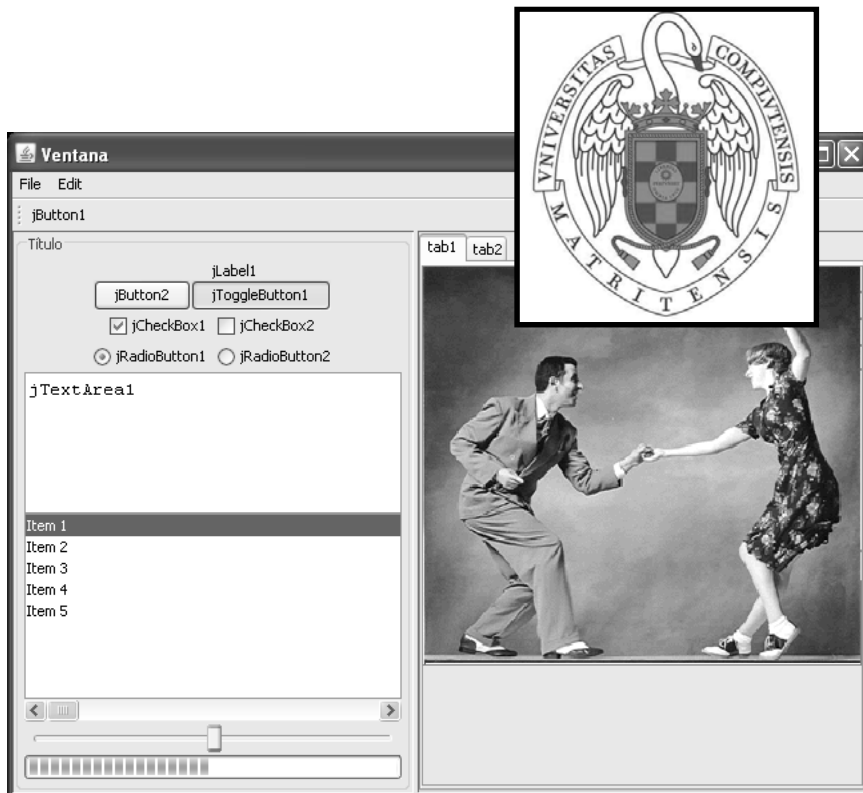


LPS: Interfaces gráficas de usuario con Swing



Federico Peinado
www.federicopeinado.es

Depto. de Ingeniería del Software e
Inteligencia Artificial
disia.fdi.ucm.es

Facultad de Informática
www.fdi.ucm.es

Universidad Complutense de Madrid
www.ucm.es

Interfaces gráficas de usuario

- ◉ Bibliotecas para programar interfaces gráficas de usuario (GUIs) en Java:
 - Abstract Windowing Toolkit (AWT), la primera que ofreció el lenguaje Java
 - Swing, muy popular y también integrada en Java
 - Standard Widget Toolkit (SWT), creada por IBM y usado en Eclipse
www.eclipse.org/swt
 - JavaFX, orientado a la web como *Flash* o *Silverlight*
javafx.com/
 - XML User Interface Language (XUL)
developer.mozilla.org/En/XUL
 - ...
- ◉ Herramientas WYSIWYG (*what you see is what you get*) para crear GUIs:
 - NetBeans Swing GUI Builder (y ahora también para *JavaFX*)
<http://netbeans.org/>
 - Eclipse WindowBuilder (para *Swing*, *SWT*, *RCP*, *XWT* y *GWT*... ¡muy prometedor!)
www.eclipse.org/windowbuilder/
 - JavaServer Faces, orientado a la web
www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.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
- Único estándar que ofrecía Java hasta la versión 1.1.5

Swing

◉ Swing (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“
 - A las aplicaciones se les puede dar una apariencia en función de la plataforma específica
- Nuevas funcionalidades
 - API de accesibilidad para personas con necesidades específicas

Creación de una interfaz gráfica de usuario

- ◉ Composición de la interfaz gráfica de la aplicación
 - Elección de un contenedor (ventana) en la que se incluyen el resto de los elementos gráficos de interacción
 - Diseño del interfaz gráfico añadiendo componentes gráficos de interacción (p.e. Botones, etiquetas, menús, ...)
 - Establecer la ubicación de los elementos manualmente o mediante un `LayoutManager`
 - Un `Layout Manager` gestiona la organización de los componentes gráficos de la interfaz
- ◉ Establecer los gestores de eventos para responder a las interacciones de los usuarios con la interfaz gráfica
- ◉ Visualizar la interfaz gráfica

Diseño y creación de la GUI

Tres elementos esenciales en la interfaz gráfica

Contenedores (containers)

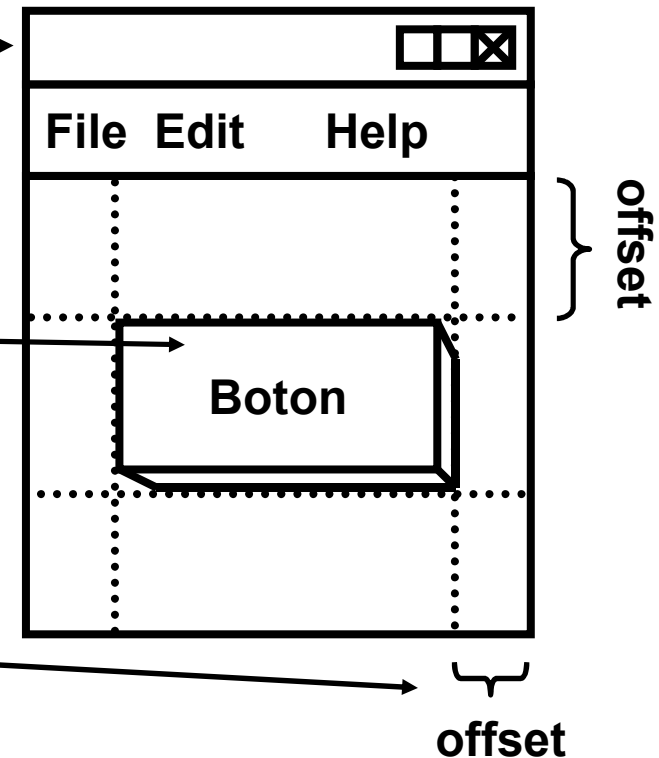
-- agrupan el resto de los elementos gráficos (e.g. ventana principal de la aplicación)

Componentes gráficos

-- elementos gráficos de interacción (e.g., botones)

Gestores de disposición (LayoutManagers)

-- algoritmo utilizado para organizar los elementos gráficos dentro del contenedor



Componentes de Swing

◉ Contenedores

- Contienen otros componentes (o contenedores)
 - Estos componentes se tienen que añadir al contenedor y para ciertas operaciones se pueden tratar como un todo
 - Mediante un gestor de diseño controlan la disposición (*layout*) de estos componentes en la pantalla
 - Ejemplo: JPanel, JFrame, JApplet

◉ Lienzo (clase *Canvas*)

- Superficie simple de dibujo

◉ Componentes de interfaz de usuario

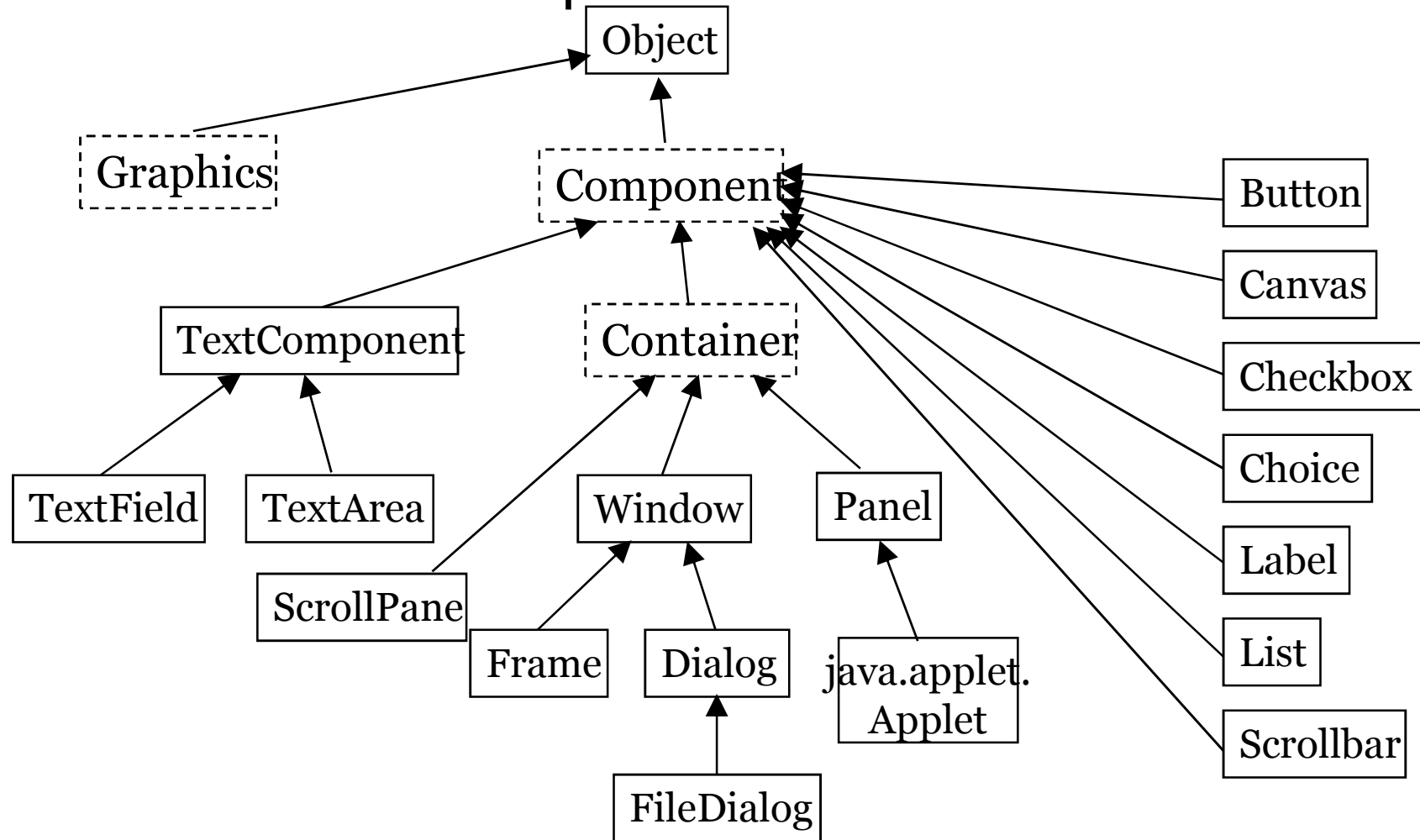
- botones, listas, menús, casillas de verificación, campos de texto, etc.

