(FlowLayout.CENTER, 35, 35))`

Paneles (Jframe). Layout

➤ GridLayout (Ejemplo adicional: DemoGridLayoutA.java)

- ❖ Coloca los componentes en forma de matriz (cuadrícula), estirándolos para que tengan todos el mismo tamaño.
- ❖ Es necesario indicar en el constructor el número de filas y columnas que va a tener la rejilla (grid)
- ❖ También se puede indicar el espacio entre los componentes
- ❖ Es adecuado para hacer tableros, calculadoras en que todos los botones son iguales, etc.



Boton0	Boton1	Boton2
Boton3	Boton4	Boton5
Boton6	Boton7	Boton8
Boton9	Boton10	Boton11
Boton12	Boton13	Boton14
Boton15	Boton16	Boton17
Boton18	Boton19	

Paneles (Jframe). Layout

➤ FlowLayout vs BoxLayout vs GridLayout (DemoLayouts.java)

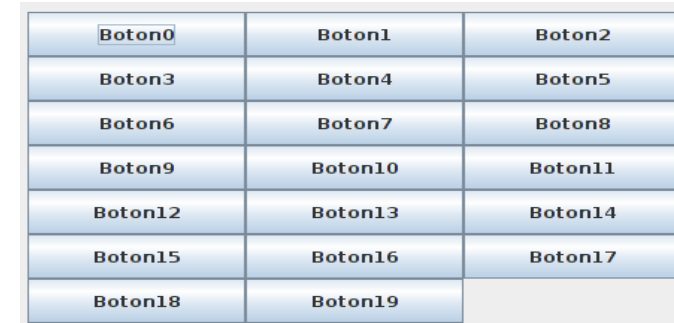
```
public class DemoLayouts extends JFrame {  
    .....  
    public DemoLayouts() {  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 450, 300);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        //Distribución como FlowLayout  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
        //Distribución como BoxLayout  
        contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.Y_AXIS));  
        //Distribución como GridLayout  
        contentPane.setLayout(new GridLayout(0, 3, 0, 0));  
        for (int i=0;i<20;i++){  
            JButton b= new JButton("Boton"+i);  
            contentPane.add(b);  
        }  
    }  
}
```



FlowLayout



BoxLayout

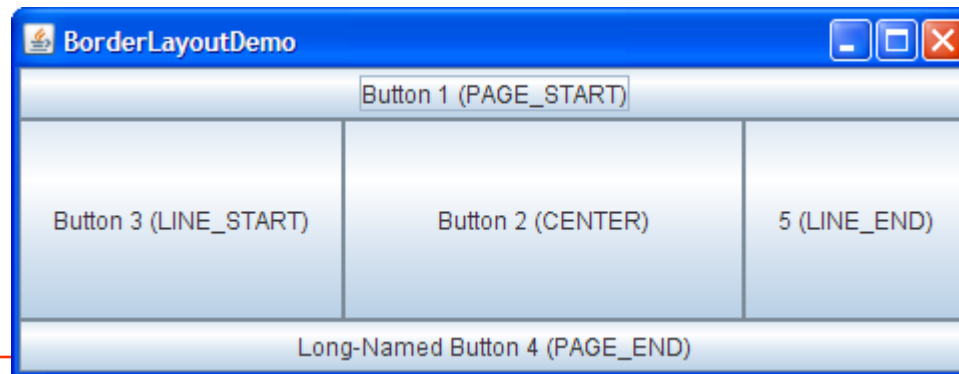


GridLayout

Paneles (Jframe). Layout

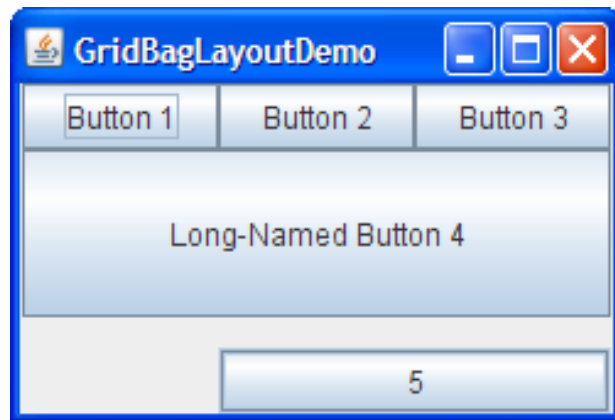
➤ BorderLayout (DemoBorderLayout.java)

- ❖ Divide la ventana en 5 partes: centro, arriba, abajo, derecha e izquierda.
- ❖ Los componentes situados arriba y abajo ocupan el alto que necesitan, pero los estirará horizontalmente hasta ocupar toda la ventana.
- ❖ Los componentes de derecha e izquierda ocuparán el ancho que necesiten, pero se les estirará en vertical hasta ocupar toda la ventana.
- ❖ El componente central se estirará en ambos sentidos hasta ocupar toda la ventana.
- ❖ Es adecuado para ventanas en las que hay un componente central importante (una tabla, una lista, etc) y tiene menús o barras de herramientas situados arriba, abajo, a la derecha o a la izquierda.



Paneles (Jframe). Layout

- GridBagLayout (DemoGridBagLayout.java)
 - ❖ Es el controlador de disposición más flexible - y complejo - proporcionado por la plataforma Java
 - ❖ Sitúa los componentes en una parrilla de filas y columnas, permitiendo que los componentes se expandan más de una fila o columna.
 - ❖ No es necesario que todas las filas tengan la misma altura, ni que las columnas tengan la misma anchura.



➤ CardLayout

- ❖ Los componentes ocupan el máximo espacio posible, superponiendo unos a otros. Sólo es visible uno de los componentes, los otros quedan detrás.
- ❖ Por ejemplo es usado por JTabbedPane (el de las pestañas) de forma que en función de la pestaña que pinchemos, se ve uno u otro.

➤ SpringLayout

- ❖ Se añaden los componentes y para cada uno de ellos tenemos que decir qué distancia en pixel queremos que tenga cada uno de sus bordes respecto al borde de otro componente.

➤ GroupLayout

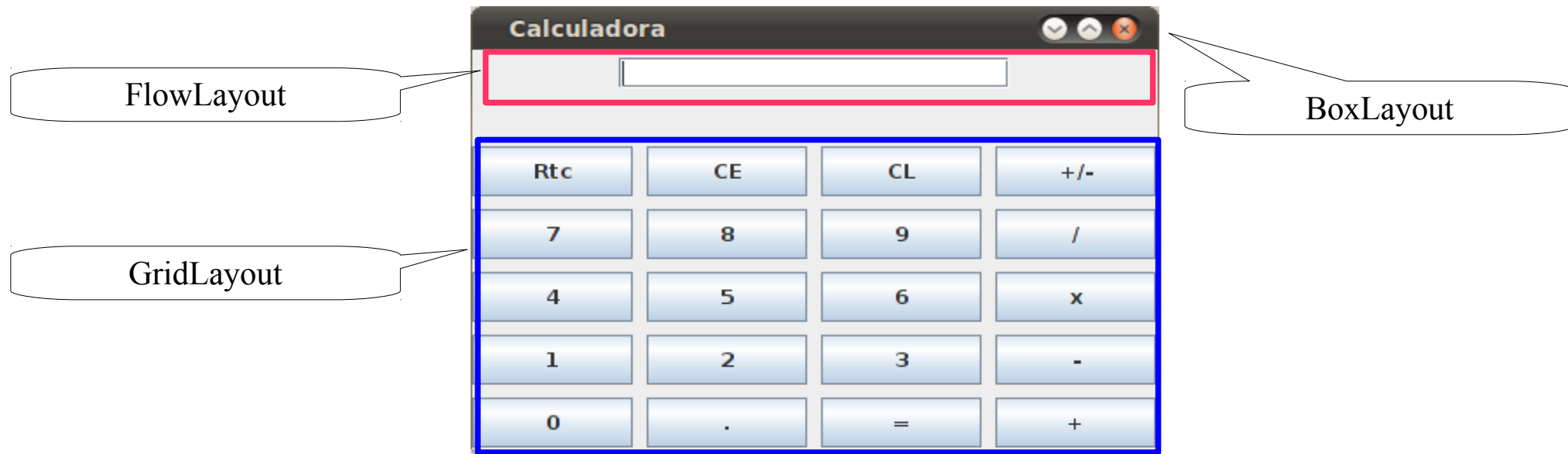
- ❖ Se centra en la creación anidada de grupos de componentes tanto horizontal (`createSequentialGroup()`) como verticalmente (`createParallelGroup()`), pudiéndose añadir tanto grupos de componentes como componentes solamente.

