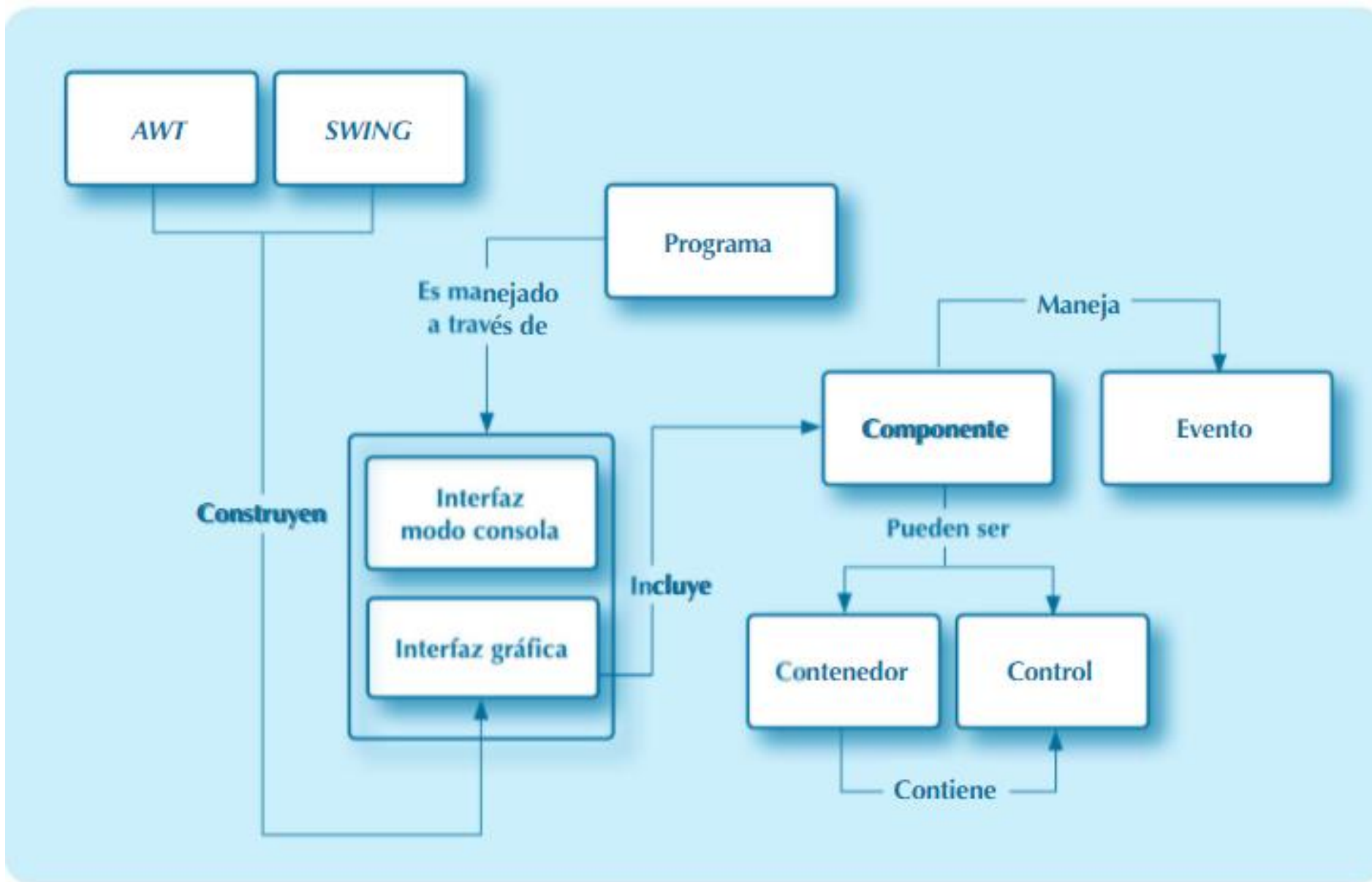
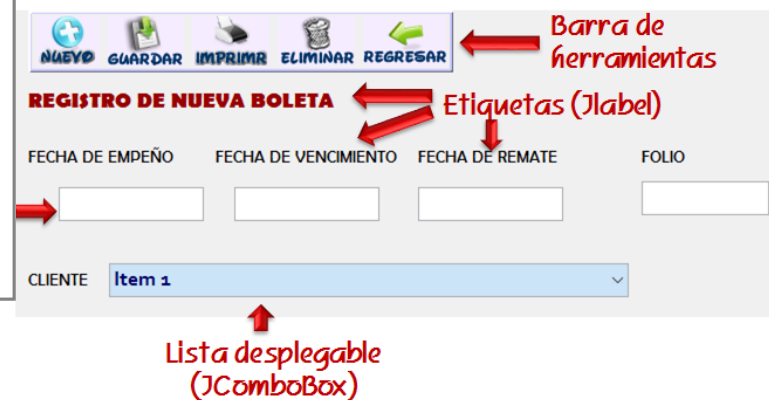
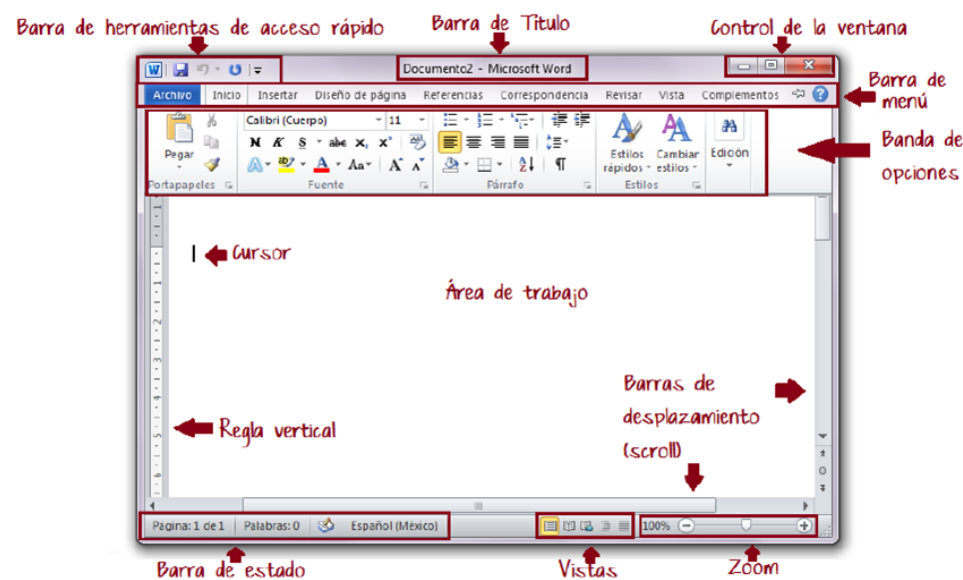


# Interfaz Gráfica de Usuario (GUI)



# Interfaces Gráficas de Usuario

- Una interfaz gráfica de usuario (GUI) es el conjunto de elementos gráficos que permiten establecer una relación entre una aplicación y el usuario de dicha aplicación, por ejemplo: ventanas, botones, menús, cajas de diálogo, etc. Las interfaces gráficas de usuario deben ser fáciles de usar para que la interacción sea lo más instintiva e intuitiva posible.



# Interfaces Gráficas de Usuario en Java

- ▶ El lenguaje Java dispone de una gran cantidad de clases para la creación de interfaces gráficas de usuario:
  - ▶ Java AWT es una API para el desarrollo de aplicaciones con interfaz gráfica cuyos componentes se encuentra en el paquete *java.awt*. Su característica principal es que utiliza clases que crean componentes nativos del SO, por lo que su aspecto depende del SO sobre el que se ejecute.
  - ▶ Java Swing se basa en AWT (de hecho muchas de sus clases heredan de clases de AWT), pero añade componentes más completos y sofisticados que esta, ubicados en el paquete *javax.swing*. A diferencia de AWT, los componentes están escritos directamente en Java sin utilizar componentes del SO, por lo que su aspecto es independiente del SO sobre el que se ejecute.
  - ▶ Java FX es la tecnología de Oracle empleada en el desarrollo de aplicaciones gráficas de cliente enriquecidas que puedan correr en varias plataformas, comportándose de la misma manera en todas ellas.