◉ Componentes de construcción de ventanas

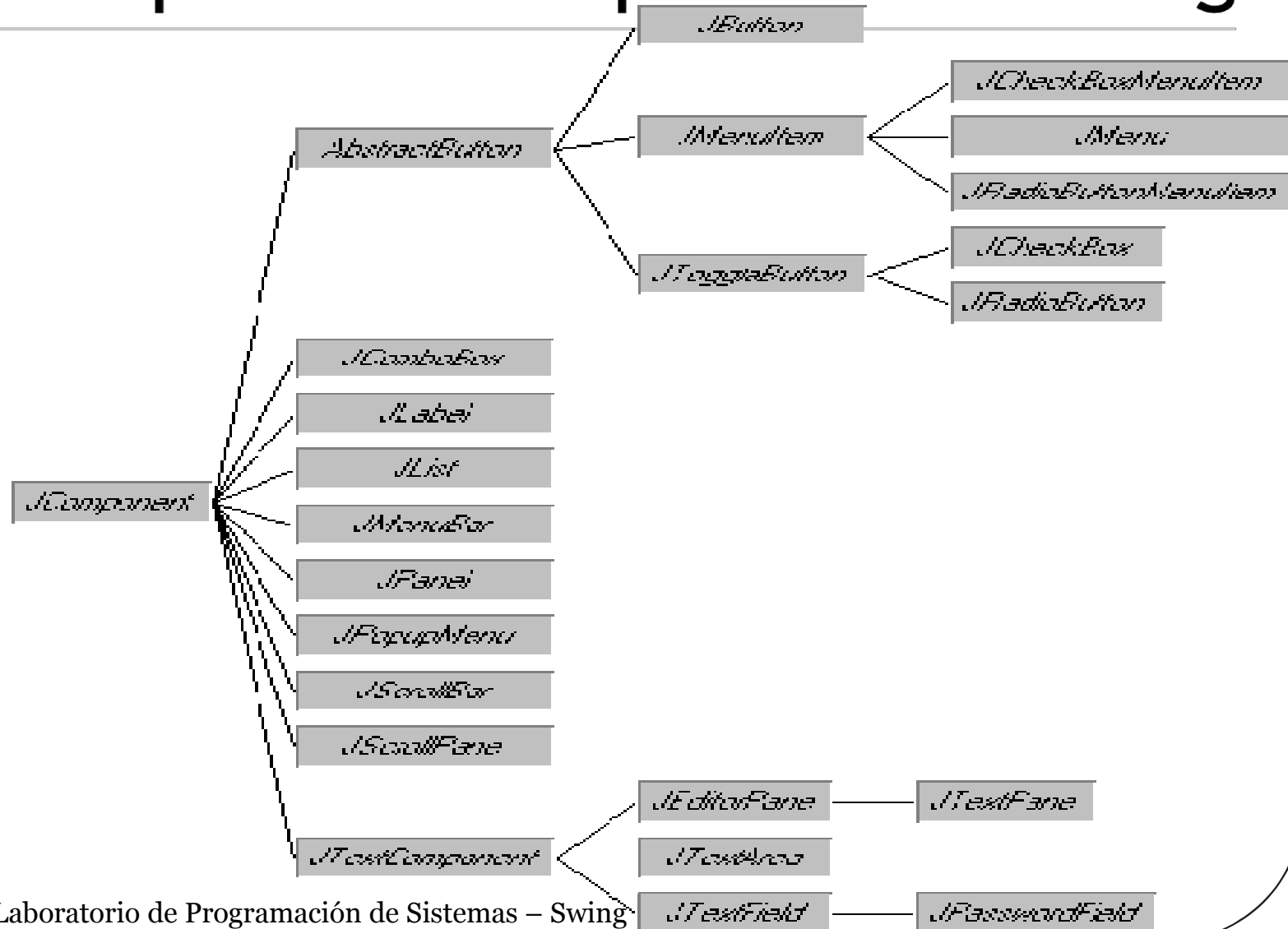
- ventanas, marcos, barras de menús, cuadros de diálogo