Paneles (Jframe). Layout

- Ejemplo1. Analizar los paneles y layout de la siguiente aplicación

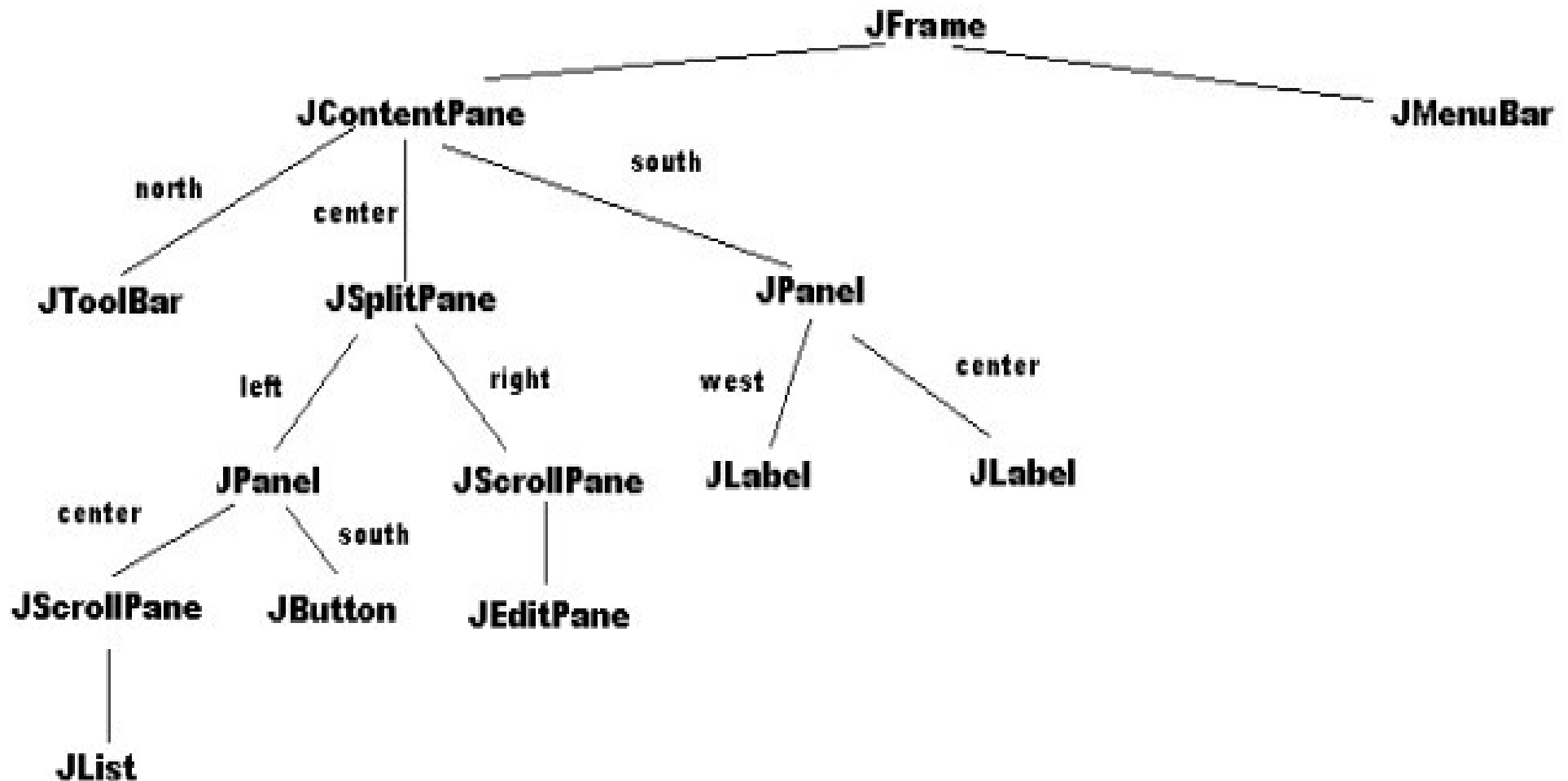


➤ Ejemplo1. Analizar los paneles y layout de la siguiente aplicación



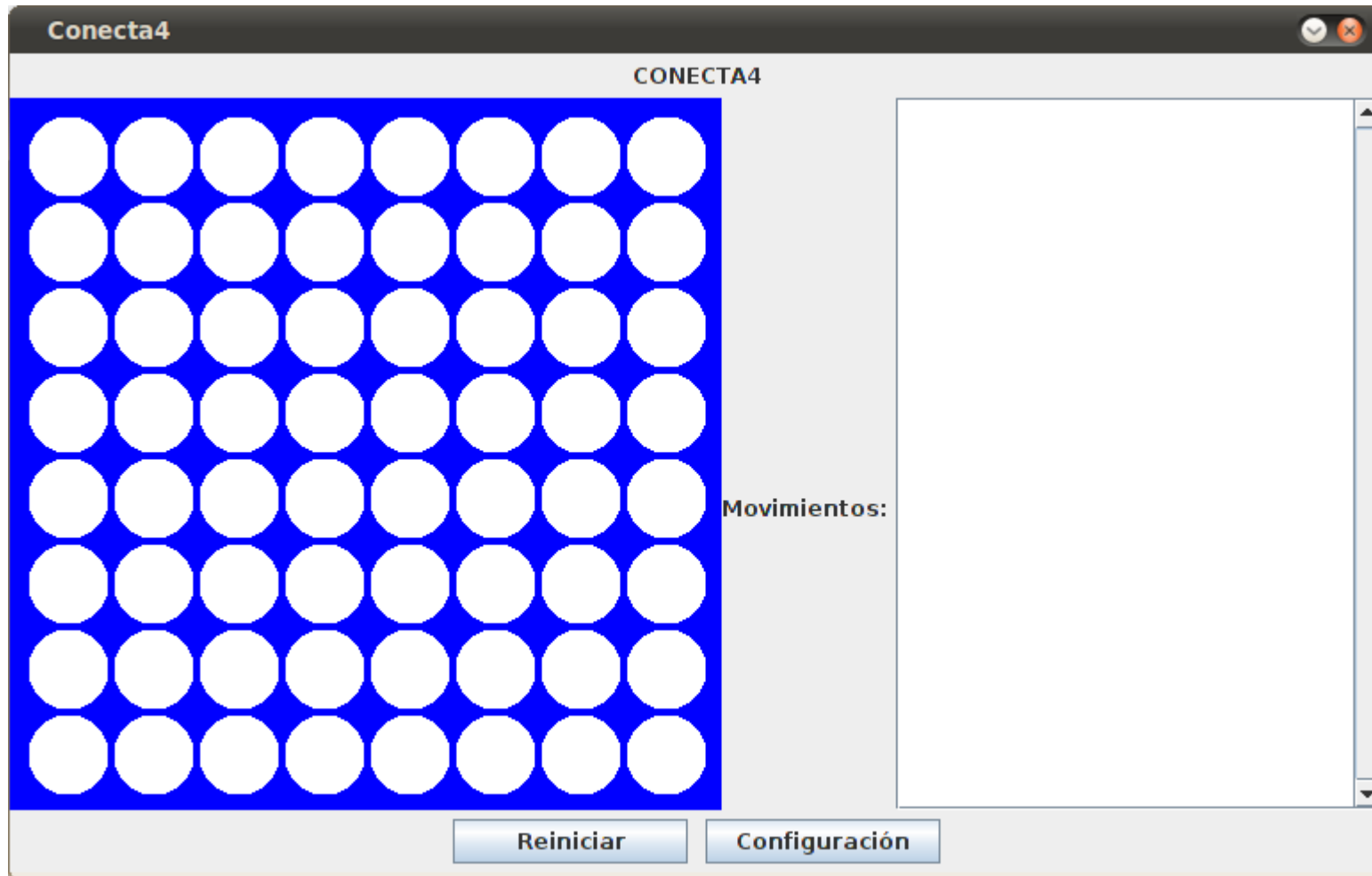
Paneles (Jframe). Layout

- Ejemplo2. Analizar los paneles y layout de la siguiente aplicación



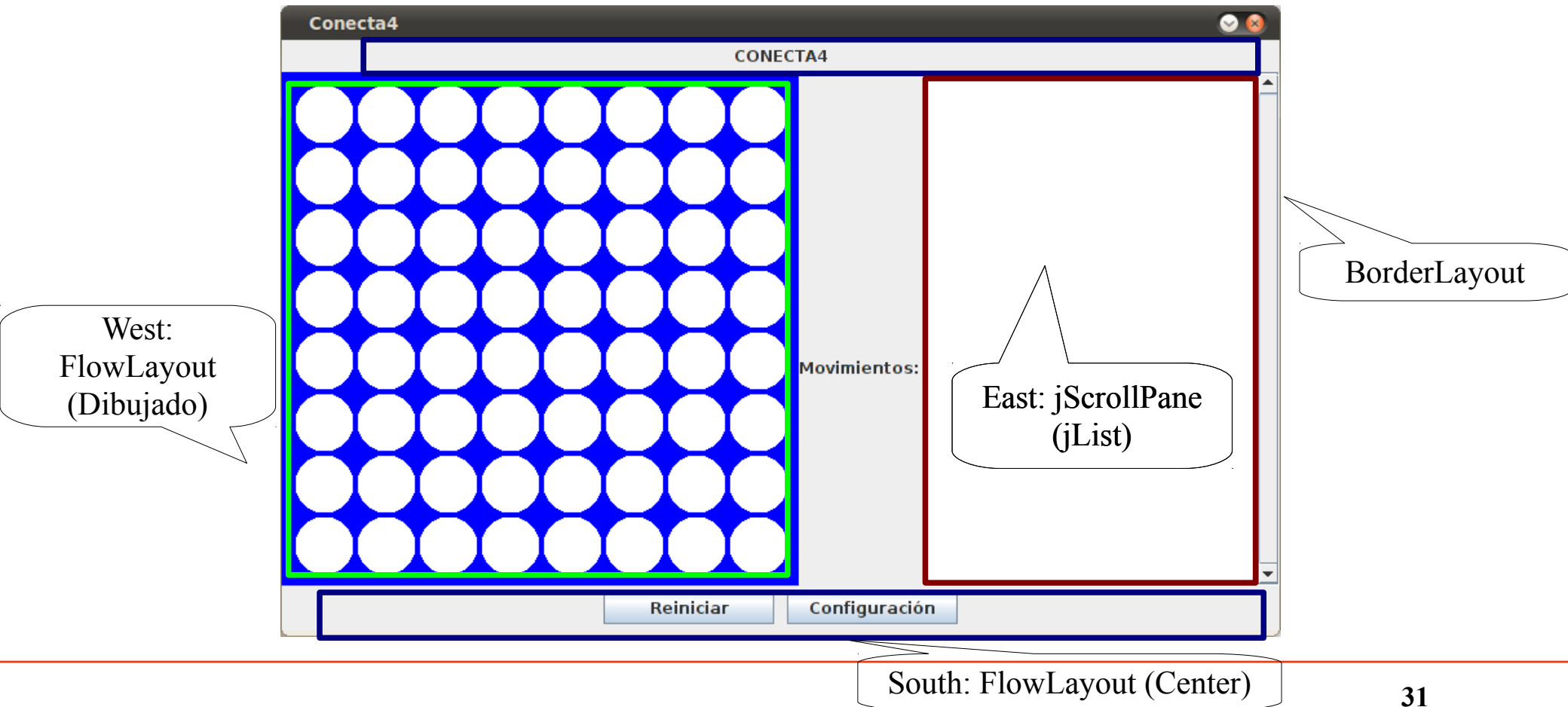
Paneles (Jframe). Layout

- Ejemplo3. Analizar los paneles y layout de la siguiente aplicación



Paneles (Jframe). Layout

- Ejemplo3. Analizar los paneles y layout de la siguiente aplicación

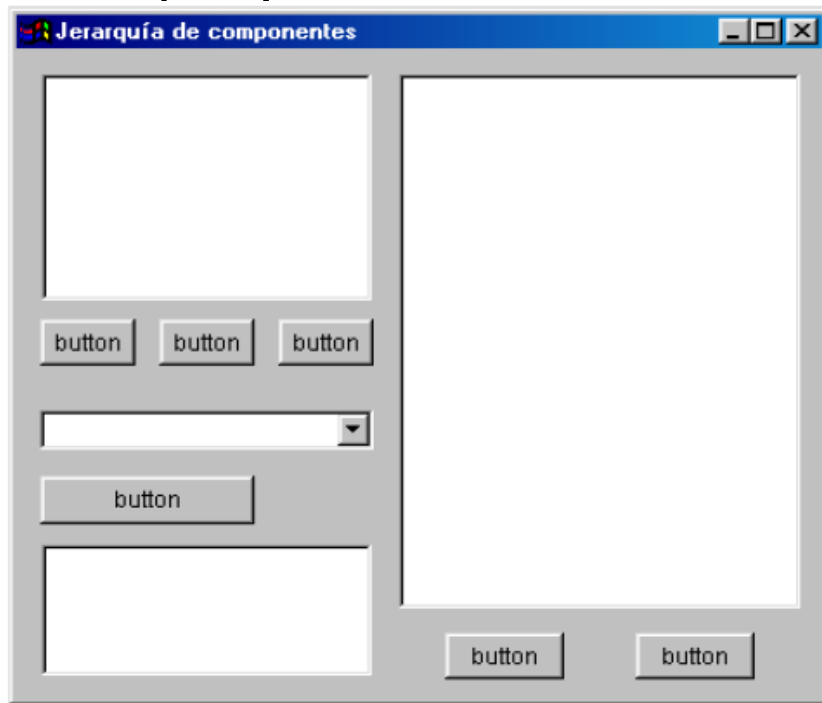


Paneles (Jframe). Layout

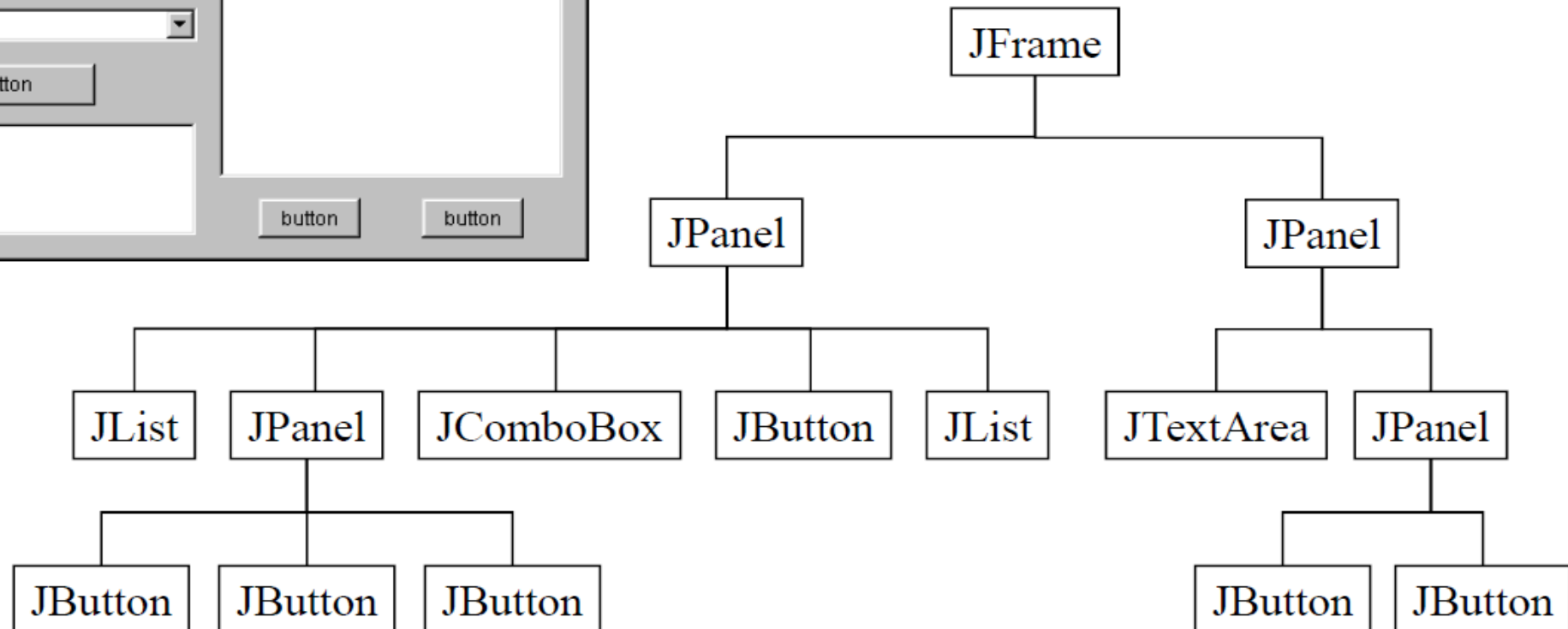
- Ejemplo2. Analizar los paneles y layout de la siguiente aplicación