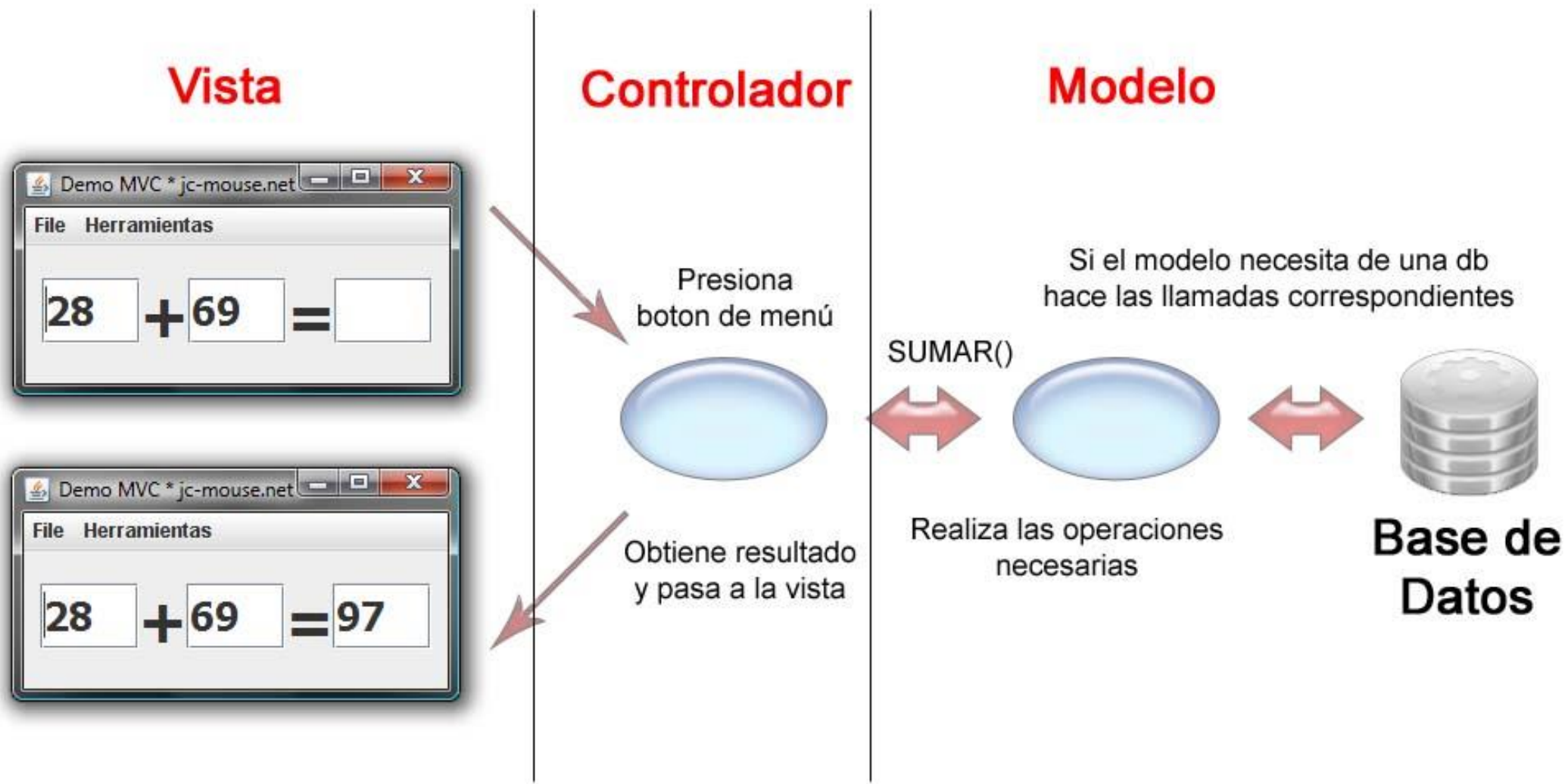


# Modelo Vista Controlador (MVC)



- ▶ Java Swing permite implementar el modelo vista controlador, que es un patrón de arquitectura de software que separa la lógica de negocio de la aplicación de la interfaz de usuario.
- ▶ El Modelo Vista Controlador (MVC) separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos:
  - ▶ El modelo contiene una representación de los datos que maneja el sistema, su lógica de negocio y sus mecanismos de persistencia.
  - ▶ La vista, o interfaz de usuario, compone la información que se envía al cliente y los mecanismos de interacción con este.
  - ▶ El controlador actúa como intermediario entre el modelo y la vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

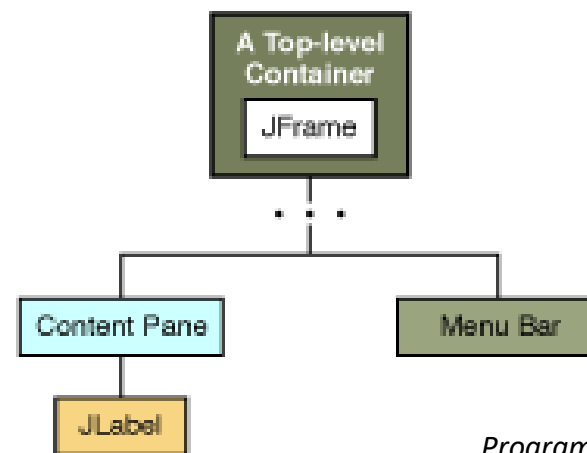
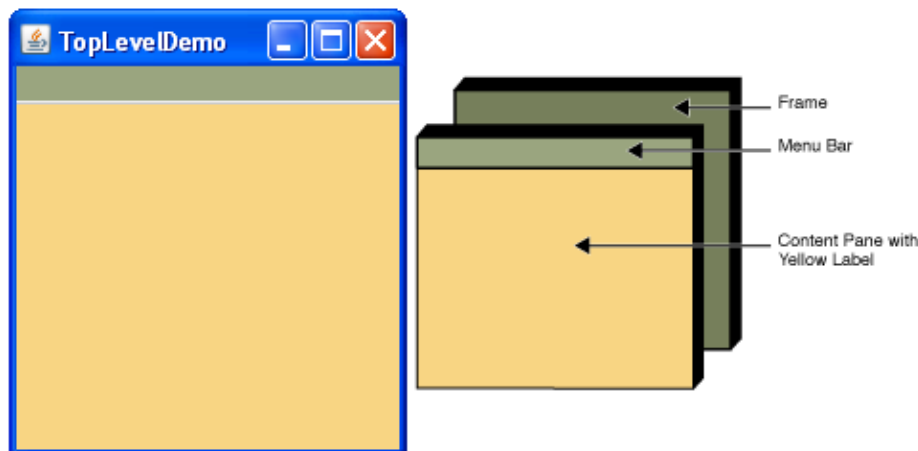
# Ejemplo Modelo Vista Controlador (MVC)



# Contenedores y componentes



- ▶ Para la construcción de interfaces gráficas en Java utilizaremos elementos gráficos que se dividen en dos grandes grupos:
  - ▶ Componentes: cualquier elemento que tiene una representación gráfica en pantalla y el usuario puede interactuar con él. Heredan de la clase base **JComponent**. También se les llama controles.
  - ▶ Contenedores: es un tipo especial de componente que puede incluir varios componentes u otros contenedores. Los contenedores de alto nivel más utilizados en Java son **JFrame** (proporciona una ventana o marco principal de la aplicación) y **JDialog** (genera ventanas secundarias, como los cuadros de diálogo). Entre los contenedores intermedios está **JPanel**, que no genera una ventana independiente y cuyo cometido es agrupar componentes.