Jerarquía de componentes AWT

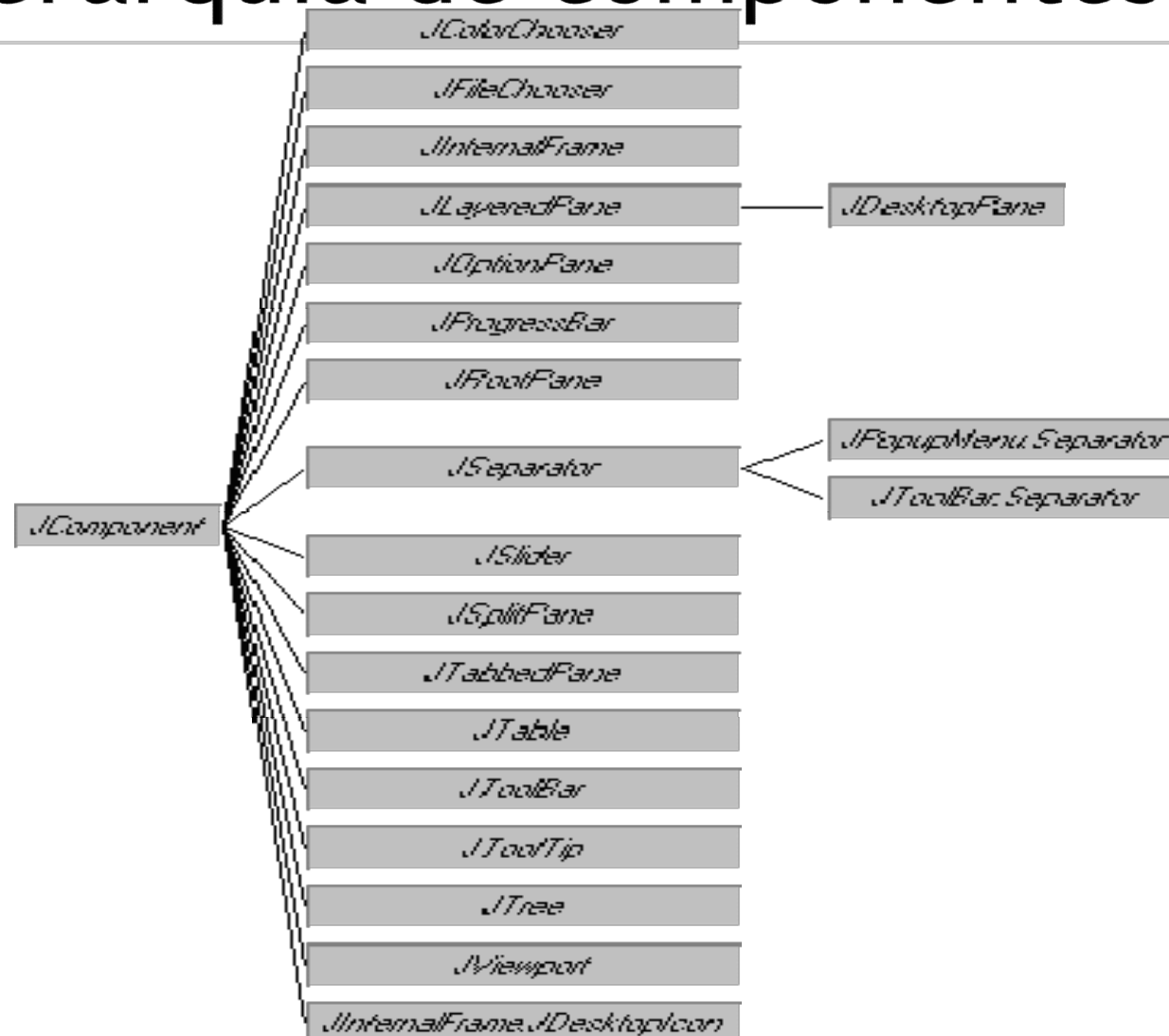
Jerarquía de clases



Jerarquía de componentes Swing

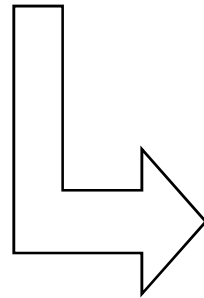


Jerarquía de componentes Swing



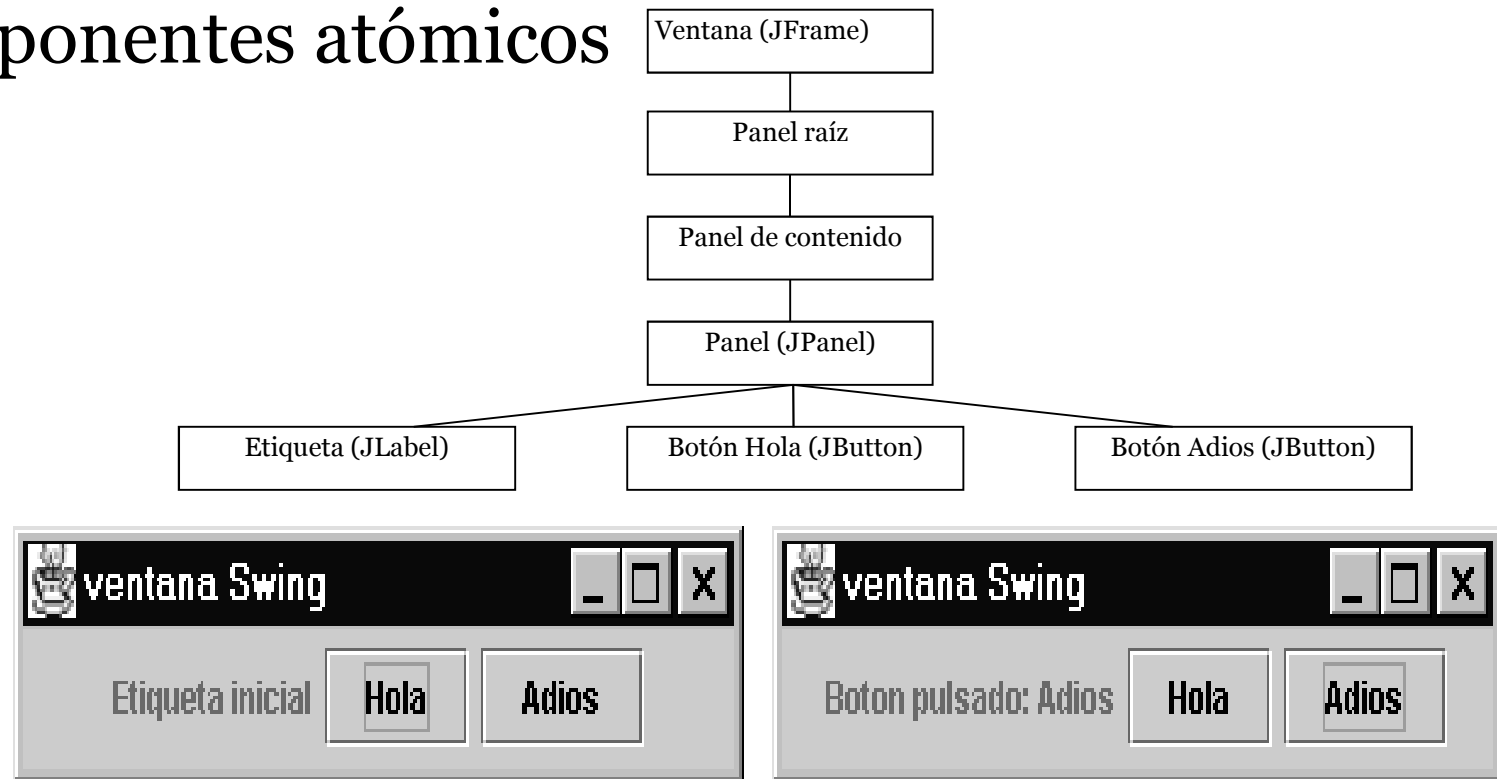
Ejemplo: GUI simple con un JFrame

```
import javax.swing.*;  
public class GUISimple extends JFrame {  
    public GUISimple () {  
        setSize(200, 100);  
        setVisible(true);  
    }  
    public static void main(String args[]) {  
        GUISimple ventana = new GUISimple();  
        ventana.setTitle("ventana tipo frame");  
    }  
}
```



Jerarquía de composición

- ◉ Contenedores de alto nivel
- ◉ Contenedores intermedios
- ◉ Componentes atómicos

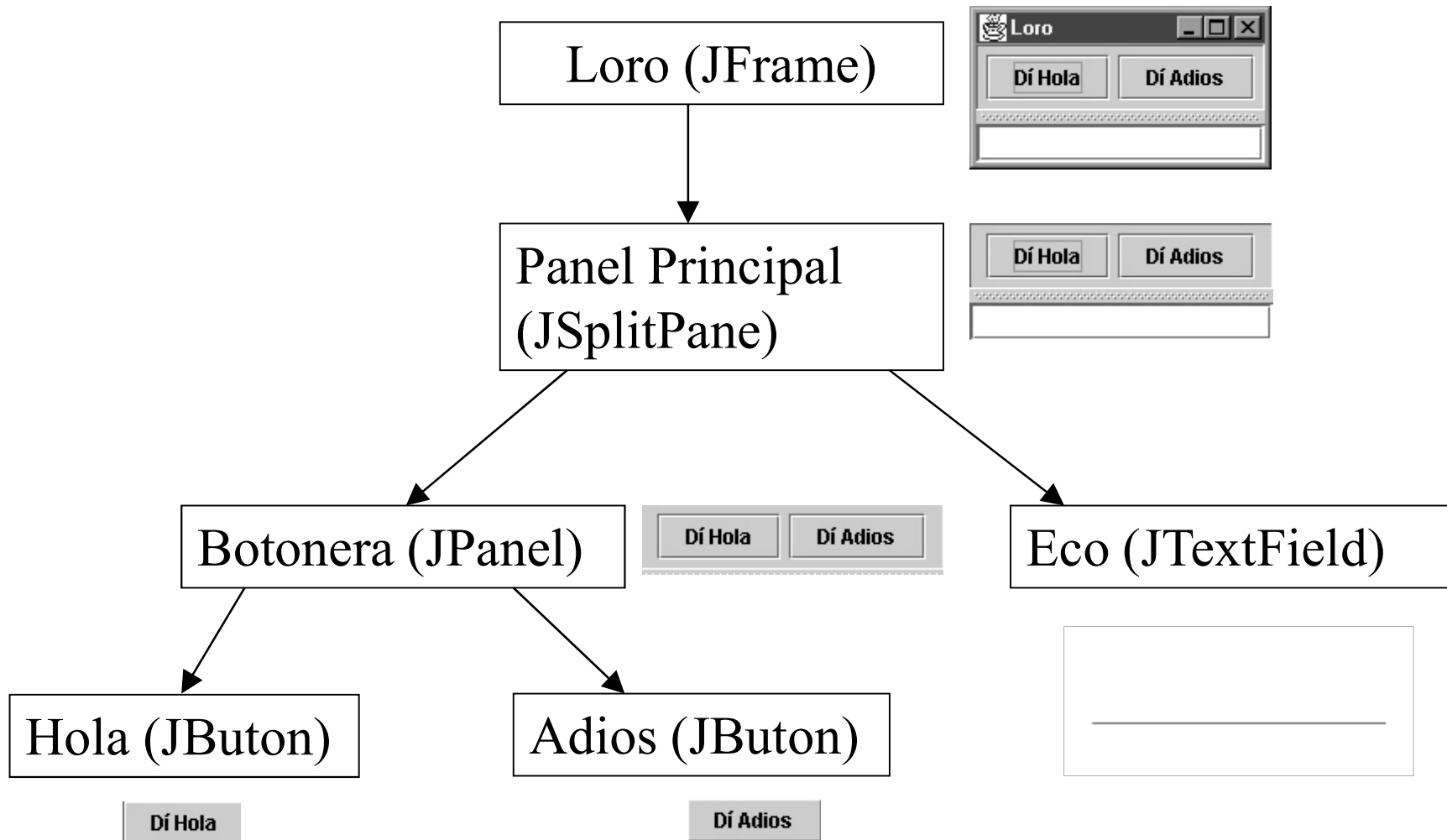


Ejemplo: Una aplicación Swing sencilla

- ◉ Al pulsar los botones, aparece texto en el cuadro blanco



Ejemplo: Una aplicación Swing sencilla: estructura

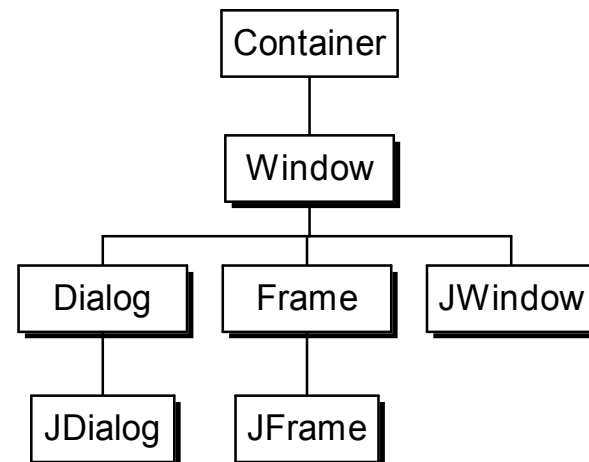


Clases básicas

- ◉ `java.awt.Component`
 - Esta clase abstracta define la funcionalidad básica de todos los componentes gráficos en Java
 - Proporciona, entre otros, los métodos de registro y eliminación de oyentes
- ◉ `java.awt.Container`
 - Clase abstracta que permite agrupar uno o varios componentes de forma que se puedan tratar como una unidad.
 - Proporciona métodos para añadir y eliminar componentes o para definir el tipo de presentación que se realiza de los componentes en la pantalla (mediante layout Managers)
- ◉ `javax.swing.JComponent`
 - Es la clase base de casi todos los componentes de interacción que incorpora Swing excepto los contenedores de alto nivel (p.e. *JFrame*).

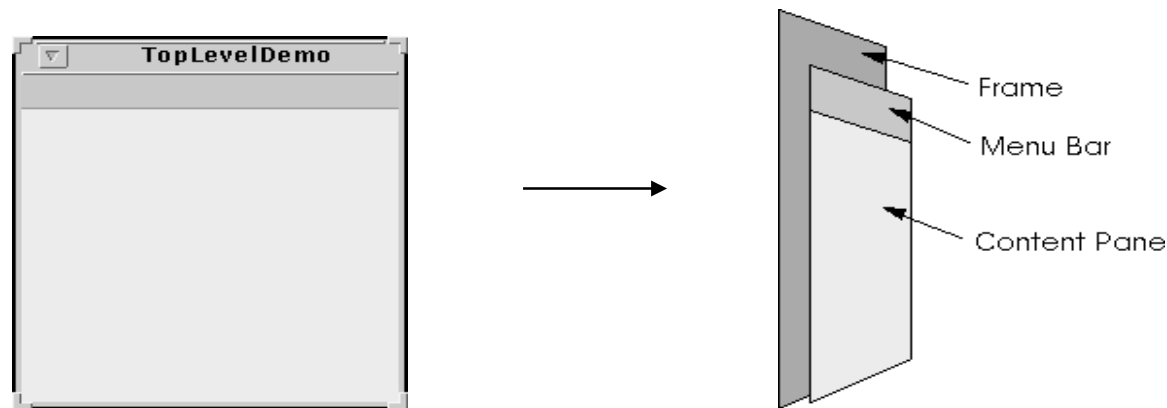
Contenedores de alto nivel

- ◉ `javax.swing.JFrame`
 - Habitualmente la clase *JFrame* se emplea para crear la ventana principal de una aplicación en Swing
- ◉ `javax.swing.JDialog`
 - Genera ventanas secundarias de interacción con el usuario
 - Cuadros de diálogo configurables y modificables
 - Son modales: el usuario no puede interactuar con otra ventana hasta que no cierre la actual



JFrame

- ◉ La clase JFrame proporciona una ventana principal de aplicación con su funcionalidad normal (p.e. borde, título, menús) y un panel de contenido.
- ◉ Los contenidos se añaden en el panel de contenidos (content pane) accesible a través del método `getContentPane` (por defecto, un objeto de tipo `JPane`, aunque puede cambiarse con `setContentPane`).
- ◉ La barra de menú puede fijarse con `setJMenuBar`.