Paneles (Jframe). Layout

➤ Ejemplo 4

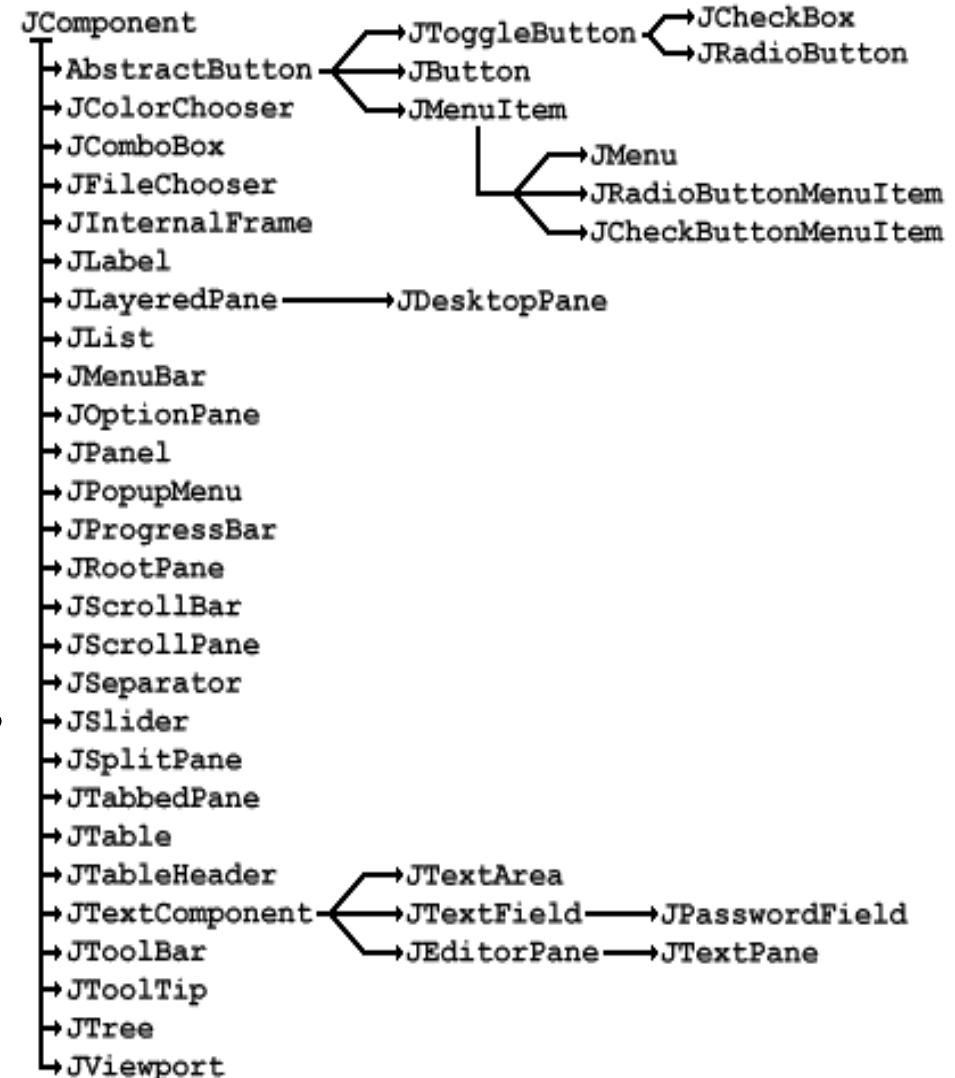


Ejemplo



Componentes en Java

- La mayoría de los componentes Swing están implementados como subclases de la clase `JComponent`, que descende de la clase `Container`.
- Todos los componentes Swing heredan las siguientes funcionalidades de `JComponent`.
 - ❖ Bordes, Tool tips, Navegación con Teclado, Aspecto y Comportamiento, Soporte de accesibilidad, Soporte de Localización, etc



➤ Clases Auxiliares:

❖ Posiciones y tamaños:

- ✓ Dimension: width, height
- ✓ Point: x, y
- ✓ Rectangle: x, y, width, height, contains(Point)
- ✓ Polygon: npoints, xpoints, ypoints, addPoint(Point)

❖ Color:

- ✓ new Color(0.8f, 0.3f, 1.0f) en RGB
- ✓ Constantes de tipo Color: Color.white, Color.blue, etc.