# Contenedores de alto nivel



- ▶ Para aparecer en pantalla, cualquier componente GUI debe ser parte de una jerarquía en forma de árbol de componentes con un contenedor de alto nivel como raíz.
- ▶ Cada componente GUI que crees solo puede estar en un contenedor, por lo que si tienes un componente en un contenedor y lo añades a otro se eliminará del primero.
- ▶ Cada contenedor de alto nivel (como JFrame) tiene un panel de contenidos (***Content pane***) que contiene los componentes visibles de la GUI.
- ▶ Opcionalmente, puedes añadir una barra de menú al contenedor de alto nivel, esta barra de menús estará fuera del panel de contenidos y típicamente se mostrará en la parte superior de la ventana.
- ▶ Por lo general, una aplicación Java con GUI basada en Swing tiene un contenedor de alto nivel que es JFrame.

# JFrame



- ▶ Para obtener una ventana en nuestra aplicación se crea un objeto de la clase `JFrame` (`javax.swing`) que proporciona los atributos y comportamientos básicos de una ventana: una barra de título con los botones minimizar, maximizar y cerrar.
- ▶ Todos los objetos de la clase `JFrame` tienen un panel raíz de la clase `JRootPane` que gestiona el interior de la ventana. En él, está el panel de contenido (`Content Pane`), que existe siempre y la barra de menú que es opcional. Los componentes gráficos de la ventana se añaden a este panel de contenido.
  - ▶ *`JFrame` / `JFrame(String)`*: crea una ventana (inicialmente oculta) con el título correspondiente.
  - ▶ *`getContentPane()` / `setContentPane(Container)`*: obtiene o establece el panel de contenidos de la ventana, que contiene todos los componentes gráficos de la ventana.
  - ▶ *`getMenuBar()` / `setJMenuBar(JMenuBar)`*: obtiene o establece la barra de menús de la ventana.
  - ▶ *`setDefaultCloseOperation(int)`*: establece el comportamiento de la ventana cuando el usuario trata de cerrarla.



# Ejemplo Contenedores de alto nivel

```
// Create and set up the window.
JFrame frame = new JFrame("TopLevelDemo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Create the menu bar. Make it have a green background.
JMenuBar greenMenuBar = new JMenuBar();
greenMenuBar.setOpaque(true);
greenMenuBar.setBackground(new Color(154, 165, 127));
greenMenuBar.setPreferredSize(new Dimension(200, 20));

// Create a yellow label to put in the content pane.
JLabel yellowLabel = new JLabel();
yellowLabel.setOpaque(true);
yellowLabel.setBackground(new Color(248, 213, 131));
yellowLabel.setPreferredSize(new Dimension(200, 180));

// Create OK button
JButton btnOK = new JButton("OK");
btnOK.setBackground(new Color(254, 8, 2));

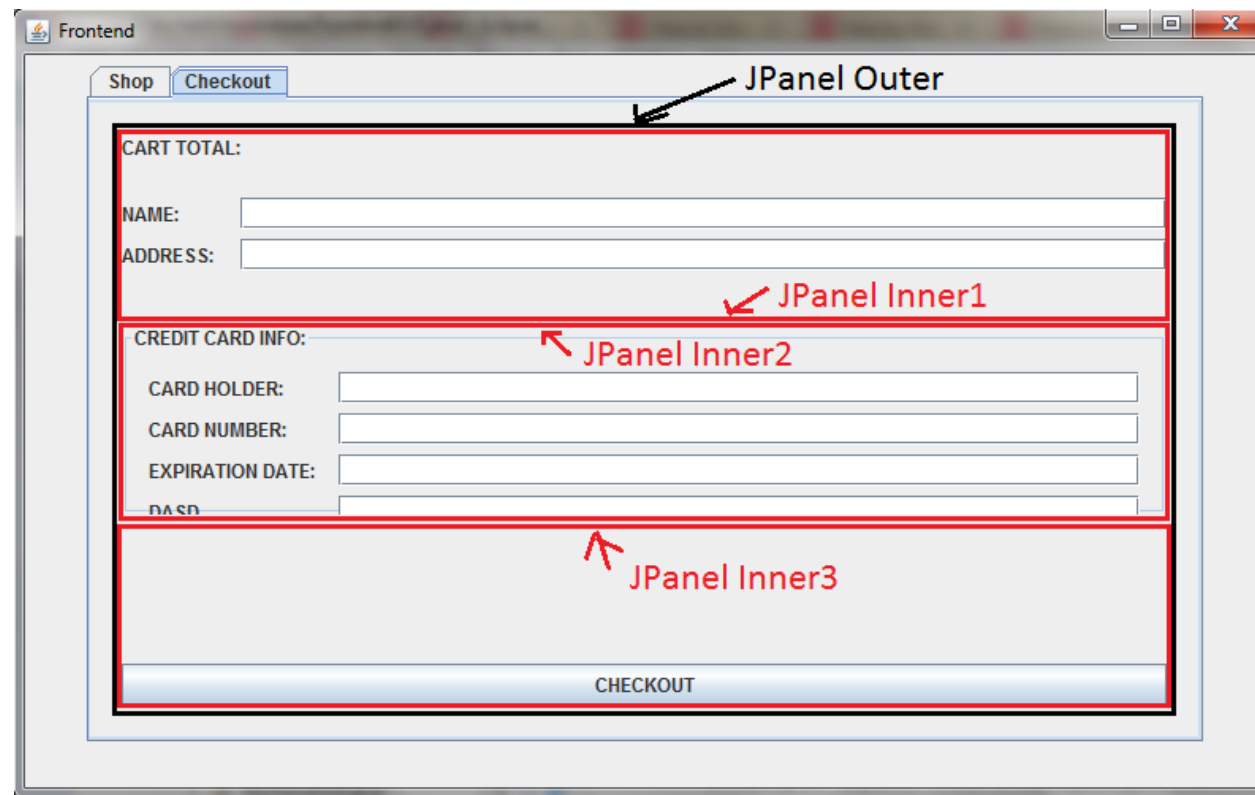
// Set the menu bar and add the label and button to the content pane.
frame.setJMenuBar(greenMenuBar);
frame.getContentPane().add(yellowLabel);
frame.getContentPane().add(btnOK, BorderLayout.PAGE_END);

// Display the window.
frame.pack();
frame.setVisible(true);
```



# Contenedores intermedios

- Se pueden añadir componentes directamente al panel de contenidos del contenedor de alto nivel (JFrame), pero típicamente se utilizan contenedores intermedios como JPanel, que hereda de JComponent y así nos permite utilizar toda la funcionalidad de esta clase.