JDialog

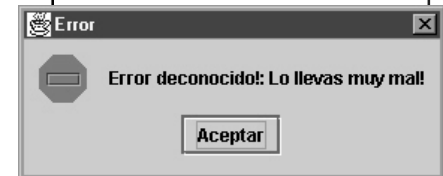
- ◉ La clase *JDialog* es la clase raíz de las ventanas secundarias que implementan cuadros de diálogo en Swing.
 - dependen de una ventana principal (normalmente *JFrame*) y si la ventana principal se cierra, se maximiza o minimiza las ventanas secundarias realizan la misma operación de forma automática.
- ◉ Las ventanas modales bloquean la interacción del usuario con otras ventanas.
 - Se utilizan sólo cuando hay que garantizar que el usuario recibe un mensaje o proporciona una información que es necesaria

Cuadro de diálogo estándar: JOptionPane

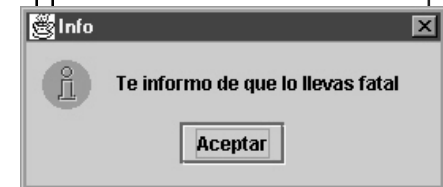
- ◉ Tipos de cuadros de dialogo más habituales
 - Message para informar al usuario sobre algún hecho relevante
 - Confirm para realizar una pregunta al usuario con las posibilidades básicas de respuesta de si, no o cancelar.
 - Input para solicitar una entrada del usuario
 - Option permite crear una ventana personalizada de cualquiera de los tipos anteriores
- ◉ Todos los cuadros de diálogo que implementa JOptionPane son modales

Cuadro de diálogo estándar: JOptionPane

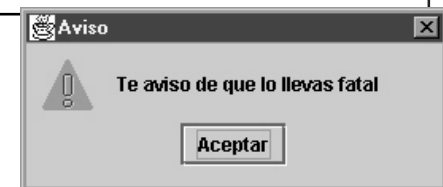
```
JOptionPane.showMessageDialog(this, // La ventana  
padre.  
    "Error desconocido!: Lo llevas muy mal!", //El  
mensaje.  
    "Error", // El título de la ventana de diálogo.  
    JOptionPane.ERROR_MESSAGE // El tipo de mensaje  
);
```



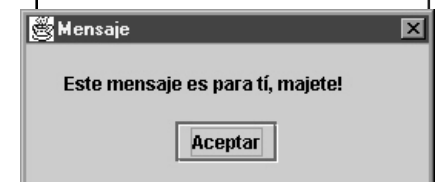
```
JOptionPane.showMessageDialog(this,  
"Te informo de que lo llevas fatal", "Info",  
JOptionPane.INFORMATION_MESSAGE);
```



```
JOptionPane.showMessageDialog(this,  
"Te aviso de que lo llevas fatal", "Aviso",  
JOptionPane.WARNING_MESSAGE);
```

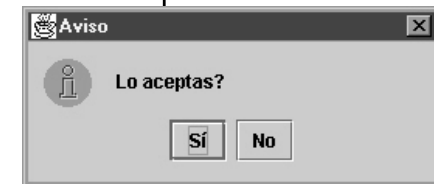


```
JOptionPane.showMessageDialog(this,  
    Este mensaje es para tí, majete!", "Mensaje",  
JOptionPane.PLAIN_MESSAGE);
```

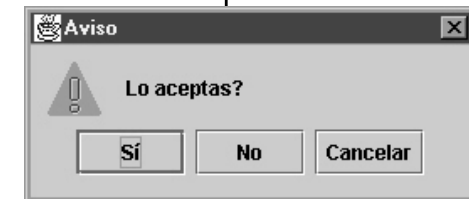


Cuadro de diálogo estándar: JOptionPane

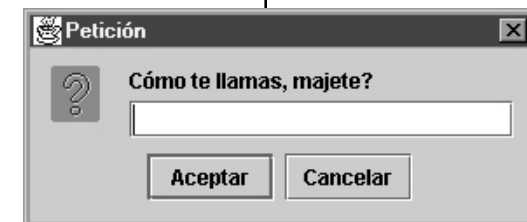
```
int seleccionada =
    JOptionPane.showConfirmDialog(this,
        "Lo aceptas?", "Aviso",
        JOptionPane.YES_NO_OPTION, // Configuración del mensaje
        JOptionPane.INFORMATION_MESSAGE);
switch(seleccionada) {
    case JOptionPane.YES_OPTION: ... // tratar SI
    case JOptionPane.NO_OPTION: .. // tratar NO
    case JOptionPane.CLOSED_OPTION: .. // tratar ventana cerrada
}
```



```
int seleccionada =
    JOptionPane.showConfirmDialog(this,
        "Lo aceptas?", "Aviso",
        JOptionPane.YES_NO_CANCEL_OPTION,
        JOptionPane.WARNING_MESSAGE);
... // los posibles valores devueltos son los anteriores y
... // JOptionPane.CANCEL_OPTION
```

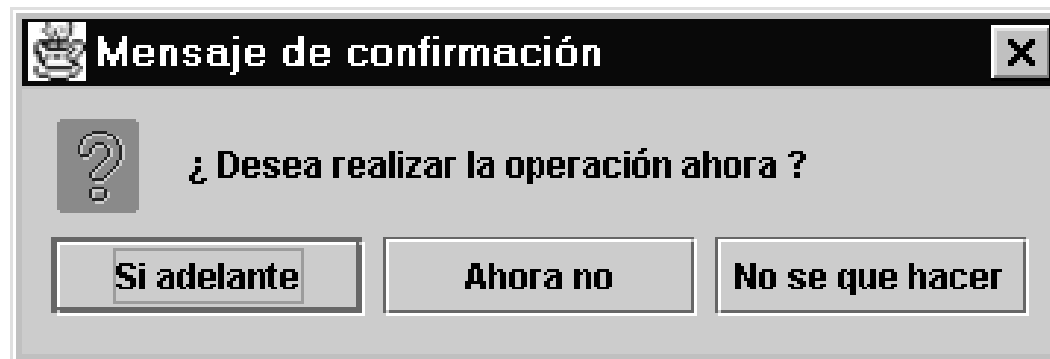


```
String nombre = JOptionPane.showInputDialog(this,
    "Cómo te llamas, majete?",
    "Petición", JOptionPane.QUESTION_MESSAGE
);
// ... procesar entrada
```



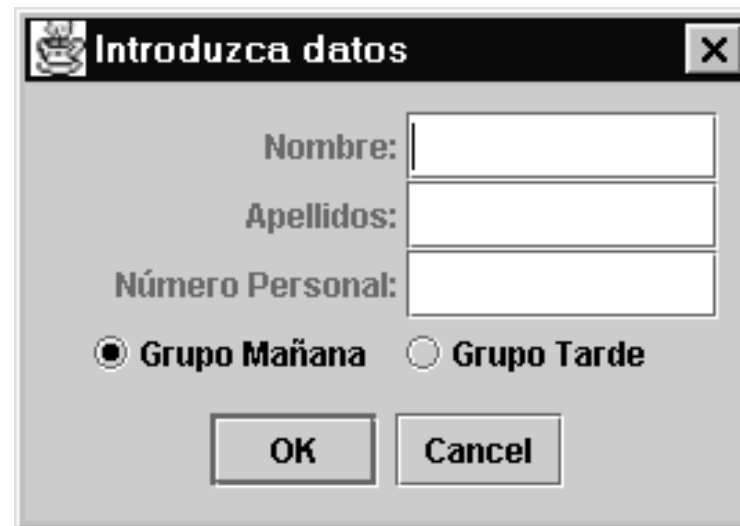
Cuadro de diálogo estándar: JOptionPane

```
// cuadro de opción personalizado
Object[] textoOpciones = {"Si adelante", "Ahora no",
                          "No se que hacer"};
int opcion = JOptionPane.showOptionDialog(ventana,
    "? Desea realizar la operación ahora ?",
    "Mensaje de confirmación",
    JOptionPane.YES_NO_CANCEL_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null, //utilizar el icono predeterminado
    textoOpciones,
    textoOpciones[0]); //botón predeterminado
}
```



Petición de datos con JOptionPane

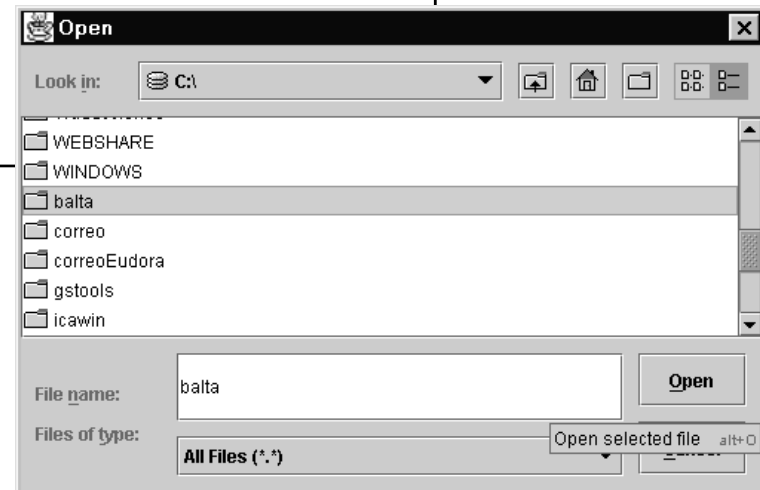
```
if (JOptionPane.showConfirmDialog(this, panel
    ,"Introduzca datos"
    ,JOptionPane.OK_CANCEL_OPTION
    ,JOptionPane.PLAIN_MESSAGE)
    == JOptionPane.OK_OPTION) {
    String nombre = campoNombre.getText();
    String apellidos = campoApellidos.getText();
    int numPer=Integer.parseInt(campoNP.getText());
}
```



The screenshot shows a standard Java Swing dialog box with a title bar that says "Introduzca datos". Inside the dialog, there are three text input fields. The first is labeled "Nombre:", the second "Apellidos:", and the third "Número Personal:". Below these fields, there are two radio buttons: "Grupo Mañana" (which is selected) and "Grupo Tarde". At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

JFileChooser

```
import javax.swing.*;
// se crea el selector de ficheros
JFileChooser selector = new JFileChooser();
// solo posibilidad de seleccionar directorios
selector.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
// se muestra; se comprueba si el usuario acepta o cancela
int opcion = selector.showOpenDialog(null);
if (opcion == JFileChooser.APPROVE_OPTION) {
    //obtenemos el fichero o directorio seleccionado
    File archivo = selector.getSelectedFile();
    System.out.println("archivo seleccionado: " + archivo);
}
else
    System.out.println("operacion cancelada ");
```