❖ Font:

- ✓ new Font("Helvetica", Font.BOLD + Font.ITALIC, 18)
- ✓ getName(), getStyle(), getSize()
- ✓ Constantes de estilo: Font.BOLD, Font.ITALIC,

❖ Cursor:

- ✓ new Cursor(Cursor.HAND_CURSOR), Cursor.CROSSHAIR_CURSOR, etc.

➤ Clase javax.swing.JComponent

- ❖ Dibujarse en la pantalla: `paintComponent(Graphics)`

- ❖ Control de la apariencia visual:

 - ✓ Color: `setForeground(Color)`, `getForeground()`, `setBackground(Color)`, `getBackground()`

 - ✓ Font: `setFont(Font)`, `getFont()`

 - ✓ Cursor: `setCursor(Cursor)`, `getCursor()`

- ❖ Tamaño y posición:

 - ✓ `setSize(int,int)`, `getSize()` → Dimension,

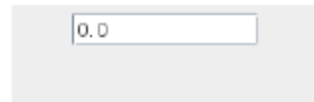
 - ✓ `getLocation()` → Point,

 - ✓ `getLocationOnScreen()` → Point,

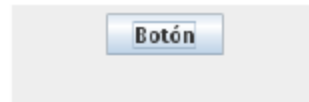
 - ✓ `setBounds(int,int,int,int)`,

 - ✓ `getBounds()` → Rectangle

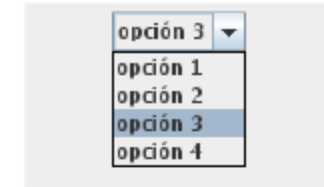
Componentes en Java



JTextField



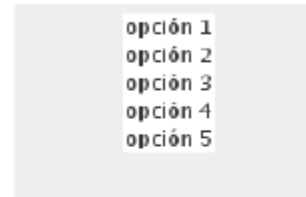
JButton



JComboBox



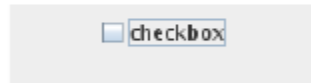
JLabel



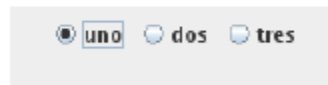
JList



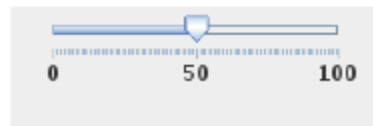
JRadioButton



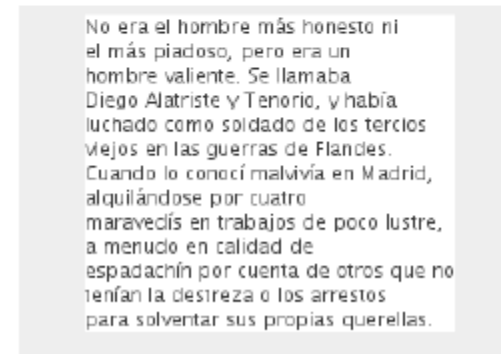
JCheckBox



ButtonGroup



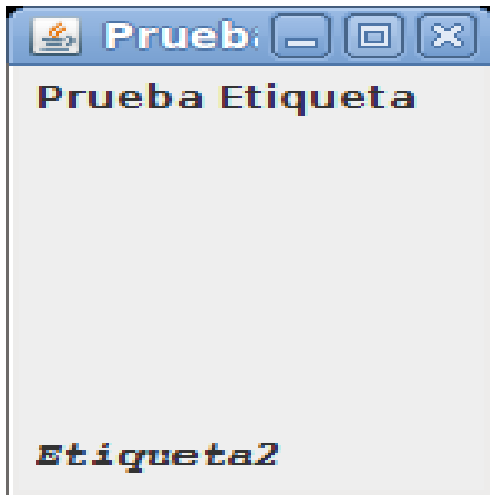
JSlider



JTextArea

➤ JLabel (DemoJLabel.java)

- ❖ Nos permite mostrar un texto, gráficos o ambos
- ❖ Se trata de un componente de lectura, es decir, no puede ser editado (aunque si modificado).
- ❖ Métodos más importantes
 - ✓ Constructor (JLabel(String text)). Establece el texto
 - ✓ **String getText(): Retorna el texto de la etiqueta** (en muchos componentes)
 - ✓ **void setText(String text): Cambia el texto de la etiqueta** (en muchos componentes)
 - ✓ Void setFont(Font): Establece configuración de la fuente de letra



```
JLabel lblPruebaEtiqueta = new JLabel("Prueba Etiqueta");
contentPane.add(lblPruebaEtiqueta, BorderLayout.NORTH);

JLabel lblEtiqueta = new JLabel("Etiqueta2");
lblEtiqueta.setFont(new Font("Courier 10 Pitch", Font.BOLD
    | Font.ITALIC, 14));
contentPane.add(lblEtiqueta, BorderLayout.SOUTH);
```

➤ JButton

❖ Es un botón que puede contener texto, gráficos, o ambos.

- ✓ Fijar el texto siempre centrado, en caso de contener una imagen, ha de ir a la izquierda o encima del texto.

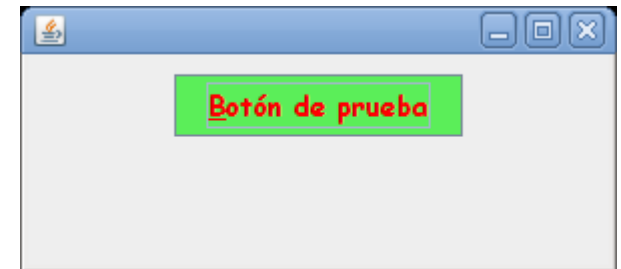
❖ Métodos importantes

- ✓ Constructor (JButton(String text)): Establece el texto del botón
- ✓ setText("Texto"); Establece el texto
- ✓ setToolTipText("Tooltip"); Establece el contenido del texto emergente
- ✓ setBackground(new Color(R, G, B)); Establece el color de fondo
- ✓ setForeground(Color.color); Establece el color principal
- ✓ setIcon(new ImageIcon("ruta")); Establece un icono
- ✓ setFont(new Font("tipo", estilo, tamaño)); Establece la fuente
- ✓ setBounds(new Rectangle(posX,posY,tamX,tamY));

❖ Y sus correspondientes get

➤ JButton (DemoJButton.java)

```
public class DemoJButton extends JFrame {  
    .....  
    public DemoJButton() {  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 450, 300);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
        JButton boton1 = new JButton();  
        boton1.setBounds(new Rectangle(107, 50, 102, 41));  
        boton1.setBackground(new Color(91, 238, 89));  
        boton1.setForeground(Color.red);  
        boton1.setToolTipText("Prueba");  
        boton1.setFont(new Font("Comic Sans MS", Font.BOLD, 14));  
        boton1.setText("Botón de prueba");  
        boton1.setMnemonic(KeyEvent.VK_B);  
        contentPane.add(boton1);  
    }  
}
```



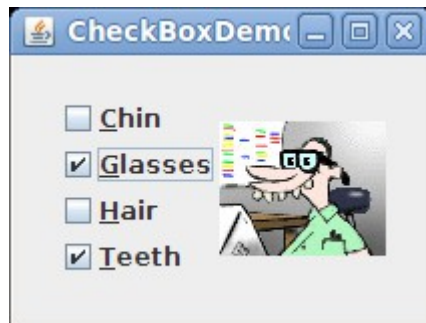
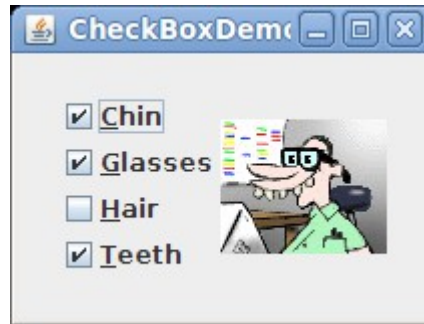
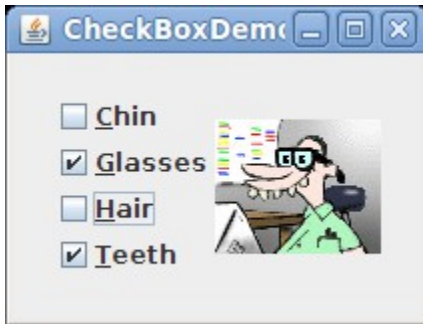
➤ JToggleButton (DemoJToggleButton.java)

- ❖ Es un botón que representa dos estados (On y Off):
 - ✓ Por ejemplo: botones de negrita, cursiva de un editor de texto.
- ❖ Tiene las mismas propiedades que JButton.
- ❖ Existen dos posibles casos:
 - ✓ Independientes (Checkboxes): donde cada uno de los botones tiene su estado por separado
 - ✓ Exclusivas (RadioButton): donde cada uno de los botones comparten un único estado.
- ❖ Tiene dos métodos adicionales principalmente:
 - ✓ `isSelected()`; Que determina si el botón esta en ON
 - ✓ `setSelected(boolean)`; Establece el estado del botón



➤ JCheckBox

- ❖ Es un botón que representa dos estados (On y Off) (independientes)
- ❖ Métodos: `isSelected()` y `setSelected()` (boolean)



```
//In initialization code:
chinButton = new JCheckBox("Chin");
chinButton.setMnemonic(KeyEvent.VK_C);
chinButton.setSelected(true);
glassesButton = new JCheckBox("Glasses");
glassesButton.setMnemonic(KeyEvent.VK_G);
glassesButton.setSelected(true);
hairButton = new JCheckBox("Hair");
hairButton.setMnemonic(KeyEvent.VK_H);
hairButton.setSelected(true);
teethButton = new JCheckBox("Teeth");
teethButton.setMnemonic(KeyEvent.VK_T);
teethButton.setSelected(true);
//Register a listener for the check boxes.
chinButton.addItemListener(this);
glassesButton.addItemListener(this);
hairButton.addItemListener(this);
teethButton.addItemListener(this);

...
public void itemStateChanged(ItemEvent e) {
    ...
    Object source = e.getItemSelectable();
    if (source == chinButton) {
        //...make a note of it...
    } else if (source == glassesButton) {
        //...make a note of it...
    } else if (source == hairButton) {
        //...make a note of it...
    } else if (source == teethButton) {
        //...make a note of it...
    }
    if (e.getStateChange() ==
        ItemEvent.DESELECTED)
        //...make a note of it...
    ...
    updatePicture();
}
```