# Ejemplo Contenedores intermedios

```
// Create and set up the window.
JFrame frame = new JFrame("JPanelDemo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Create the menu bar. Make it have a green background.
JMenuBar greenMenuBar = new JMenuBar();
greenMenuBar.setOpaque(true);
greenMenuBar.setBackground(new Color(154, 165, 127));
greenMenuBar.setPreferredSize(new Dimension(200, 20));

// Create a yellow label to put in the content pane.
JLabel yellowLabel = new JLabel();
yellowLabel.setOpaque(true);
yellowLabel.setBackground(new Color(248, 213, 131));
yellowLabel.setPreferredSize(new Dimension(200, 180));

// Create OK button
JButton btnOK = new JButton();
ImageIcon imagenBoton = new ImageIcon("img/boton_rojo.png");
btnOK.setIcon(imagenBoton);
btnOK.setBackground(new Color(248, 10, 0));
btnOK.setFocusable(false);

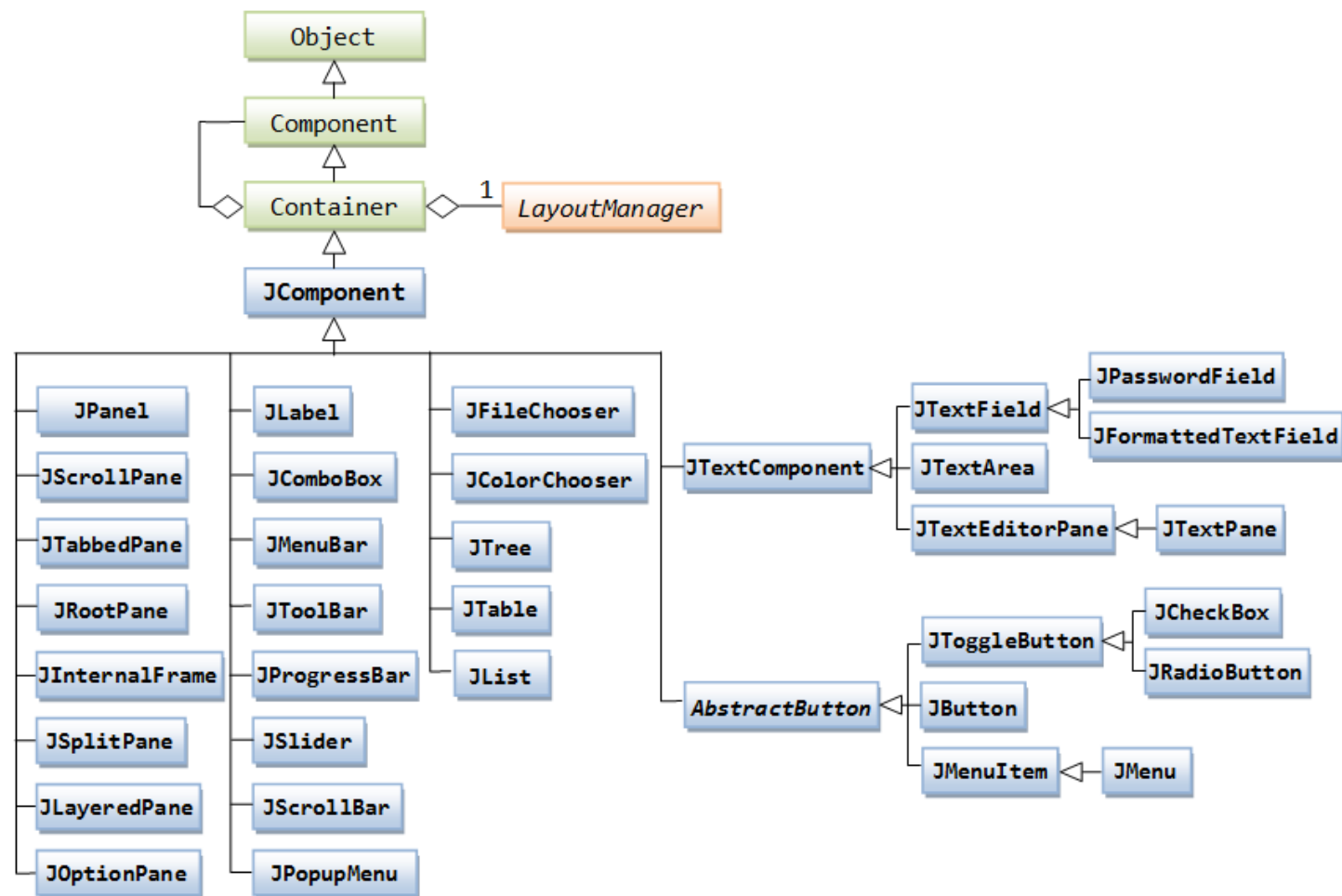
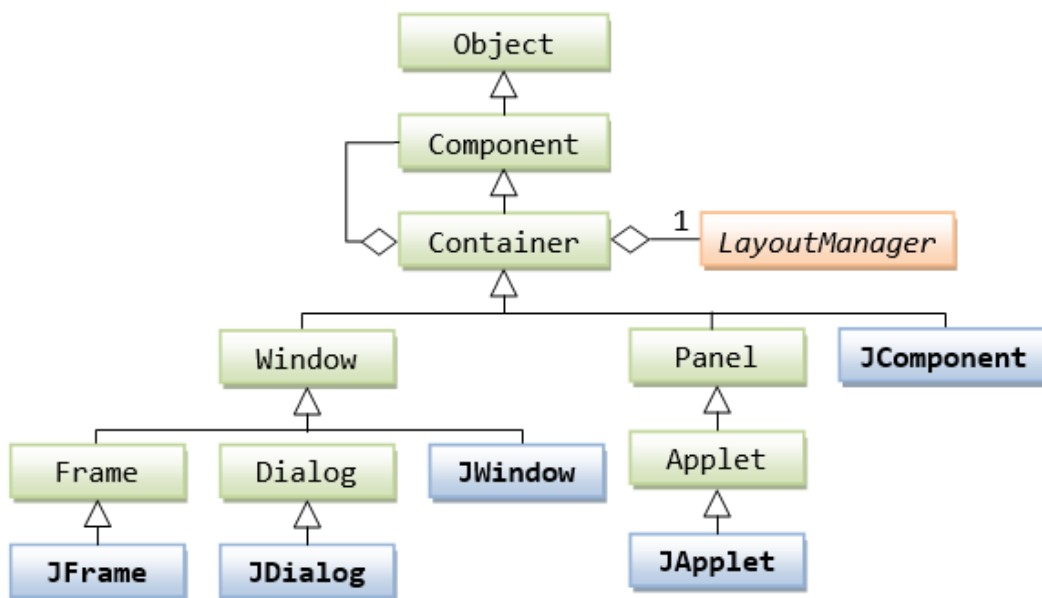
// Create a panel and add components to it.
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.add(yellowLabel, BorderLayout.CENTER);
contentPane.add(btnOK, BorderLayout.PAGE_END);

// Set the menu bar and add the label and button to the content pane.
frame.setJMenuBar(greenMenuBar);
frame.setContentPane(contentPane);

// Display the window.
frame.pack();
frame.setVisible(true);
```



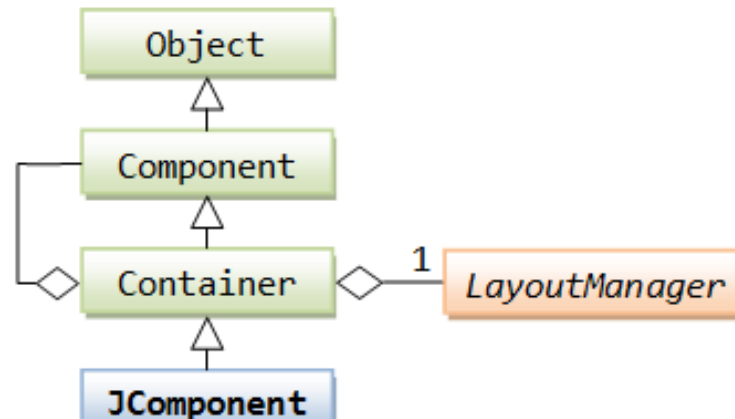
# Classes Java Swing



# Clases y métodos básicos



- ▶ Las clases básicas son aquellas que aportan soporte y funcionalidad básica al resto de componentes gráficos. Los principales métodos que utilizaremos en la construcción de nuestra interfaz gráfica de usuario con Java Swing pertenecen a las siguientes clases:
  - ▶ `java.awt.Component`
  - ▶ `java.awt.Container`
  - ▶ `javax.swing.JComponent`
- ▶ La clase `JComponent` hereda de las dos anteriores por lo que cuenta con todos los métodos de estas más varios añadidos.



# Métodos de contenedores



Method	Purpose
Component add(Component) Component add(Component, int) void add(Component, Object)	Add the specified component to this container. The one-argument version of this method adds the component to the end of the container. When present, the <code>int</code> argument indicates the new component's position within the container. When present, the <code>Object</code> argument provides layout constraints to the current layout manager.
void remove(int) void remove(Component) void removeAll()	Remove one of or all of the components from this container. When present, the <code>int</code> argument indicates the position within the container of the component to remove.
JRootPane getRootPane()	Get the root pane that contains the component.
Container getTopLevelAncestor()	Get the topmost container for the component — a <code>Window</code> , <code>Applet</code> , or null if the component has not been added to any container.
Container getParent()	Get the component's immediate container.
int getComponentCount()	Get the number of components in this container.
Component getComponent(int) Component[] getComponents()	Get the one of or all of the components in this container. The <code>int</code> argument indicates the position of the component to get.
Component getComponentZOrder(int) Component[] getComponentZOrder()	Returns the z-order index of the component inside the container. The higher a component is in the z-order hierarchy, the lower its index. The component with the lowest z-order index is painted last, above all other child components.