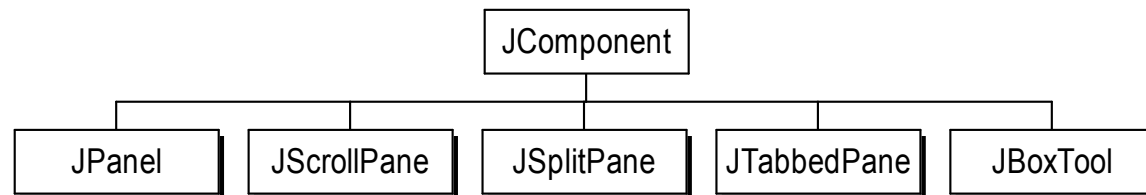
Contenedores intermedios

◉ JPanel

- Agrupa a otros componentes
- No tiene presentación gráfica pero se le pueden añadir bordes o cambiar el color de fondo

◉ JScrollPane

- Incluye barras de desplazamiento



Panel con datos del usuario

```
JPanel panel = new JPanel(new GridLayout(4,2));
JLabel etiquetaNombre = new JLabel("Nombre: ", JLabel.RIGHT);
JTextField campoNombre = new JTextField();
panel.add(etiquetaNombre);
panel.add(campoNombre);
JLabel etiquetaApellidos = new JLabel("Apellidos: ", JLabel.RIGHT);
JTextField campoApellidos = new JTextField();
panel.add(etiquetaApellidos);
panel.add(campoApellidos);
JLabel etiquetaNP = new JLabel("Número Personal: ", JLabel.RIGHT);
JTextField campoNP = new JTextField();
panel.add(etiquetaNP);
panel.add(campoNP);
ButtonGroup grupoBotones = new ButtonGroup();
JRadioButton mañana = new JRadioButton("Grupo Mañana", true);
JRadioButton tarde = new JRadioButton("Grupo Tarde");
grupoBotones.add(mañana);
grupoBotones.add(tarde);
panel.add(mañana);
panel.add(tarde);
```

**Panel que agrupa,
tres etiquetas, tres
campos de texto y dos
botones de radio**



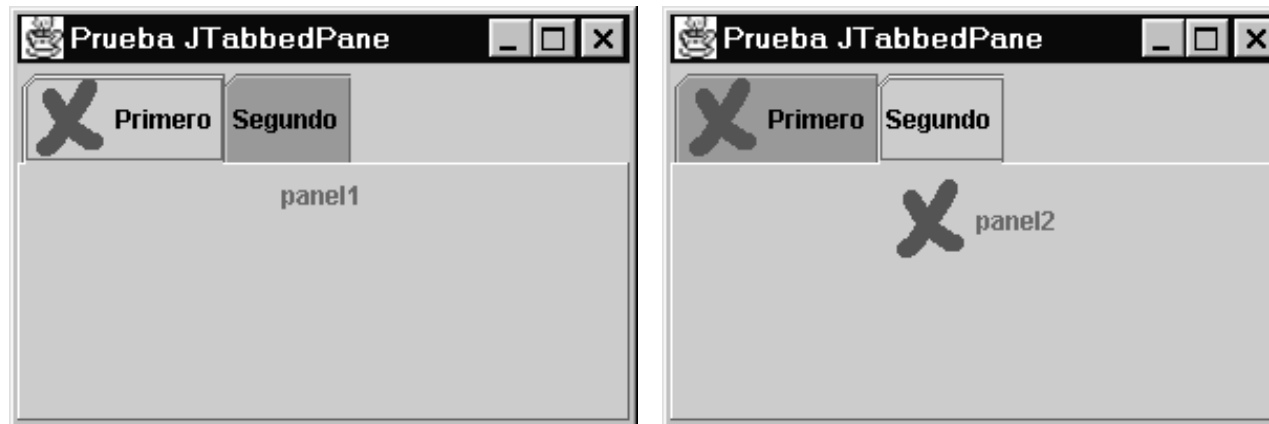
JSplitPanel

- ◉ Es un contenedor que gestiona dos componentes (normalmente paneles) colocados vertical u horizontalmente y diferenciados por un separador que puede ser reposicionado por el usuario.
- ◉ Hay que hacer una asignación inicial del espacio dedicado a cada parte



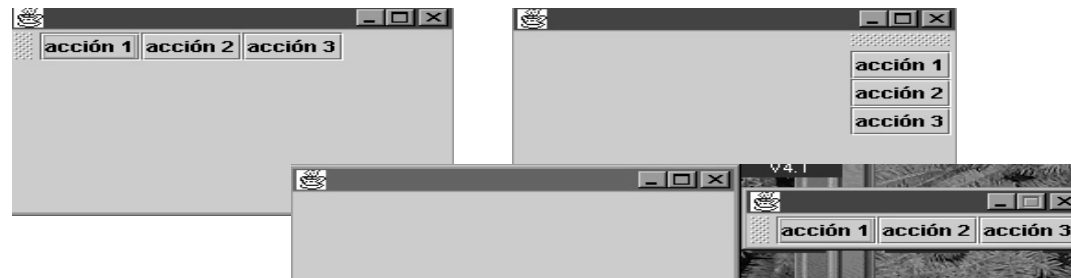
JTabbedPane

- ◉ El panel con solapas un contenedor que gestiona varios componentes (o grupos de componentes aunque habitualmente son paneles) como una pila de fichas
 - Sólo uno de los componentes es visible en cada momento
 - El usuario puede decidir cual de los componentes se visualiza seleccionando la solapa o lengüeta correspondiente a dicho componente.



JToolBar

- ◉ Implementa una barra de herramientas, formada normalmente por botones o controles que incluyen iconos, y que aparecen organizados como una fila o una columna dependiendo de la zona de la pantalla donde se coloque
 - Una barra de herramientas que puede cambiarse de situación por los diferentes bordes de su contenedor, e, incluso, llevarse fuera (este comportamiento puede variarse: método `setFloatable`).
 - Las herramientas suelen ser (aunque no necesariamente) botones.
 - Útil para proporcionar controles que dan acceso rápido a acciones, normalmente disponibles a través de menú.
 - Mediante el método `addSeparator` es posible añadir separadores.
 - JToolBar es, en realidad, una especialización de `Box`.



Iconos y etiquetas

◎ Iconos

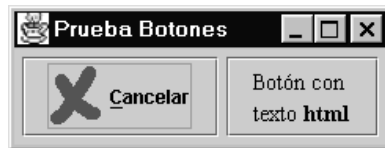
- Elementos gráficos que se pueden añadir a los componentes

◎ Etiquetas

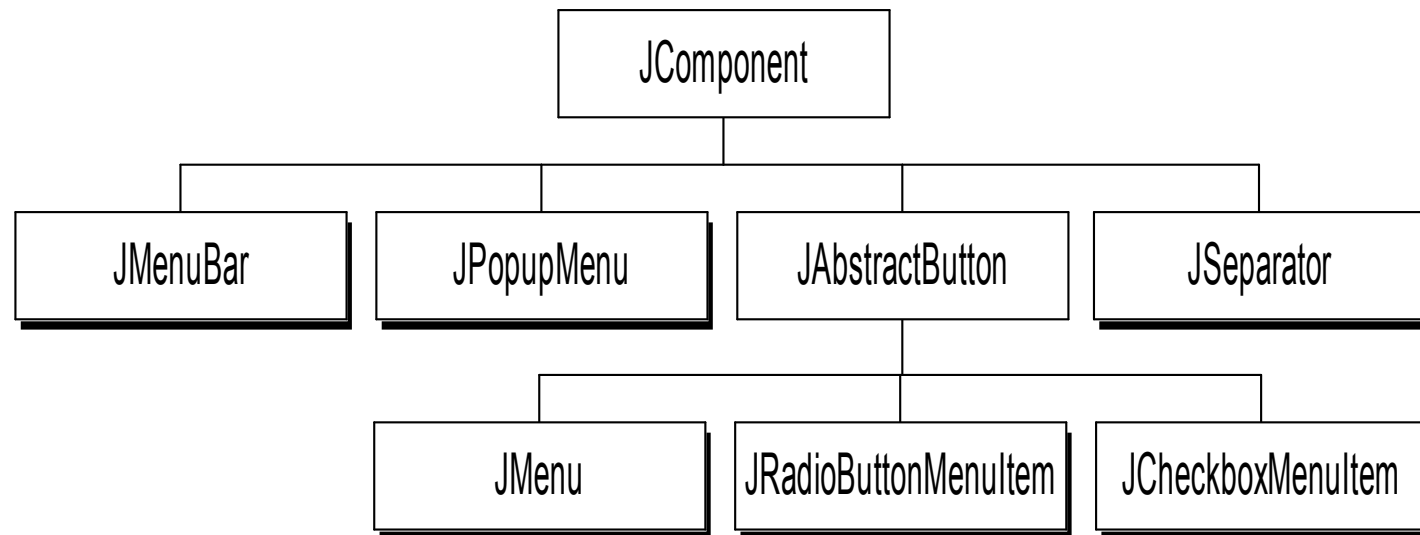
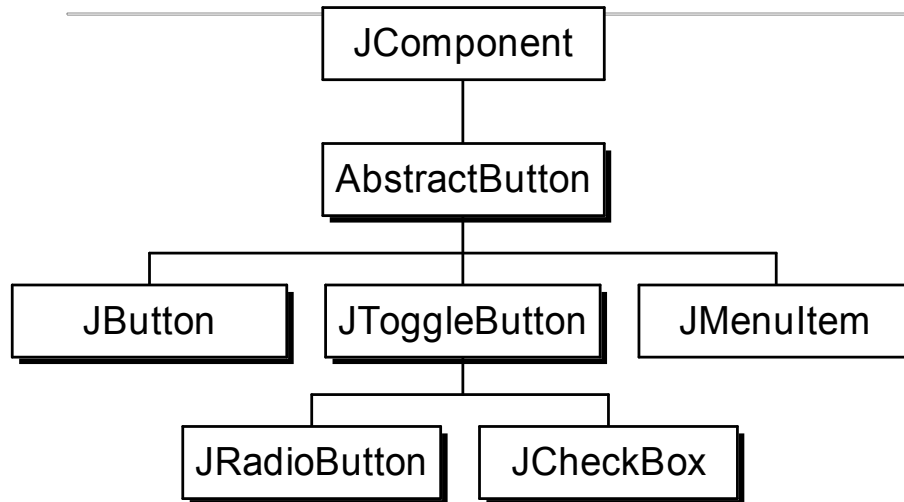
- Elementos para mostrar información
- Una etiqueta puede incluir un icono
- El texto puede estar escrito con formato HTML En este caso el texto debe empezar por “<html>”
- Es posible cambiar dinámicamente el texto de la etiqueta con setText