Componentes en Java

➤ JRadioButton (DemoJRadioButton.java)

- ❖ Permiten seleccionar una única opción dentro de un conjunto
- ❖ Sólo puede haber una opción seleccionada a la vez.
- ❖ Para darle este comportamiento, tiene que usarse un ButtonGroup.



```
//In initialization code:
//Create the radio buttons.
JRadioButton birdButton = new JRadioButton(birdString);
birdButton.setMnemonic(KeyEvent.VK_B);
birdButton.setActionCommand(birdString);
birdButton.setSelected(true);

JRadioButton catButton = new JRadioButton(catString);
catButton.setMnemonic(KeyEvent.VK_C);
catButton.setActionCommand(catString);

JRadioButton dogButton = new JRadioButton(dogString);
dogButton.setMnemonic(KeyEvent.VK_D);
dogButton.setActionCommand(dogString);

JRadioButton rabbitButton = new
JRadioButton(rabbitString);
rabbitButton.setMnemonic(KeyEvent.VK_R);
rabbitButton.setActionCommand(rabbitString);

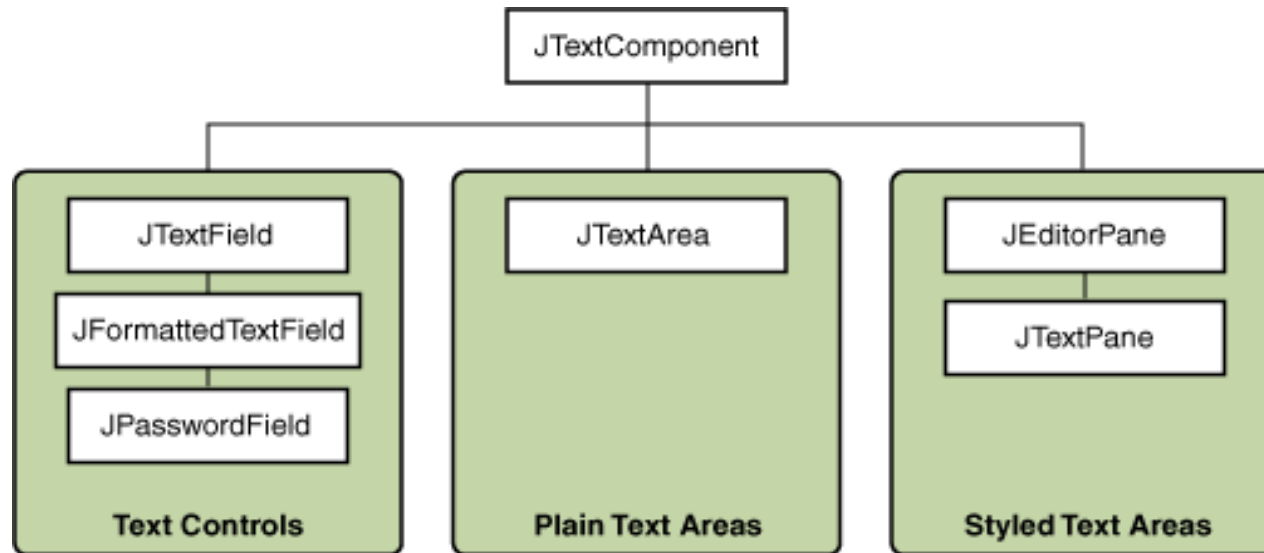
JRadioButton pigButton = new JRadioButton(pigString);
pigButton.setMnemonic(KeyEvent.VK_P);
pigButton.setActionCommand(pigString);
//Group the radio buttons.
ButtonGroup group = new ButtonGroup();
group.add(birdButton);
group.add(catButton);
group.add(dogButton);
group.add(rabbitButton);
group.add(pigButton);
//Register a listener for the radio buttons.
birdButton.addActionListener(this);
catButton.addActionListener(this);
dogButton.addActionListener(this);
rabbitButton.addActionListener(this);
pigButton.addActionListener(this);

...
public void actionPerformed(ActionEvent e) {
    picture.setIcon(new ImageIcon("images/"
                                + e.getActionCommand()
                                + ".gif"));
}
```

➤ JTextComponent

❖ Componentes que muestran texto que puede ser editable.

- ✓ setText("Texto") se le asigna el texto.
- ✓ String() getText(): recupera el texto almacenado



➤ JTextComponent esta compuesto por: (1/2)

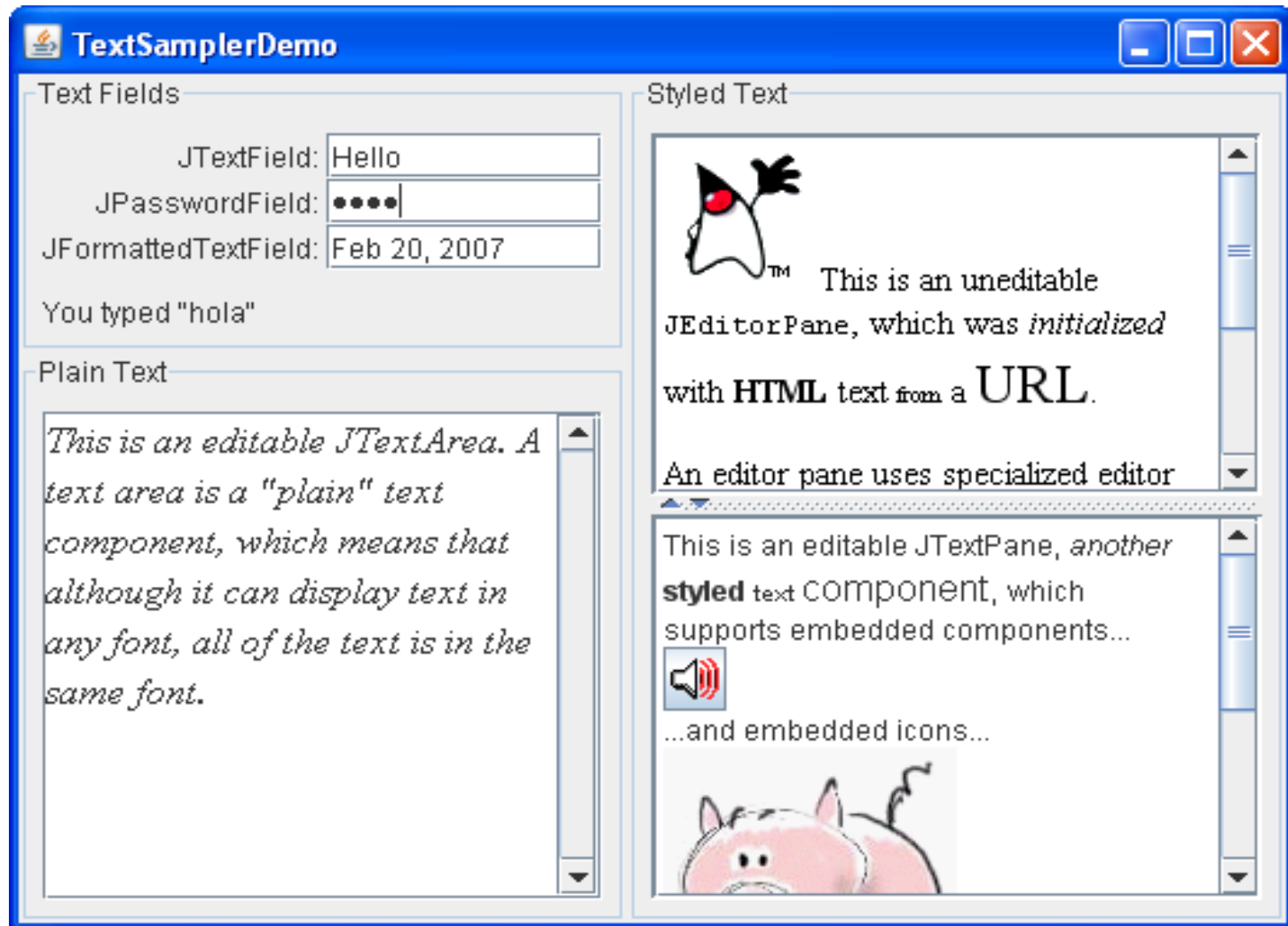
- ❖ JTextField: Una línea de texto. No admite mezclar fuentes de texto ni colores.
 - ✓ Constructor: `JTextField()`, `JTextField(int columns)`, `JTextField(String text)`, `JTextField(String text, int columns)`
 - ✓ `getText()` y `setText(String)`: Permiten modificar el texto
- ❖ JFormattedTextField: Permite pedir datos, de una sola línea, que cumplan unas ciertas restricciones o un cierto formato.
 - ✓ Adecuado para pedir fechas, horas, direcciones IP, datos numéricos para controlar que no se puedan escribir letras.
 - ✓ <http://www.chuidiang.com/java/ejemplos/JFormattedTextField/EjemplosJFormattedTextField.php>
- ❖ JPasswordField: Oculta los caracteres introducidos por el usuario
 - ✓ `setEchoChar(char)` establece el carácter mostrado como máscara
 - ✓ `String getPassword()` recupera el password almacenado en el componente

➤ JTextComponent esta compuesto por: (2/2)

- ❖ JTextArea: Espacio rectangular en el que ver y editar múltiples líneas de texto. Es necesario añadir JScrollPane para que aparezca la barra de desplazamiento
 - ✓ En los constructores se puede establecer el número de filas y columnas así como el texto: `JTextArea()`, `JTextArea(int rows, int columns)`, `JTextArea(String text)`, `JTextArea(String text, int rows, int columns)`
 - ✓ Existe un método para determinar si es editable o no el `JTextArea` (`void setEditable(boolean b)`)
- ❖ JEditorPane (más complejo `JTextArea`): Admite texto plano, HTML y RTF y puede ser ampliado. Este sí permite mezclar fuentes, colores e imágenes.
 - ✓ Con el método `setContentTypes` se establece el tipo que puede ser ("`text/html`"), ("`text/rtf`") o ("`text/plain`")
 - ✓ Ejemplo: `editor.setText("hola
");`
- ❖ JtextPane: Similar al anterior aunque admite añadir texto donde cada uno tiene sus propios atributos de texto -fuentes, colores, etc-.