# Métodos de disposición de componentes



Method	Purpose
<code>void setPreferredSize(Dimension)</code> <code>void setMaximumSize(Dimension)</code> <code>void setMinimumSize(Dimension)</code>	Set the component's preferred, maximum, or minimum size, measured in pixels. The preferred size indicates the best size for the component. The component should be no larger than its maximum size and no smaller than its minimum size. Be aware that these are hints only and might be ignored by certain layout managers.
<code>Dimension getPreferredSize()</code> <code>Dimension getMaximumSize()</code> <code>Dimension getMinimumSize()</code>	Get the preferred, maximum, or minimum size of the component, measured in pixels. Many <code>JComponent</code> classes have setter and getter methods. For those non- <code>JComponent</code> subclasses, which do not have the corresponding setter methods, you can set a component's preferred, maximum, or minimum size by creating a subclass and overriding these methods.
<code>void setAlignmentX(float)</code> <code>void setAlignmentY(float)</code>	Set the alignment along the <i>x</i> - or <i>y</i> - axis. These values indicate how the component would like to be aligned relative to other components. The value should be a number between 0 and 1 where 0 represents alignment along the origin, 1 is aligned the furthest away from the origin, and 0.5 is centered, and so on. Be aware that these are hints only and might be ignored by certain layout managers.
<code>float getAlignmentX()</code> <code>float getAlignmentY()</code>	Get the alignment of the component along the <i>x</i> - or <i>y</i> - axis. For non- <code>JComponent</code> subclasses, which do not have the corresponding setter methods, you can set a component's alignment by creating a subclass and overriding these methods.
<code>void setLayout(LayoutManager)</code> <code>LayoutManager getLayout()</code>	Set or get the component's layout manager. The layout manager is responsible for sizing and positioning the components within a container.
<code>void</code> <code>applyComponentOrientation(ComponentOrientation)</code> <code>void</code> <code>setComponentOrientation(ComponentOrientation)</code>	Set the <code>ComponentOrientation</code> property of this container and all the components contained within it. See <a href="#">Setting the Container's Orientation</a> for more information.



# Métodos de tamaño y posición



Method	Purpose
<code>int getWidth()</code> <code>int getHeight()</code>	Get the current width or height of the component measured in pixels.
<code>Dimension getSize()</code> <code>Dimension</code> <code>getSize(Dimension)</code>	Get the component's current size measured in pixels. When using the one-argument version of this method, the caller is responsible for creating the <code>Dimension</code> instance in which the result is returned.
<code>int getX()</code> <code>int getY()</code>	Get the current x or y coordinate of the component's origin relative to the parent's upper left corner measured in pixels.
<code>Rectangle getBounds()</code> <code>Rectangle</code> <code>getBounds(Rectangle)</code>	Get the bounds of the component measured in pixels. The bounds specify the component's width, height, and origin relative to its parent. When using the one-argument version of this method, the caller is responsible for creating the <code>Rectangle</code> instance in which the result is returned.
<code>Point getLocation()</code> <code>Point</code> <code>getLocation(Point)</code>	Gets the current location of the component relative to the parent's upper left corner measured in pixels. When using the one-argument version of <code>getLocation</code> method, the caller is responsible for creating the <code>Point</code> instance in which the result is returned.
<code>Point</code> <code>getLocationOnScreen()</code>	Returns the position relative to the upper left corner of the screen.
<code>Insets getInsets()</code>	Get the size of the component's border.

Method	Purpose
<code>void setLocation(int, int)</code> <code>void</code> <code>setLocation(Point)</code>	Set the location of the component, in pixels, relative to the parent's upper left corner. The two <code>int</code> arguments specify x and y, in that order. Use these methods to position a component when you are not using a layout manager.
<code>void setSize(int, int)</code> <code>void</code> <code>setSize(Dimension)</code>	Set the size of the component measured in pixels. The two <code>int</code> arguments specify width and height, in that order. Use these methods to size a component when you are not using a layout manager.
<code>void setBounds(int, int, int, int)</code> <code>void</code> <code>setBounds(Rectangle)</code>	Set the size and location relative to the parent's upper left corner, in pixels, of the component. The four <code>int</code> arguments specify x, y, width, and height, in that order. Use these methods to position and size a component when you are not using a layout manager.

# Métodos de estado del componente



Method	Purpose
<code>void setComponentPopupMenu(JPopupMenu)</code>	<p>Sets the <code>JPopupMenu</code> for this <code>JComponent</code>. The UI is responsible for registering bindings and adding the necessary listeners such that the <code>JPopupMenu</code> will be shown at the appropriate time. When the <code>JPopupMenu</code> is shown depends upon the look and feel: some may show it on a mouse event, some may enable a key binding.</p> <p>If <code>popup</code> is null, and <code>getInheritsPopupMenu</code> returns <code>true</code>, then <code>getComponentPopupMenu</code> will be delegated to the parent. This provides for a way to make all child components inherit the <code>popupmenu</code> of the parent.</p>
<code>void setTransferHandler(TransferHandler) TransferHandler getTransferHandler()</code>	Set or remove the <code>transferHandler</code> property. The <code>TransferHandler</code> supports exchanging data via cut, copy, or paste to/from a clipboard as well as a drag and drop. See <a href="#">Introduction to DnD</a> for more details.
<code>void setToolTipText(String)</code>	Set the text to display in a tool tip. See <a href="#">How to Use Tool Tips</a> for more information.
<code>void setName(String) String getName()</code>	Set or get the name of the component. This can be useful when you need to associate text with a component that does not display text.
<code>boolean isShowing()</code>	Determine whether the component is showing on screen. This means that the component must be visible, and it must be in a container that is visible and showing.
<code>void setEnabled(boolean) boolean isEnabled()</code>	Set or get whether the component is enabled. An enabled component can respond to user input and generate events.
<code>void setVisible(boolean) boolean isVisible()</code>	Set or get whether the component is visible. Components are initially visible, with the exception of top-level components.

# Métodos de apariencia del componente



Method	Purpose
<code>void setBorder(Border) Border getBorder()</code>	Set or get the border of the component. See <a href="#">How to Use Borders</a> for details.
<code>void setForeground(Color) void setBackground(Color)</code>	Set the foreground or background color for the component. The foreground is generally the color used to draw the text in a component. The background is (not surprisingly) the color of the background areas of the component, assuming that the component is opaque.
<code>Color getForeground() Color getBackground()</code>	Get the foreground or background color for the component.
<code>void setOpaque(boolean) boolean isOpaque()</code>	Set or get whether the component is opaque. An opaque component fills its background with its background color.
<code>void setFont(Font) Font getFont()</code>	Set or get the component's font. If a font has not been set for the component, the font of its parent is returned.
<code>void setCursor(Cursor) Cursor getCursor()</code>	Set or get the cursor displayed over the component and all components it contains (except for children that have their own cursor set). Example: <code>aPanel.setCursor ( Cursor.getPredefinedCursor ( Cursor.WAIT_CURSOR) ) ;</code>

# LAYOUT MANAGERS



Sorting Techniques

Winning Algorithm: Insertion Sort

List Properties

☐ InOrder ☐ ReverseOrder  
☐ AlmostOrder ☒ Random

15286

Create The List

Experimental Results

N: 15286  
DataType: Random  
Sort: Insertion  
Comparisons: 116823255  
Movements: 45855  
Total time: 230

Insertion Sort  
Selection Sort  
Quick Sort  
Merge Sort  
Heap Sort  
Radix Sort

AbsoluteLayoutDemo

one two three

FlowLayout

Button One Button Two  
Button Three

Border Layout

North  
East

GridLayout

Botón 1 Botón 2  
Botón 3 Botón 4 con nombre largo  
5

Ataque: 0 Defensa: 0

Calcular

Button One  
Button Two  
Button Three

7	8	9
4	5	6
1	2	3
*	0	#

GridBag

One Two Three

BorderLayout

Botón 1 (PAGE\_START)