Botones

- ◉ Los botones, junto con los menús, son los controles más típicos
- ◉ Existen diferentes tipos (todos ellos especializan a `AbstractButton`)
 - `JButton`: Botón aislado. Puede pulsarse, pero su estado no cambia
 - `JToggleButton` : Botón seleccionable. Cuando se pulsa el botón, su estado pasa a seleccionado, hasta que se pulsa de nuevo (entonces se deselecciona). `isSelected` permite chequear su estado
 - `JCheckBox` : Especialización de `JToggleButton` que implementa una casilla de verificación. Botón con estado interno, que cambia de apariencia de forma adecuada según si está o no está seleccionado
 - `JRadioButton`: Especialización de `JToggleButton` que tiene sentido dentro de un mismo grupo de botones (`ButtonGroup`) que controla que solamente uno de ellos está seleccionado (importante: `ButtonGroup` es únicamente un controlador, no un componente)



Botones y menús

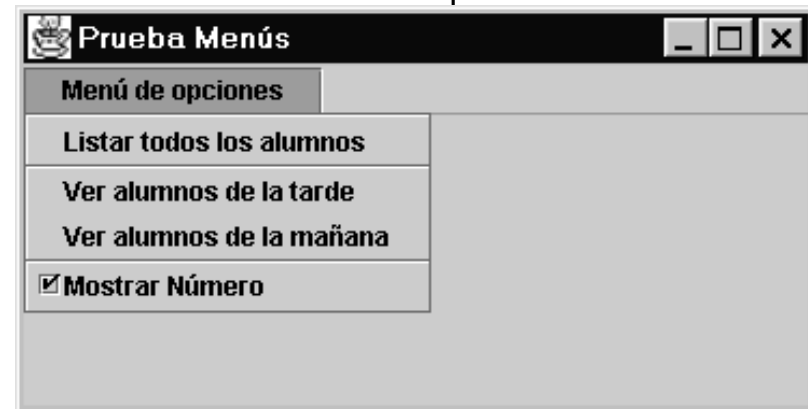


Menús

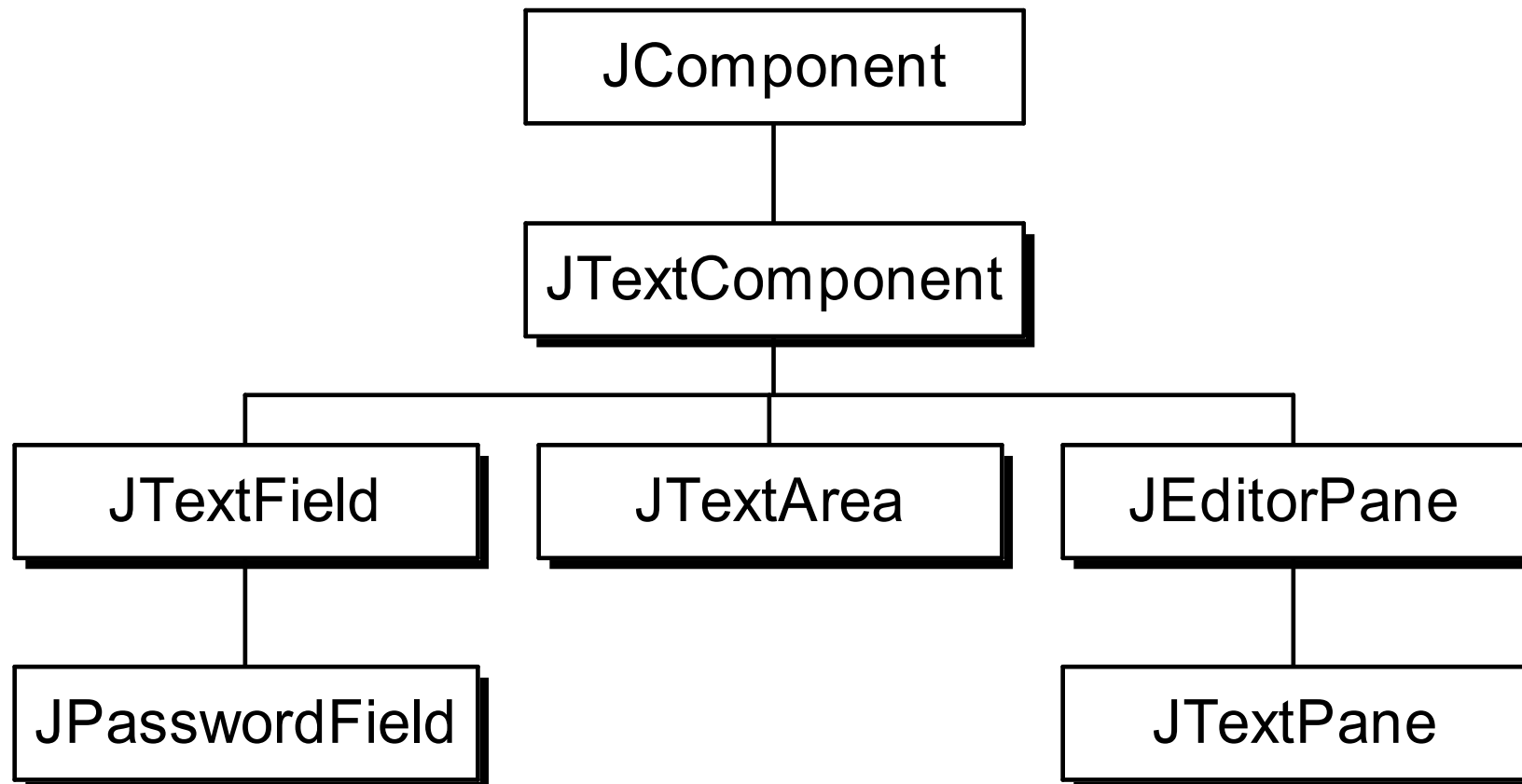
- ◉ La creación de una barra de menús básica supone:
 - Crear un objeto de tipo `JMenuBar`
 - Para cada entrada, crear un objeto de tipo `JMenu`
 - Incluir objetos de tipo `JMenuItem` en el menú. Esto puede incluir menús anidados
 - Asociar a los items acciones apropiadas (notifican eventos semánticos de tipo `ActionEvent`, ya que, en realidad, especializan a `AbstractButton`)
- ◉ Con `setJMenuBar` es posible añadir una barra de menús a una ventana (`JFrame`)
- ◉ En una GUI, muchas veces existen controles ligados a la misma acción (eg. un botón que hace lo mismo que un item de un menú). En este caso ambos controles pueden compartir el mismo oyente (y es aconsejable hacerlo así)
- ◉ El diseño de una barra de menús debe ser consistente (poner opciones semánticamente relacionadas juntas). También pueden usarse separadores

Ejemplo de menú

```
import javax.swing.*;  
JMenuBar barraMenu = new JMenuBar();  
JMenu menuOpciones = new JMenu("Menú de opciones");  
JMenuItem listar = new JMenuItem("Listar todos los alumnos");  
menuOpciones.add(listar);  
// separador  
menuOpciones.add(new JSeparator());  
JMenuItem listarTarde = new JMenuItem("Ver alumnos de la tarde");  
menuOpciones.add(listarTarde);  
JMenuItem listarMañana = new JMenuItem("Ver alumnos de la  
mañana");  
menuOpciones.add(listarMañana);  
menuOpciones.add(new JSeparator());  
JCheckBoxMenuItem verNumero =  
    new JCheckBoxMenuItem("Mostrar Número");  
menuOpciones.add(verNumero);  
barraMenu.add(menuOpciones);  
// establecer como barra de menús  
// en contenedor de alto nivel  
setJMenuBar(barraMenu);
```



Elementos de manejo de texto



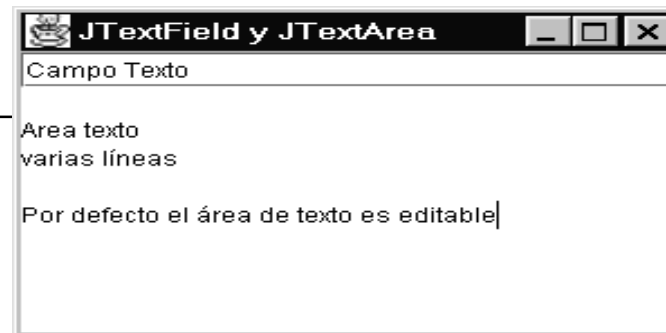
JTextField

- ◉ Permite incluir un control para introducir una línea de texto
- ◉ JPasswordField es análogo a JTextField, salvo que no se visualiza lo que se escribe
- ◉ Con `setEditable` es posible establecer si puede escribirse o no en el campo de texto
- ◉ Notifica un `ActionEvent` cuando el usuario indica que la línea de texto está completa (normalmente pulsando retorno de carro)
- ◉ Mediante el método `getText` es posible consultar el texto escrito (con `setText` puede fijarse desde el programa dicho texto)

JTextArea

- ◉ Una forma simple de editar/visualizar varias líneas de texto
- ◉ Con append es posible añadir texto. También existe getText y setText (JTextField y JTextArea heredan ambos de JTextComponent)

```
public class PanelTexto extends JPanel {  
    final String FIN = "\n";  
    public PanelTexto() {  
        setLayout(new BorderLayout());  
        JTextField campoTexto = new JTextField("Campo Texto");  
        add(campoTexto, BorderLayout.NORTH);  
        String texto = FIN+"Area texto"+FIN+"varias líneas";  
        JTextArea areaTexto = new JTextArea(texto);  
        add(areaTexto, BorderLayout.CENTER);  
    }  
}
```