Componentes Java

➤ Ejemplo de JTextComponent



➤ JComboBox

- ❖ Permite seleccionar una opción dentro de un conjunto de opciones en un desplegable.
- ❖ Se trata de un componente de selección exclusiva, sólo una puede ser seleccionada.
- ❖ Se “alimenta” de un array de Object con los valores del desplegable
 - ✓ Puede proporcionarse en el constructor: `JComboBox(Object[] items)`
 - ✓ Mediante el método: `void addItem(Object anObject)`
- ❖ Métodos más importante:
 - ✓ `setEditable(boolean)`: Permite editar o no el combo
 - ✓ `getSelectedItem()`: Devuelve un Objecto con el Object seleccionado (es necesario realizar un upcasting)



Componentes Java

➤ JComboBox (DemoJComboBox.java)

```
public class DemoJComboBox extends JPanel
                                implements ActionListener {
    JLabel label;
    public DemoJComboBox() {
        super(new BorderLayout());

        String[] numeros = { "Uno", "Dos", "Tres", "Cuatro", "Cinco" };

        //Create the combo box, select the item at index 4.
        //Indices start at 0, so 4 specifies the five number.
        JComboBox numeroList = new JComboBox(numeros);
        numeroList.setSelectedIndex(4);
        numeroList.addActionListener( this );

        //Lay out the demo.
        add(numeroList, BorderLayout. PAGE_START);
        setBorder(BorderFactory. createEmptyBorder (20,20,20,20));

        label= new JLabel( "Valor" );
        add( label, BorderLayout. CENTER);
        label.setText( "Valor:" + (String)numeroList.getSelectedItem());
    }

    /** Listens to the combo box. */
    public void actionPerformed(ActionEvent e) {
        JComboBox cb = (JComboBox)e.getSource();
        String num = (String)cb.getSelectedItem();
        label.setText( "Valor:" + num);
    }
}
```

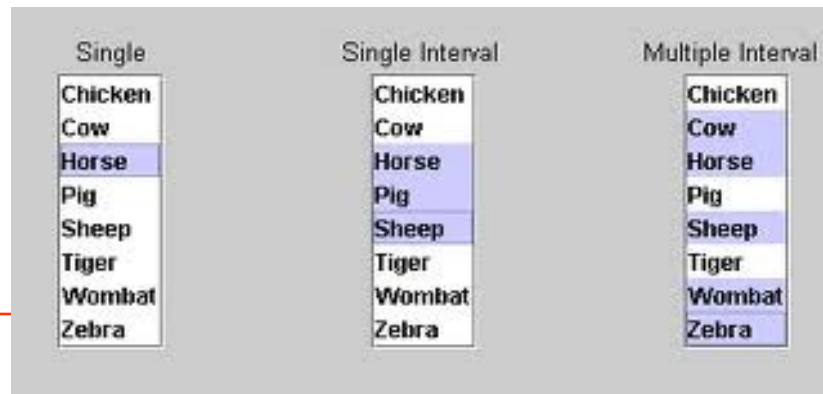


➤ JList

- ❖ Muestra un conjunto de ítems de texto, gráfico o ambos y permite la selección de los mismos.
- ❖ Al igual que el componente anterior se “alimenta” de un array o lista de objetos.
 - ✓ Constructores: `JList(Object[] listData)` o `JList(ListModel dataModel)`
- ❖ En este caso se pueden seleccionar uno o varios elementos mediante el método:

`void setSelectionMode(int selectionMode)` donde:

- ✓ `SINGLE_SELECTION`: Ítem único
- ✓ `SINGLE_INTERVAL_SELECTION`: Rango simple
- ✓ `MULTIPLE_INTERVAL_SELECTION`: Rango Múltiple



➤ JList

❖ Dependiendo del tipo que se permite, se debe usar un método u otro:

- ✓ `int getSelectedIndex()`: Número de la selección que se ha escogido
- ✓ `int[] getSelectedIndices()`: Números de la selección que se han escogidos
- ✓ `Object getSelectedValue()`: Objeto de la selección que se ha escogido
- ✓ `Object[] getSelectedValues()`: Objetos de la selección que se ha escogido

❖ Cambia la opción u opciones seleccionadas

- ✓ `void setSelectedIndex(int index)`
- ✓ `void setSelectedIndices(int[] índices)`

```
String[] numeros = { "Uno", "Dos", "Tres", "Cuatro", "Cinco" };  
JList jl=new JList(numeros);
```

```
.....
```

```
label1.setText((String)jl.getSelectedValue());
```

➤ JList (DemoJList.java)

```
public class DemoJList extends JPanel {
    JLabel label;
    JList numeroList;
    public DemoJList() {
        super(new BorderLayout());

        String[] numeros = { "Uno", "Dos", "Tres", "Cuatro", "Cinco" };

        //Create the combo box, select the item at index 4.
        //Indices start at 0, so 4 specifies the five number.
        numeroList = new JList(numeros);
        numeroList.setSelectedIndex(4);

        //Lay out the demo.
        add(numeroList, BorderLayout.PAGE_START);
        setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

        label = new JLabel("Valor");
        add(label, BorderLayout.CENTER);

        JButton btnActualizarValores = new JButton("Actualizar valores");
        btnActualizarValores.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

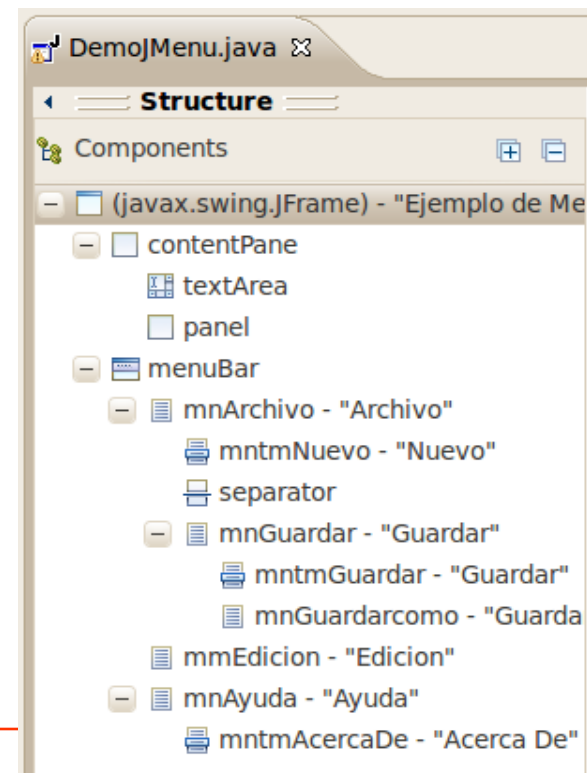
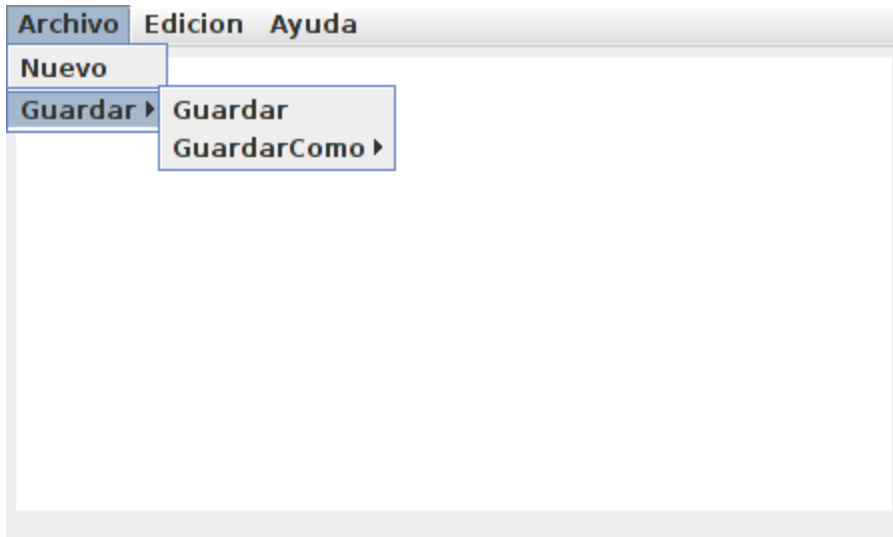
                int[] selectedIndex = numeroList.getSelectedIndices();
                StringBuffer sb = new StringBuffer("Valor:");
                // Get all the selected items using the indices
                for (int i=0; i<selectedIndex.length; i++) {
                    sb.append(numeroList.getModel().getElementAt(selectedIndex[i]).toString());
                }
                label.setText(sb.toString());
            }
        });
        add(btnActualizarValores, BorderLayout.SOUTH);
    }
}
```



- Uno de los principales componentes de una aplicación gráfica son los menús.
- Los menús han de ir en la ventana principal de la aplicación.
- Pueden ser de tres tipos:
 - ❖ Drop-Down:
 - ❖ Submenu: son aquellos que salen como un grupo de un elemento de menú.
 - ❖ Contextuales: Los menús contextuales, son aplicables a la región en la que está localizado el puntero del ratón

➤ Menu

- ❖ JMenuBar. Es la barra de menú principal (sin opciones)
- ❖ JMenu. Componente que se añade a JMenuBar y constituyen cada una de los opciones de la aplicación (Archivo, Edición, Ayuda)
- ❖ JMenuItem. Cada una de las opciones del JMenu (Nuevo, guardar,..)
- ❖ JSeparator. Componente separador entre varios JMenuItem