Botón 3 (LINE\_START) Botón 2 (CENTER) 5 (LINE\_END)

Botón 4 con nombre largo (PAGE\_END)

FlowLayout

Botón 1 Botón 2 Botón 3  
Botón 4 con nombre largo 5

Howdy Doody

# Layout Managers



- ▶ Se llaman Gestores de diseño (Layout Managers) a las clases que permiten ordenar los componentes de la GUI en un contenedor con el objetivo de obtener una buena presentación. Estas clases también proporcionan herramientas (métodos) de distribución visual que son más fáciles de manejar que si tuviéramos que determinar la posición y tamaño exactos de cada componente.
- ▶ Todos los contenedores pueden tener asociado un LayoutManager, que establece cómo serán distribuidos los componentes dentro de él. Si no se ha establecido, y por tanto su valor es null, el contenedor no usará ningún administrador de diseño por lo que se deberá especificar el tamaño y la posición de cada componente dentro del contenedor. La desventaja de utilizar posicionamiento absoluto es que si se cambia el tamaño del contenedor de nivel superior, los componentes contenidos no se ajustarán correctamente.

<code>LayoutManager getLayout();</code>	Devuelve el gestor de impresión ( <i>LayoutManager</i> ) asociado a este contenedor, que es el responsable de colocar los componentes dentro del contenedor.
<code>void setLayout(LayoutManager l);</code>	Establece un gestor de impresión para este componente.

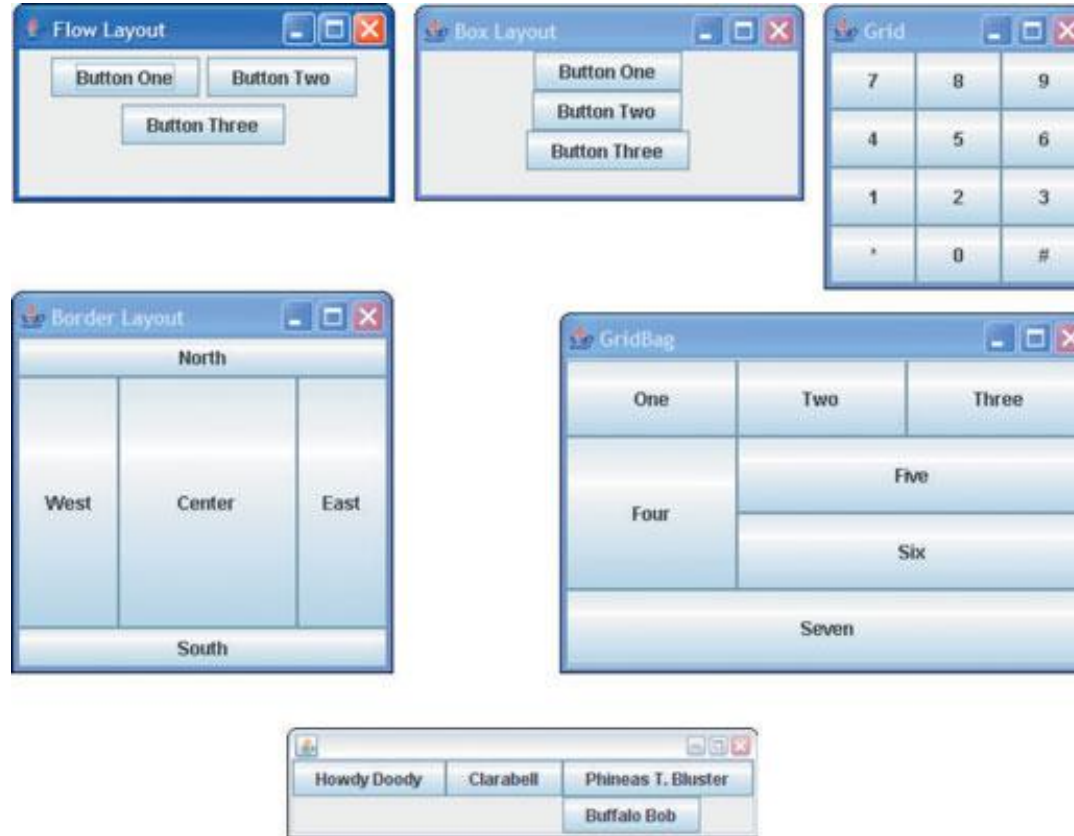


# Tipos de layouts



- ▶ Los principales tipos de layout managers para uso en nuestra aplicación son:

- ▶ BorderLayout
- ▶ BoxLayout
- ▶ CardLayout
- ▶ FlowLayout
- ▶ GridBagLayout
- ▶ GridLayout
- ▶ GroupLayout
- ▶ SpringLayout



- ▶ Por defecto, el layout manager de un content pane es BorderLayout, mientras que un JPanel tiene FlowLayout como layout manager por defecto.

# Border Layout



- BorderLayout sitúa los elementos en cinco áreas o regiones: arriba, abajo, izquierda, derecha y centro. Lo que se sitúe en la región central se expande para ocupar el espacio de las regiones desocupadas. Cada región admite un único componente, pero nada impide que este componente sea a su vez un contenedor que admita más componentes.

```
JFrame frame = new JFrame("BorderLayout");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

Container contenido = frame.getContentPane();

JButton button = new JButton("Botón 1 (PAGE_START)");
contenido.add(button, BorderLayout.PAGE_START);

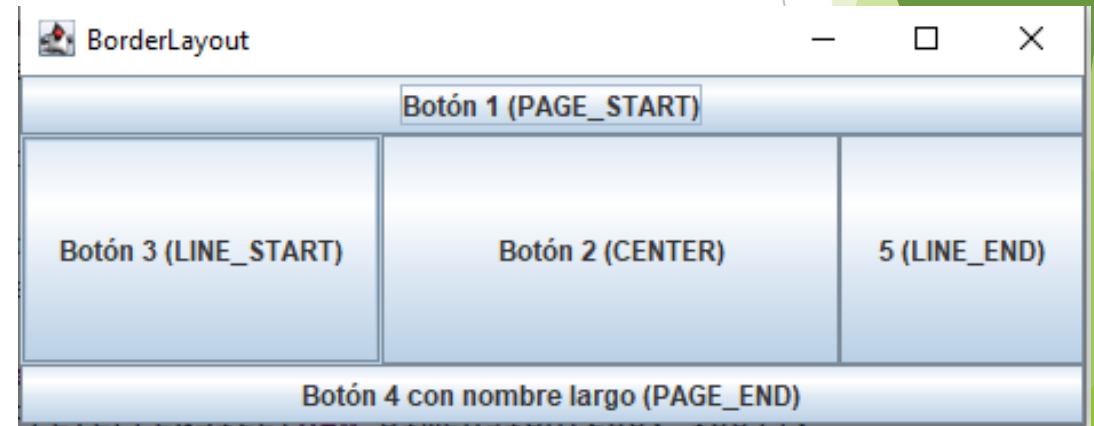
button = new JButton("Botón 2 (CENTER)");
button.setPreferredSize(new Dimension(200, 100));
contenido.add(button, BorderLayout.CENTER);

button = new JButton("Botón 3 (LINE_START)");
contenido.add(button, BorderLayout.LINE_START);

button = new JButton("Botón 4 con nombre largo (PAGE_END)");
contenido.add(button, BorderLayout.PAGE_END);

button = new JButton("5 (LINE_END)");
contenido.add(button, BorderLayout.LINE_END);

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





# Flow Layout



- FlowLayout sitúa los componentes horizontalmente de izquierda a derecha en cada línea, si se sobrepasa una línea comienza a la izquierda de la siguiente. Puede instanciarse con posicionamiento centrado, izq, dcho y también estableciendo espacio horizontal y vertical.

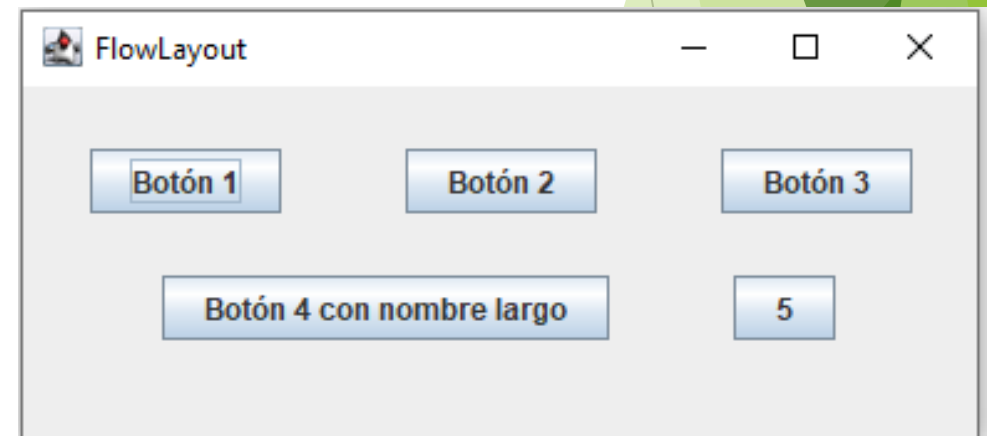
```
JFrame frame = new JFrame("FlowLayout");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