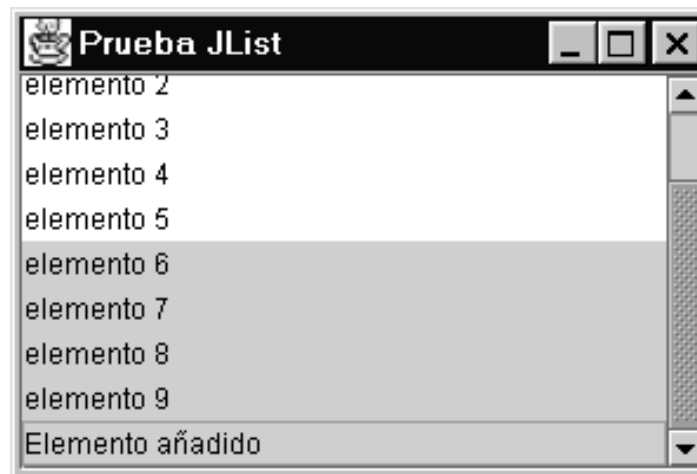


JList

- ◉ La clase *JList* implementa una lista de elementos que se presenta en forma de columna
- ◉ En esta lista el usuario puede realizar la selección de uno (comportamiento por defecto) o varios de sus elementos
- ◉ El contenido de una lista viene dado por su modelo de datos que debe implementar la interfaz Java *ListModel*
 - *DefaultListModel* clase que da una implementación por defecto del modelo de datos de lista

JList

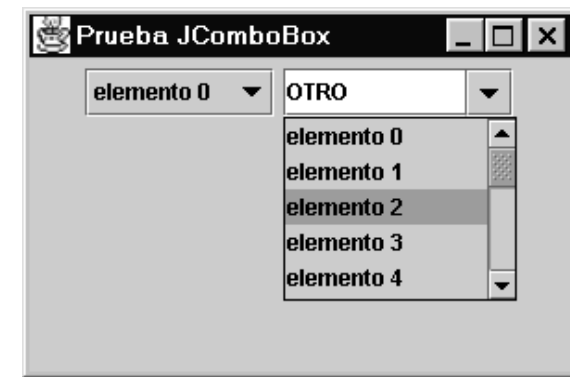
```
DefaultListModel modeloDatos = new DefaultListModel();  
for (int ind=0; ind<10; ind++)  
    modeloDatos.addElement("elemento "+ ind);  
JList lista = new JList(modeloDatos);  
// se añade un nuevo elementos al modelo  
modeloDatos.addElement("Elemento añadido");  
lista.setSelectionMode(  
    ListSelectionMode.SINGLE_INTERVAL_SELECTION);  
JScrollPane panelDesplazamiento = new JScrollPane(lista);
```



JComboBox

- ◉ Esta clase implementa un cuadro combinado desplegable, en el que se agrupan las funcionalidades de una lista y un campo de texto

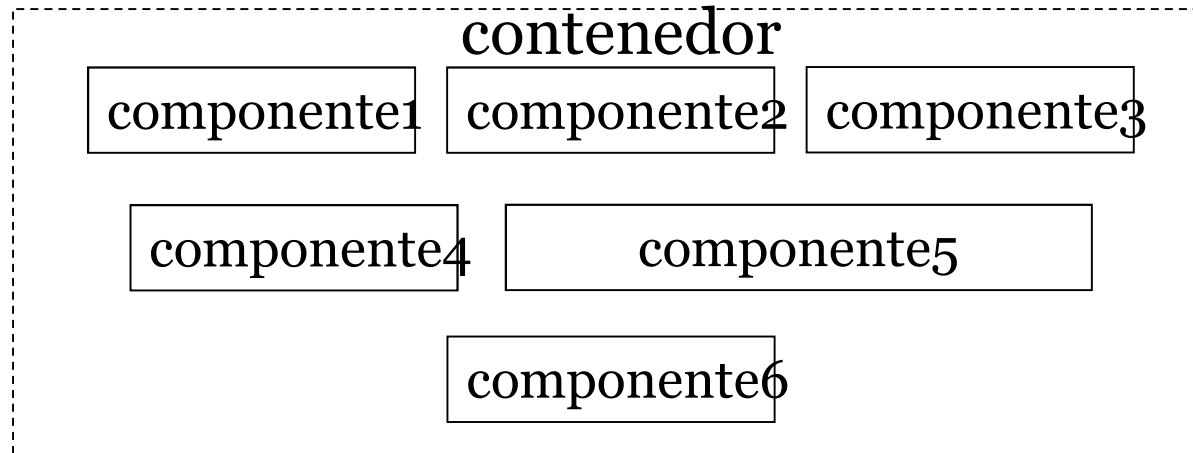
```
public class PanelComboBox extends JPanel {
    String[] listaElementos = new String[15];
    public PanelComboBox() {
        for (int ind=0; ind<listaElementos.length; ind++)
            listaElementos[ind]= new String("elemento "+ ind);
        JComboBox combo1 = new JComboBox(listaElementos);
        JComboBox combo2 = new JComboBox(listaElementos);
        // el segundo se hace editable
        combo2.setEditable(true);
        combo2.setSelectedItem("OTRO");
        // sólo se visualizan 5 filas
        combo2.setMaximumRowCount(5);
        add(combo1);
        add(combo2);
    }
}
```



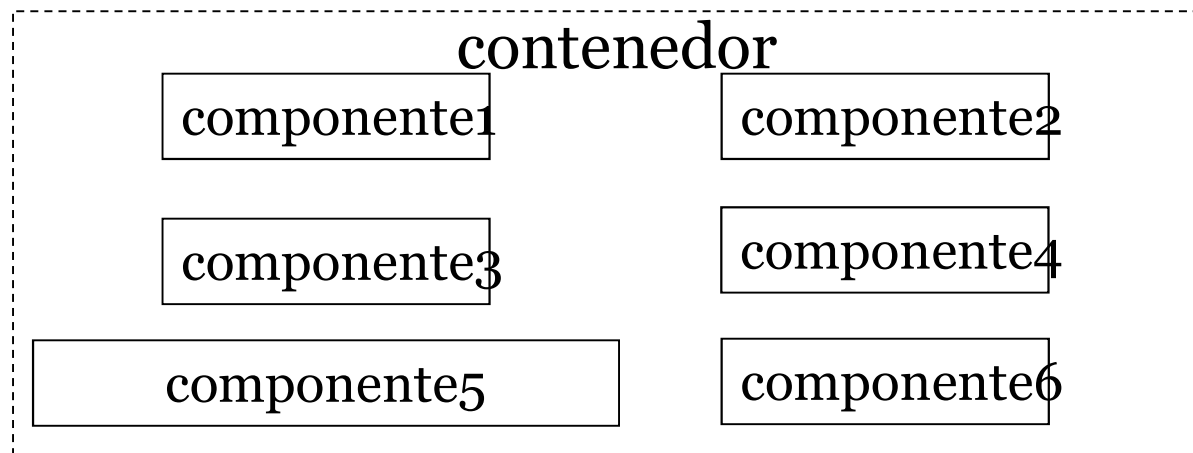
Administrador de diseño

- ⊙ Layout Manager
- ⊙ Cómo se colocan los componentes (usando el método *add*) depende de la composición (*layout*)
- ⊙ Tipos de diseños o composiciones
 - FlowLayout
 - Los componentes se ponen de izquierda a derecha hasta llenar la línea, y se pasa a la siguiente. Cada línea se centra
 - Por defecto, en paneles y applets
 - BorderLayout
 - Se ponen los componentes en un lateral o en el centro
 - se indica con una dirección: “East”, “West”, “North”, “South”, “Center”
 - Por defecto, en marcos
 - GridLayout
 - Se colocan los componentes en una rejilla rectangular (filas x cols)
 - Se añaden en orden izquierda-derecha y arriba-abajo
- ⊙ Para poner un layout se utiliza el método *setLayout()*:
GridLayout nuevayout = new GridLayout(3,2);
setLayout(nuevayout);

Administrador de diseño



FlowLayout

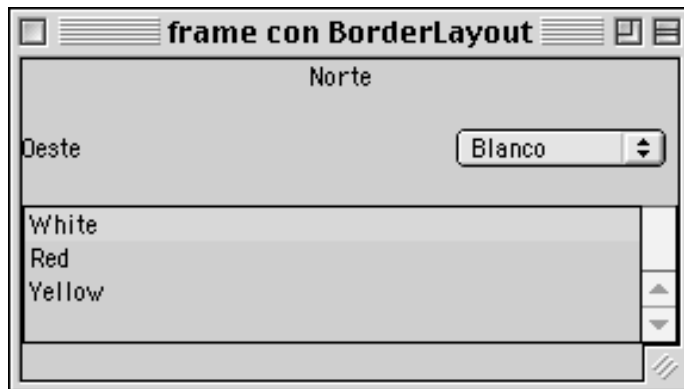


GridLayout(3,2)