➤ Menu (DemoJMenu.java)

```
public DemoJMenu() {
    setTitle("Ejemplo de Menu");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);

    JMenuBar menuBar = new JMenuBar();
    setJMenuBar(menuBar); //Se establece el menu para el ContentPane

    JMenu mnArchivo = new JMenu("Archivo");
    menuBar.add(mnArchivo);

    JMenuItem mntmNuevo = new JMenuItem("Nuevo");
    mnArchivo.add(mntmNuevo);
    JSeparator separator = new JSeparator();
    mnArchivo.add(separator);

    JMenu mnGuardar = new JMenu("Guardar");
    mnArchivo.add(mnGuardar);
    JMenuItem mntmGuardar = new JMenuItem("Guardar");
    mnGuardar.add(mntmGuardar);
    JMenu mnGuardarcomo = new JMenu("GuardarComo");
    mnGuardar.add(mnGuardarcomo);

    JMenu mmEdicion = new JMenu("Edicion");
    menuBar.add(mmEdicion);

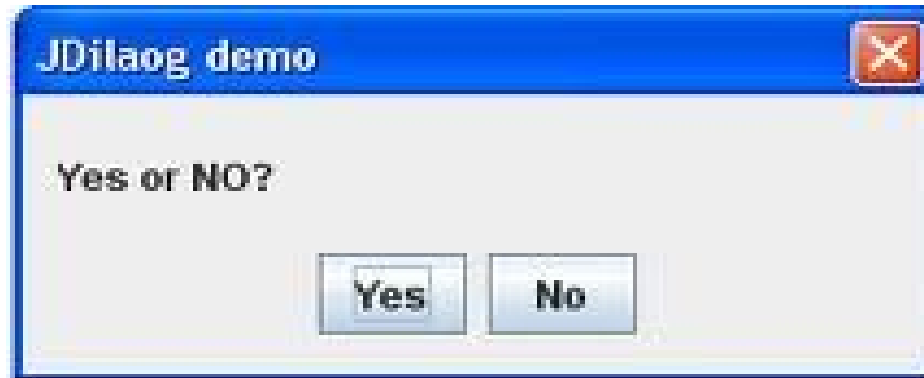
    JMenu mnAyuda = new JMenu("Ayuda");
    menuBar.add(mnAyuda);
    JMenuItem mntmAcercaDe = new JMenuItem("Acerca De");
    mnAyuda.add(mntmAcercaDe);

    JTextArea textArea = new JTextArea();
    contentPane.add(textArea, BorderLayout.CENTER);

    JPanel panel = new JPanel();
    contentPane.add(panel, BorderLayout.SOUTH);
}
```

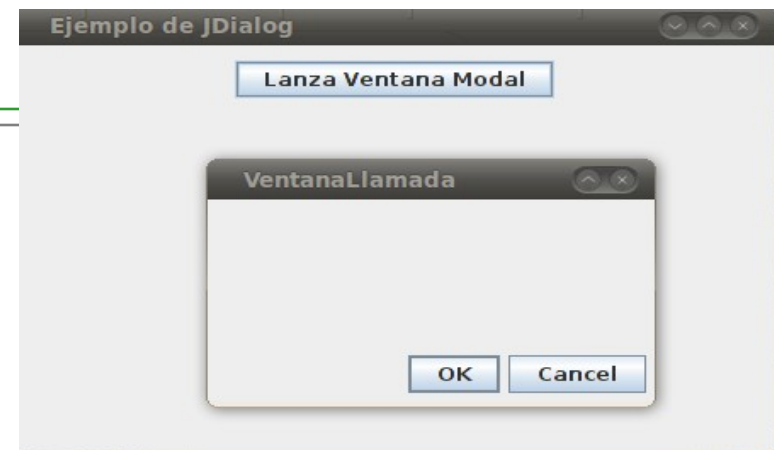
➤ JDialog

- ❖ Es una ventana secundaria de nuestra aplicación principal.
- ❖ Por el motivo anterior, admite otra ventana (JFrame o JDialog) como padre en el constructor.
 - ✓ Constructor: `JDialog(Frame, String, boolean)`
 - Frame es la ventana padre
 - String, el título
 - boolean indica si es modal o no (también con `setModal(Boolean)`)
- ❖ JDialog puede ser modal, todas las demás ventanas se deshabilitarán hasta que el usuario cierre el JDialog (al pulsar intro, al cerrar, al cancelar, etc).



➤ Jdialog (DemoJDialog.java)

```
public class DemoJDialog extends JFrame {  
  
    private JPanel contentPane;  
    DemoJDialog2 ventanaLlamada;  
    public DemoJDialog() {  
        setTitle("Ejemplo de JDialog");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setBounds(100, 100, 450, 300);  
        contentPane = new JPanel();  
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
        setContentPane(contentPane);  
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  
  
        JButton btnLanzaVentanaModal = new JButton("Lanza Ventana Modal");  
        btnLanzaVentanaModal.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent arg0) {  
                ventanaLlamada.setModal(true);  
                ventanaLlamada.setVisible(true);  
            }  
        });  
        contentPane.add(btnLanzaVentanaModal);  
        ventanaLlamada = new DemoJDialog2();  
    }  
}
```



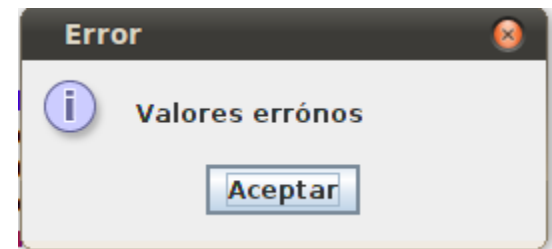
➤ JOptionPane

- ❖ Las ventanas de diálogos son bastante habituales en las aplicaciones gráficas
- ❖ Afortunadamente, Java tiene definido un conjunto de diálogos predefinidos que ayudan a esta labor
 - ✓ MensajesCortos (showMessageDialog())
 - ✓ Confirmación de Usuario (showConfirmDialog())
 - ✓ Petición de datos por el usuario (showInputDialog())
 - ✓ Seleccionar Fichero (JFileChooser())

➤ MensajesCortos (showMessageDialog())

```
JOptionPane.showMessageDialog (frame, mensaje,  
                                título_Frame, tipoMensaje)
```

```
JOptionPane.showMessageDialog(this,  
"Valores erróneos", "Error",  
JOptionPane.INFORMATION_MESSAGE);
```



➤ JOptionPane (DemoJDialog3.java)

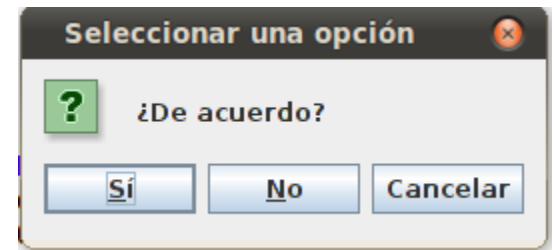
❖ Confirmación de Usuario (showConfirmDialog())

- ✓ Los valores de ese entero puede ser alguna de las constantes definidas en JOptionPane: YES_OPTION, NO_OPTION, CANCEL_OPTION, OK_OPTION, CLOSED_OPTION.

❖ Petición de datos por el usuario (showInputDialog())

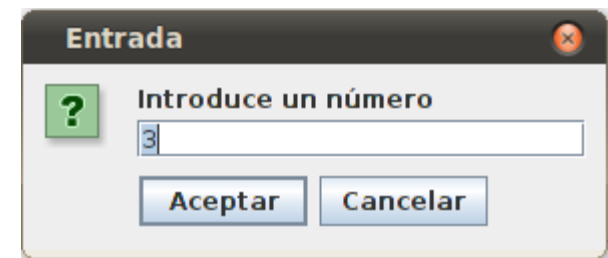
```
JOptionPane.showConfirmDialog (frame,mensaje)
```

```
int resultado = JOptionPane.showConfirmDialog(this, "¿De  
acuerdo?");  
  
if (JOptionPane.OK_OPTION == resultado)  
    System.out.println("Genial");  
else  
    System.out.println("Don't worry");
```



```
JOptionPane.showInputDialog (frame,mensaje, tipoMensaje)
```

```
String numero= JOptionPane.showInputDialog(  
    this,  
    "Introduce un número",  
    JOptionPane.QUESTION_MESSAGE);  
  
System.out.println("Valor"+numero);
```



- Las aplicaciones gráficas gestionan eventos. Tipos
 - ❖ **ComponentEvent**: El usuario mueva o redimensione un componente.
 - ❖ **FocusEvent**: Se cambie el foco de un componente.
 - ❖ **KeyEvent**: El usuario pulse una tecla.
 - ❖ **MouseEvent**: Se efectúe un movimiento con el ratón o haga un click.
 - ❖ **ContainerEvent**: Se añadan o eliminen componentes en el contenedor.
 - ❖ **WindowEvent**: Se realice algún tipo de operación con la ventana como abrirla y cerrarla.
 - ❖ **ActionEvent**: Se efectúe alguna acción sobre un componente, como por ejemplo: la pulsación de un botón.
 - ❖ **ItemEvent**: Se ha modificado el estado de algún elemento que pertenece al componente.
 - ❖ **TextEvent**: El contenido de texto de algún componente ha cambiado.
 - ❖ ...