Container contenido = frame.getContentPane();

contenido.setLayout(new FlowLayout(FlowLayout.CENTER, 50, 25));

contenido.add(new JButton("Botón 1"));
contenido.add(new JButton("Botón 2"));
contenido.add(new JButton("Botón 3"));
contenido.add(new JButton("Botón 4 con nombre largo"));
contenido.add(new JButton("5"));

frame.setPreferredSize(new Dimension(400, 180));
frame.pack();
frame.setVisible(true);
```



# Box Layout

- BoxLayout sitúa los componentes en una sola fila o columna, respetando los tamaños máximos asignados a los componentes y permitiendo alinearlos.

```
JFrame frame = new JFrame("BoxLayout");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JButton jb1 = new JButton("Botón 1 -");
JButton jb2 = new JButton("Botón 2 -----");
JButton jb3 = new JButton("Botón 3 -");
JButton jb4 = new JButton("Botón 4 -----");
JButton jb5 = new JButton("Botón 5 -");
JButton jb6 = new JButton("Botón 6 -----");

JPanel panel1 = new JPanel();
JPanel panel2 = new JPanel();
JPanel panel3 = new JPanel();

panel1.setBorder(BorderFactory.createTitledBorder("LEFT"));
panel2.setBorder(BorderFactory.createTitledBorder("CENTER"));
panel3.setBorder(BorderFactory.createTitledBorder("RIGHT"));

// Set up the BoxLayout
BoxLayout layout1 = new BoxLayout(panel1, BoxLayout.Y_AXIS);
BoxLayout layout2 = new BoxLayout(panel2, BoxLayout.Y_AXIS);
BoxLayout layout3 = new BoxLayout(panel3, BoxLayout.Y_AXIS);
panel1.setLayout(layout1);
panel2.setLayout(layout2);
panel3.setLayout(layout3);
```

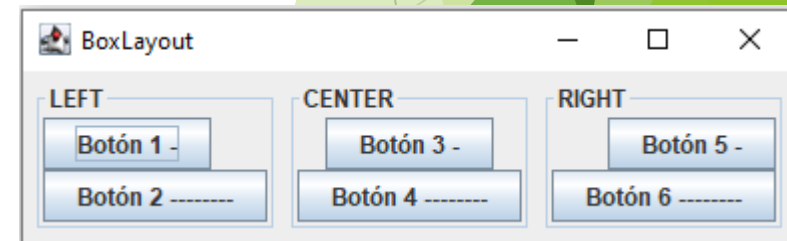
```
jb1.setAlignmentX(Component.LEFT_ALIGNMENT);
jb2.setAlignmentX(Component.LEFT_ALIGNMENT);
panel1.add(jb1);
panel1.add(jb2);

jb3.setAlignmentX(Component.CENTER_ALIGNMENT);
jb4.setAlignmentX(Component.CENTER_ALIGNMENT);
panel2.add(jb3);
panel2.add(jb4);

jb5.setAlignmentX(Component.RIGHT_ALIGNMENT);
jb6.setAlignmentX(Component.RIGHT_ALIGNMENT);
panel3.add(jb5);
panel3.add(jb6);

frame.add(panel1);
frame.add(panel2);
frame.add(panel3);

frame.setLayout(new FlowLayout());
frame.pack();
frame.setVisible(true);
```



# Grid Layout



- GridLayout sitúa los componentes en una rejilla de un determinado número de filas y columnas, con todas las casillas del mismo tamaño.

```
JFrame frame = new JFrame("GridLayout");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

Container contenido = frame.getContentPane();

final JPanel panel = new JPanel();
panel.setLayout(new GridLayout(0, 2));
JPanel controles = new JPanel();
controles.setLayout(new GridLayout(2, 3));

panel.setPreferredSize(new Dimension(365, 130));

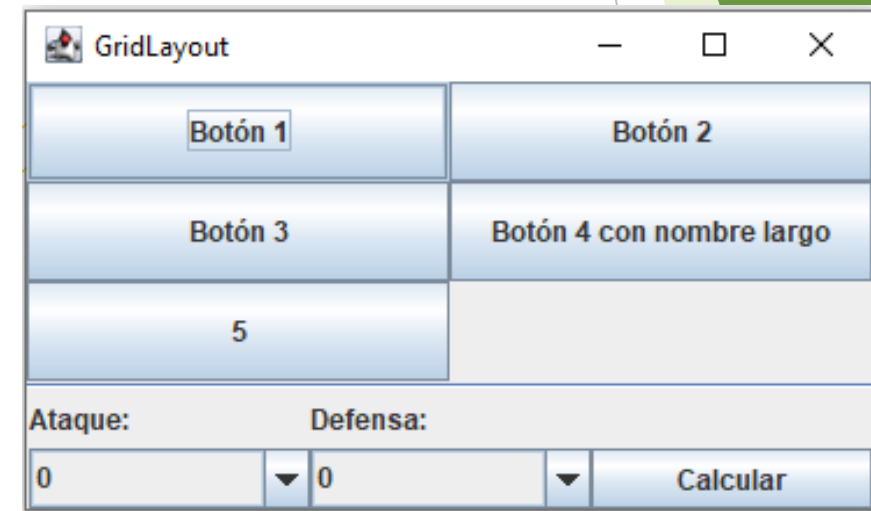
panel.add(new JButton("Botón 1"));
panel.add(new JButton("Botón 2"));
panel.add(new JButton("Botón 3"));
panel.add(new JButton("Botón 4 con nombre largo"));
panel.add(new JButton("5"));

JComboBox comboBox1 = new JComboBox(new String[] {"0", "10", "15", "20"});
JComboBox comboBox2 = new JComboBox(new String[] {"0", "10", "15", "20"});
JButton applyButton = new JButton("Calcular");

controles.add(new JLabel("Ataque:"));
controles.add(new JLabel("Defensa:"));
controles.add(new JLabel(" "));
controles.add(comboBox1);
controles.add(comboBox2);
controles.add(applyButton);