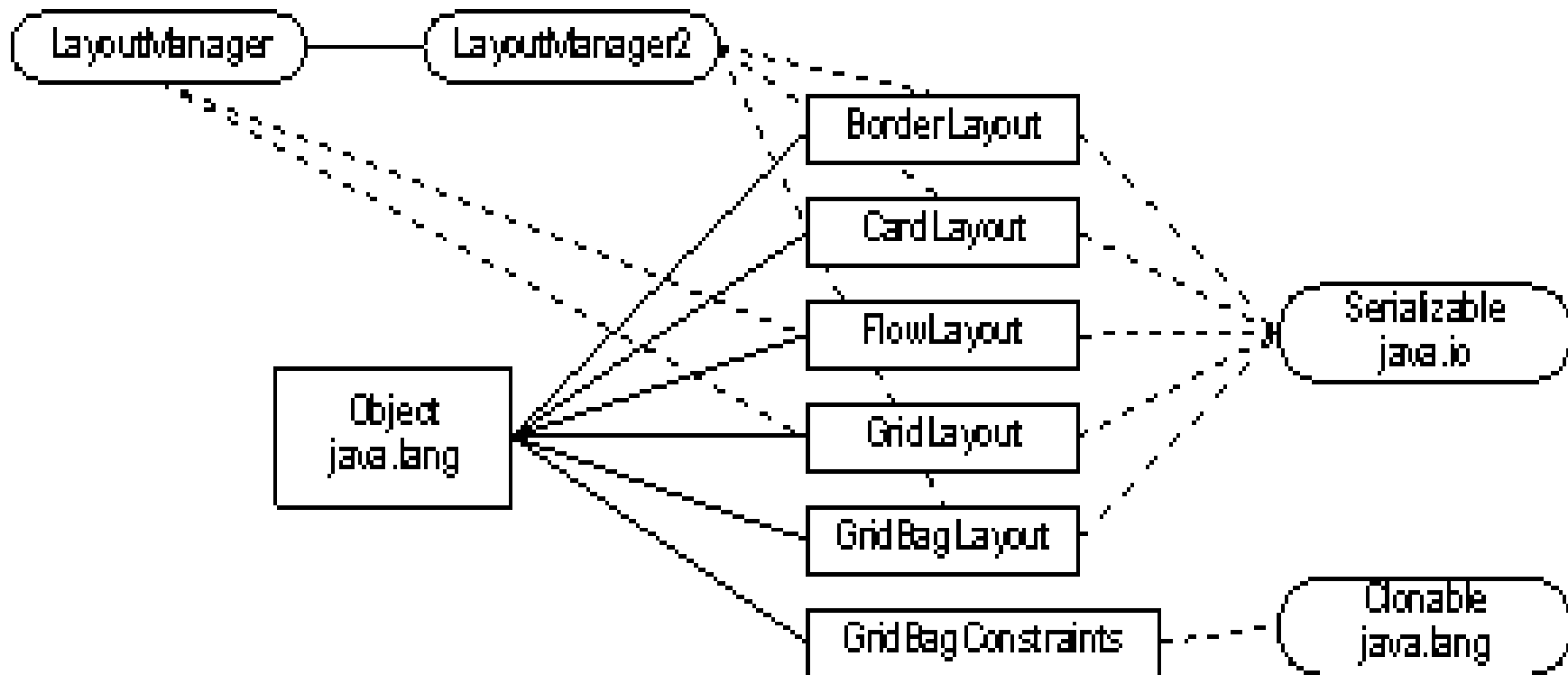
Otros administradores

◉ GridBagLayout

- Similar al GridLayout pero mas versátil
- Presenta los componentes en una rejilla, pero:
 - Un componente puede ocupar más de una fila y más de una columna
 - Las filas y las columnas pueden tener tamaños diferentes
 - No se tiene que rellenar en un orden predeterminado
- Utiliza *GridBagConstraints* para especificar como deben colocarse, distribuirse, alinearse, etc., los componentes



Administradores de diseño



Nuevos administradores de diseño en Swing

◉ *BoxLayout*

- Organiza los componentes en una única fila o columna
 - Por defecto el espacio libre se deja al final
- Los elementos pueden tener distinto tamaño y alineación

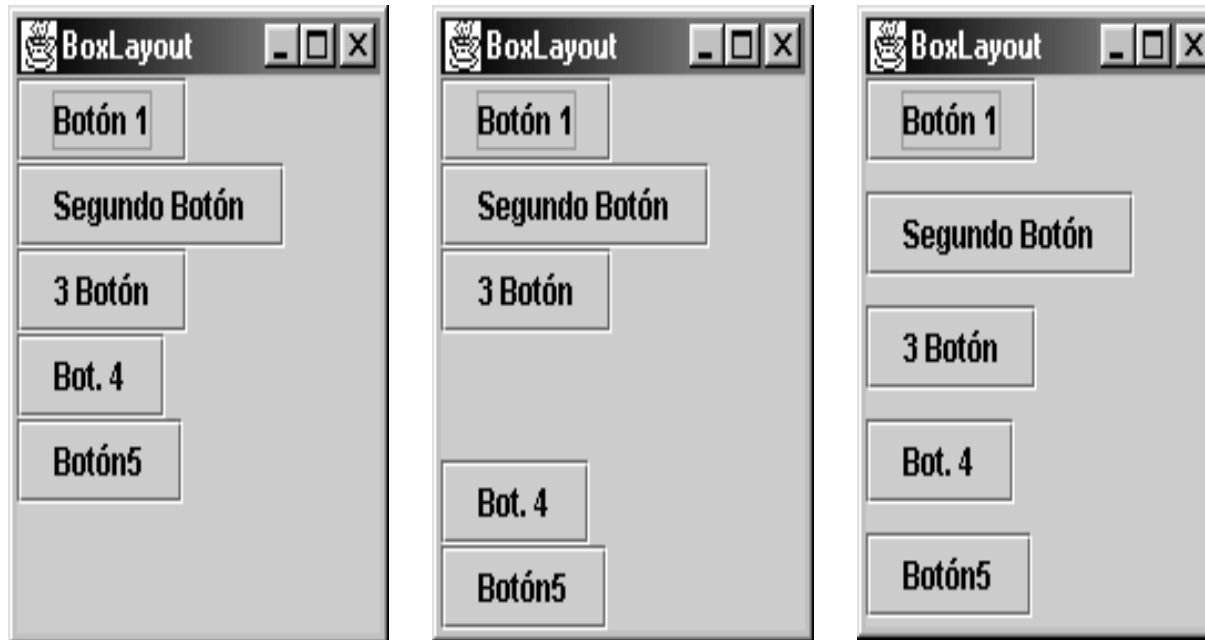
◉ Normalmente se utiliza conjuntamente con la clase *Box*

- Permite crear componentes invisibles que ocupan un tamaño fijo para mejorar la presentación (áreas rígidas y *struts*)
- Permite crear “gomas extensibles” o componentes invisibles que también se redimensionan cuando se redimensiona el contenedor

Ejemplo BorderLayout

```
public class PruebaBoxLayout extends JFrame {
    PruebaBoxLayout() {
        JButton b1, b2, b3, b4, b5;
        b1 = new JButton("Botón 1"); b2 = new JButton("Segundo Botón");
        b3 = new JButton("3 Botón"); b4 = new JButton("Bot. 4");
        b5 = new JButton("Botón5");
        JPanel panel = new JPanel();
        // se asigna un BorderLayout vertical al panel
        panel.setLayout( new BorderLayout(panel, BorderLayout.Y_AXIS));
        // se añaden los botones al panel con glue entre ellos
        panel.add(b1); panel.add(Box.createGlue());
        panel.add(b2); panel.add(Box.createGlue());
        panel.add(b3); panel.add(Box.createGlue());
        panel.add(b4); panel.add(Box.createGlue());
        panel.add(b5);
        getContentPane().add(panel);
        setTitle("BoxLayout");
        pack(); setVisible(true);
    }
    public static void main(String args[]) {
        PruebaBoxLayout ventana = new PruebaBoxLayout();}
}
```

Resultado BorderLayout



La captura de la izquierda es la situación por defecto, en la central se introduce “pegamento” entre los botones tres y cuatro, y la captura de la derecha es con “pegamento” entre todos los botones

Graphics

- ◉ Clase abstracta que es la base para los contextos gráficos que permiten a una aplicación dibujar los componentes independientemente del dispositivo de salida
- ◉ Un contexto gráfico es un objeto que funciona junto con las ventanas para mostrar los objetos gráficos
- ◉ Habitualmente no hay que crear ningún contexto gráfico ya que esto es parte del framework de AWT y de Swing
 - Cada componente tiene un objeto Graphics asociado
 - Se obtiene mediante el método `getGraphics()`
 - Se puede dibujar en en dicho objeto Graphics modificando la apariencia del componente
- ◉ Mediante el método `paint(Graphics contexto)` –AWT- o el método `paintComponent (Graphics contexto)` –Swing- se determina que es lo que se debe mostrar en dicho contexto

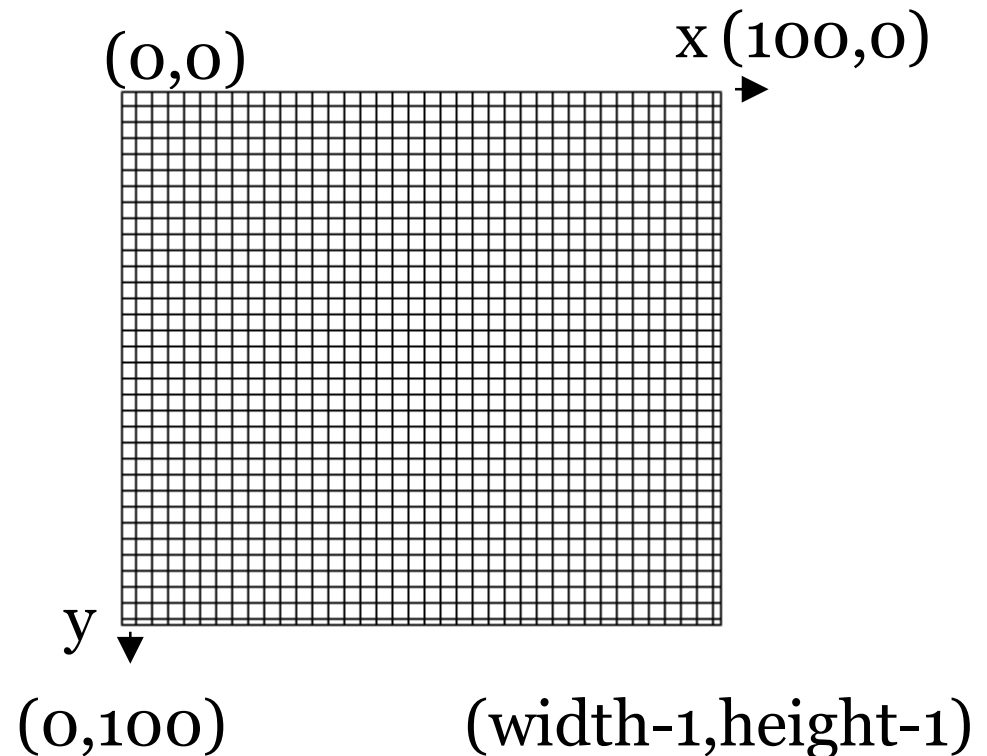
Graphics

⊙ Proporciona métodos para dibujar, rellenar, pintar imágenes, copiar áreas y pegar gráficos en pantalla

- drawLine
- drawRect y fillRect
- drawPolygon
- drawPolyline
- drawOval y fillOval
- drawArc y fillArc

⊙ y para escribir texto

- drawString
- setFont



Ejemplo gráfico con Canvas (AWT)

```
// canvas que se añade a un frame
public class EjemploCanvas extends Canvas {
    String cad = "Escrito en canvas";
    // este metodo se ejecuta automaticamente cuando Java necesita mostrar la ventana
    public void paint(Graphics g) {
        // obtener el color original
        Color colorOriginal = g.getColor();
        // escribir texto grafico en la ventana y recuadrarlo
        g.drawString(cad, 40, 20);
        g.drawRect(35, 8, (cad.length()*7), 14);
        // dibujo de algunas lineas
        for (int i=20; i< 50; i= i+3) {
            if ((i % 2) == 0)        g.setColor(Color.blue);
            else                    g.setColor(Color.red);
            g.drawLine(40, (90-i), 120, 25+i);
        }
        // dibujo y relleno de un óvalo
        g.drawOval(40, 95, 120, 20);
        g.fillOval(40, 95, 120, 20);
        g.setColor(colorOriginal);}}}
```



Críticas, dudas, sugerencias...



Federico Peinado
www.federicopeinado.es