- Para poder capturar todos los eventos, Java proporciona las interfaces de escucha (listeners) (xxxxListener).
- De este modo, para cada tipo de evento existe una interface de escucha. Ejemplo:
 - ❖ Para los eventos de tipo `ActionEvent` existe la interface escucha `ActionListener`.
 - ❖ Para los eventos de tipo `MouseEvent` existe la interface escucha `MouseListener`.
- Cada componente debe registrar qué listener quiere procesar (**registrar** listener) (`addXXXXXXListener`)
- Cada listener gestiona los eventos mediante un conjunto de métodos (métodos a implementar)

➤ Ejemplo

- ❖ Clase de evento: `ActionEvent`
- ❖ Objetos que lo emiten: `JButton`, `JMenuItem`, `JCheckBox`, `JRadioButton`, `JComboBox`, `TextField`
- ❖ Tipo de interfaz listener: `ActionListener`
- ❖ Métodos a implementar en clase listener: `actionPerformed` (`ActionEvent`)
- ❖ Método para registrar listener: `addActionListener` (`ActionListener`)

```
boton1.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        JOptionPane.showMessageDialog(null,  
            "Saludo", "Hola Luis",  
            JOptionPane.INFORMATION_MESSAGE);  
    }  
});
```

➤ Ejemplo

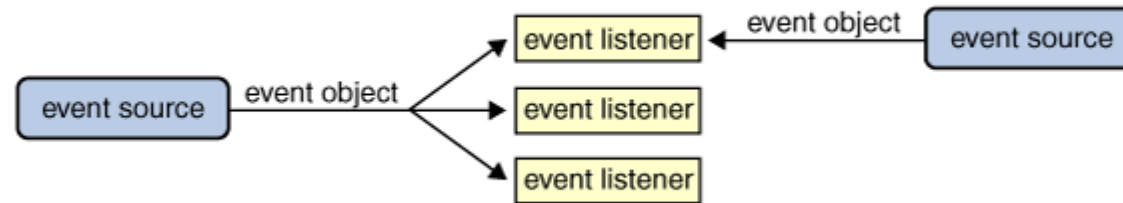
- ❖ Clase de evento: MouseEvent
- ❖ Objetos que lo emiten: JButton, JMenuItem, JcheckBox, JRadioButton, JComboBox, JTextField,....
- ❖ Tipo de interfaz listener: MouseListener
- ❖ Métodos a implementar en clase listener:
 - ✓ mouseClicked(MouseEvent), mouseEntered(MouseEvent), mouseExited(MouseEvent), mousePressed(MouseEvent), mouseReleased(MouseEvent)
- ❖ Método para registrar listener: addMouseListener (MouseListener)

```
frame = new JFrame("Hola");
frame.setBounds(100, 100, 450, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.addMouseListener(new MouseListener() {
    public void mouseClicked(MouseEvent e) {
        System.out.println("Evento Click en JFrame");
    }
    public void mouseEntered(MouseEvent e) {
        .....
    }
});
```

Gestión de Eventos

Evento	Interfaz Escucha	Métodos de Registrar	Componentes
ActionEvent	ActionListener	AddActionListener()	Jbutton, jList, jTextField, Jmenultem, jMenu,
ComponentEvent	ComponentListener	AddComponentLister()	Jbutton, jList, jTextField, Jmenultem, jMenu,
KeyEvent	KeyListener	AddKeyListener	Jcomponents y sus derivadas
MouseEvent	MouseListener	AddMouseListener	Jcomponents y sus derivadas
ItemEvent	ItemListener	addItemListener	JcheckBox, JComboBox, jList
TextEvent	TestListener	addTestListener	JtextComponent: incluyendo (jTextArea y jTextField)
....			

- Los eventos heredan de `java.util.EventObject` el método:
 - ❖ `Object getSource()` // retorna el componente que componente ha producido el evento
 - ❖ Útil cuando se quiere utilizar un manejador compartido entre varios componentes



➤ Manejadores compartidos (DemoEventosCompartido.java)

```
public DemoCompartirEventos() {
    ...
    btnHola = new JButton("Hola");
    btnHola.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            updateAction(e); //Compartido
        }
    });
    panel.add(btnHola);

    btnAdios = new JButton("Adios");
    btnAdios.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            updateAction(e); //Compartido
        }
    });
    panel.add(btnAdios);
    ...
}

public void updateAction(ActionEvent e) {

    Object e1=e.getSource();
    if (e1==btnHola){
        JOptionPane.showMessageDialog(null,"Hola");
    } else if (e1==btnAdios){
        JOptionPane.showMessageDialog(null,"Adios");
    }
}
```



➤ Adaptadores

- ❖ Cuando se desea escuchar algún tipo de evento se deben implementar todos los métodos de la Interface de escucha (listener interfase), para que nuestra clase no tenga que ser definida como abstracta. Para resolver este problema se hicieron los adaptadores.
- ❖ Son clases que implementan un listener, pero no realizan ningún tipo de operación.
- ❖ De esta forma cuando creamos una clase que hereda de MouseAdapter sólo implementaremos los métodos necesarios y que más nos interesen para gestionar los eventos.

➤ Adaptadores

- ❖ Por ejemplo, el adaptador de la clase escucha `MouseListener` es `MouseAdapter`.

```
public abstract class MouseAdapter implements MouseListener {  
    public void mouseClicked (MouseEvent e) {}  
    public void mousePressed (MouseEvent e) {}  
    public void mouseReleased (MouseEvent e) {}  
    public void mouseEntered (MouseEvent e) {}  
    public void mouseExited (MouseEvent e) {}  
}
```

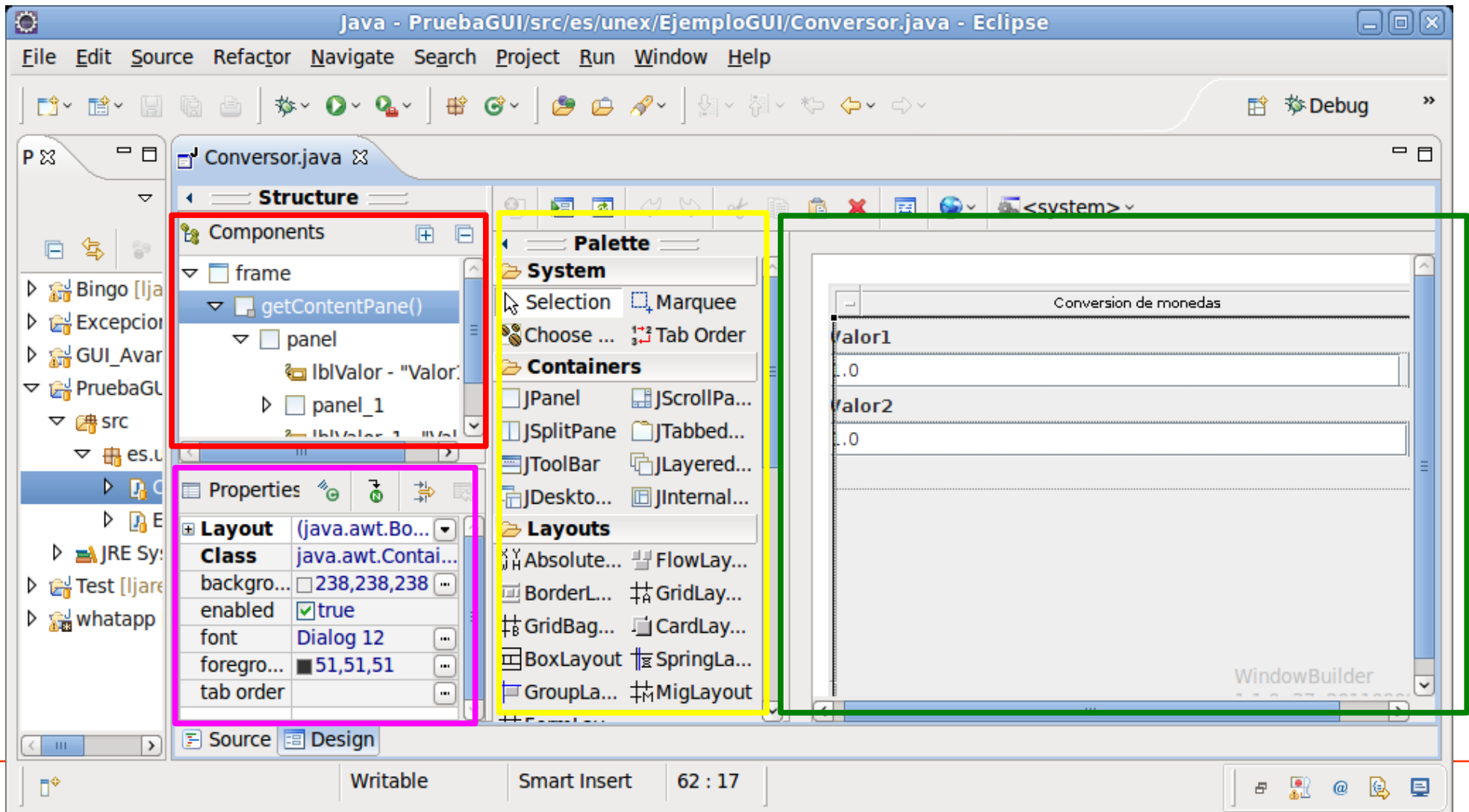
Event Listener interface	Event Listener Adapter
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
WindowListener	WindowAdapter

➤ Gráficos

- ❖ Los dibujos se realizan sobre un contexto gráfico representado por un objeto de la clase `Graphics`
- ❖ En el lenguaje Java es posible dibujar en una ventana redefiniendo el método **paint**.
- ❖ Este método se invoca automáticamente por el sistema cuando la ventana que incluye el aplique pasa a un primer plano.
- ❖ Existen un conjunto de métodos para pintar:
 - ✓ Líneas (`void drawLine(int x1, int y1, int x2, int y2)`)
 - ✓ Rectángulo (`void drawRect(int x, int y, int ancho, int alto)`)
 - ✓ Rectángulo Relleno (`void fillRect(int x, int y, int ancho, int alto)`)
 - ✓ Óvalos (`void drawOval(int x, int y, int ancho, int alto)`)
 - ✓ Cadenas (`void drawString(String str, int x, int y)`)
 - ✓ etc
- ❖ Este aspecto queda fuera del contenido de la asignatura, aunque se proporcionan algunos ejemplos para su consulta.

GUI en Eclipse. WindowBuilder

- Existe un plugin para Eclipse que facilita el trabajo con GUI
→ WindowsBuilder.



Bibliografía

- **Piensa en Java. 4ª Edición.** Bruce Eckel. Pearson Prentice Hall.
- Core Java 2. Autores Cay S. Horstmann Y Gary Cornell. Editorial Pearson Educación
- Aprenda Java como si estuviera en primero. Tecnum.
- URL:
 - ❖ The Java Tutorials: “Creating a GUI with JFC/Swing”.
<http://download.oracle.com/javase/tutorial/uiswing/layout/visual.html>
 - ❖ <http://www.chuidiang.com/java/>