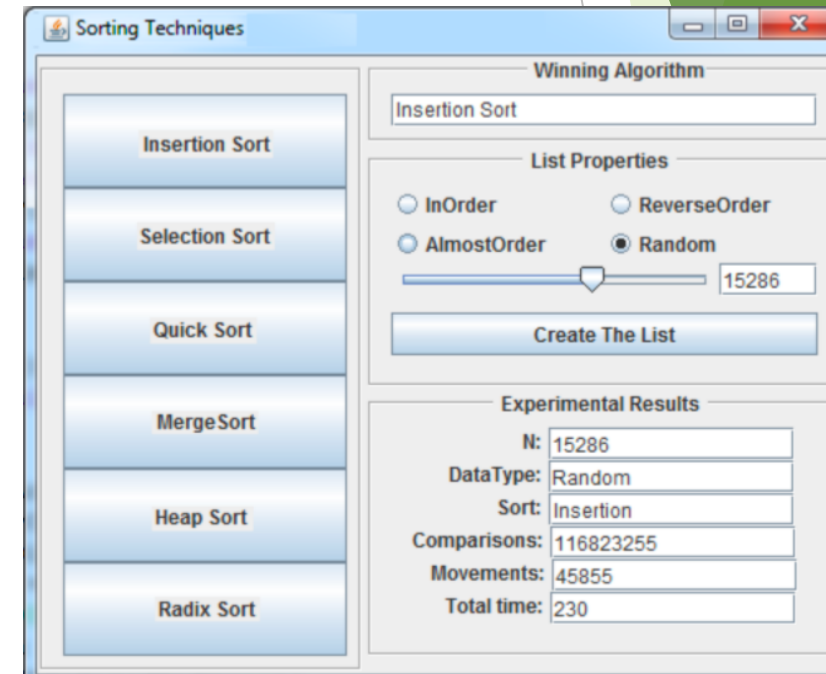
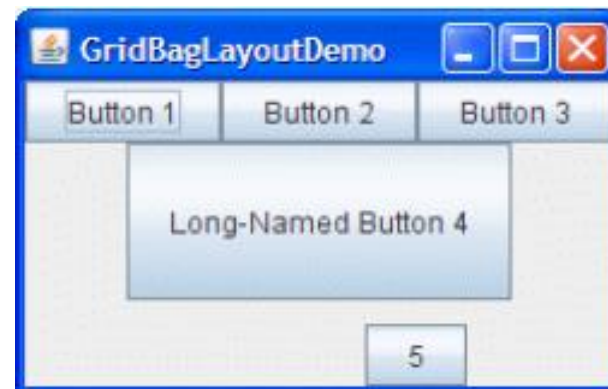
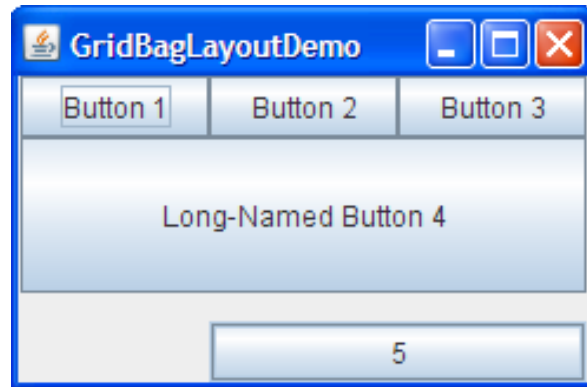
contenido.add(panel, BorderLayout.PAGE_START);
contenido.add(new JSeparator(), BorderLayout.CENTER);
contenido.add(controles, BorderLayout.PAGE_END);

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



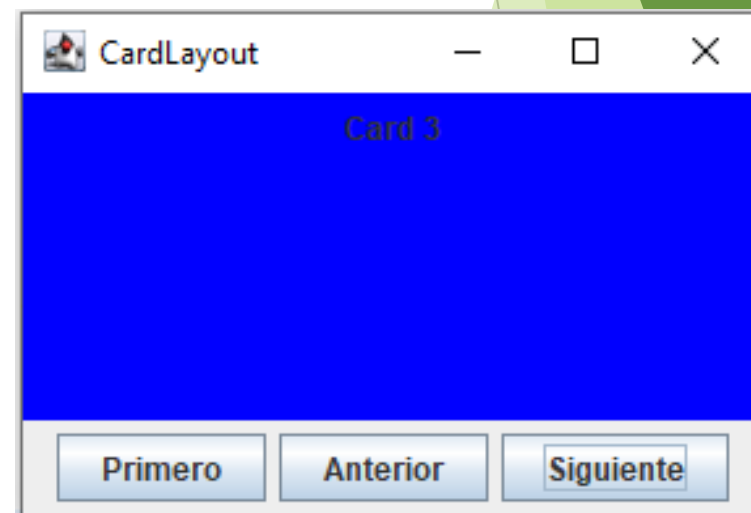
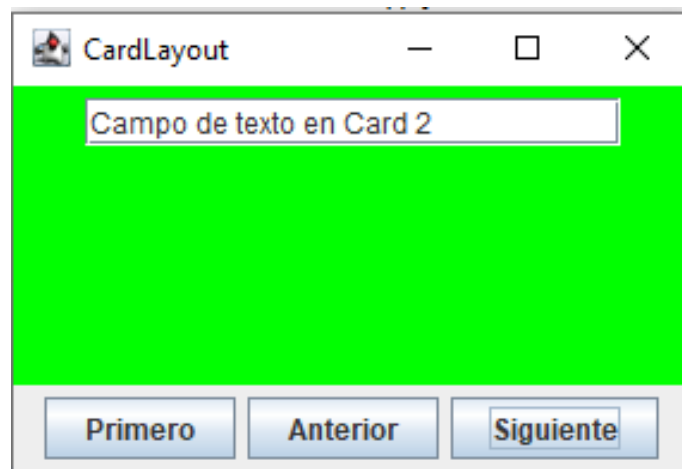
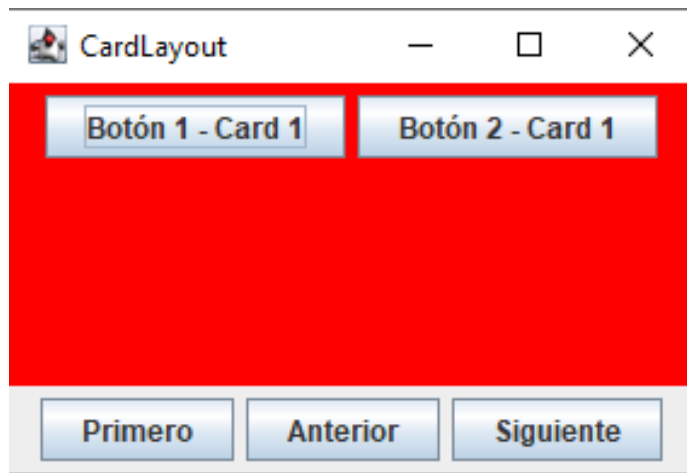
# Grid Bag Layout

- GridBagLayout es similar a GridLayout, sitúa los componentes en una rejilla de un determinado número de filas y columnas, pero permite a los componentes ocupar más de una celda, y que las filas y columnas puedan tener diferente altura/anchura.



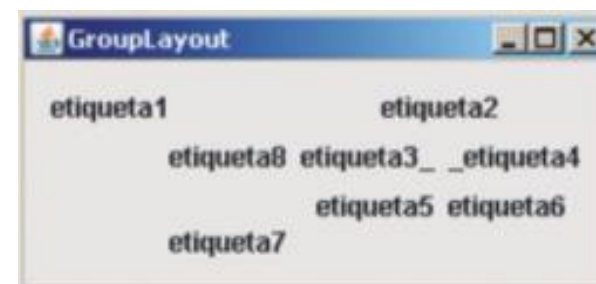
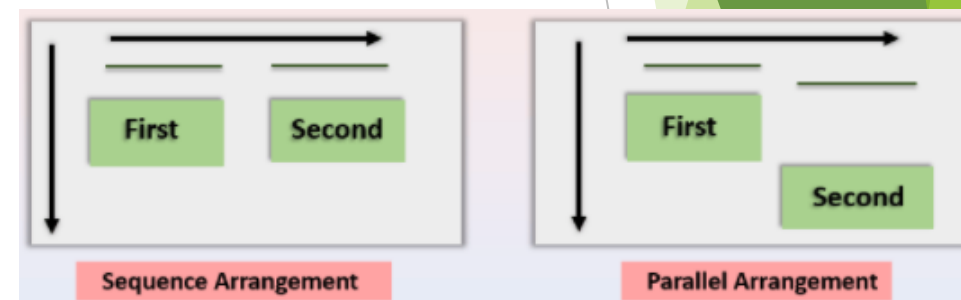
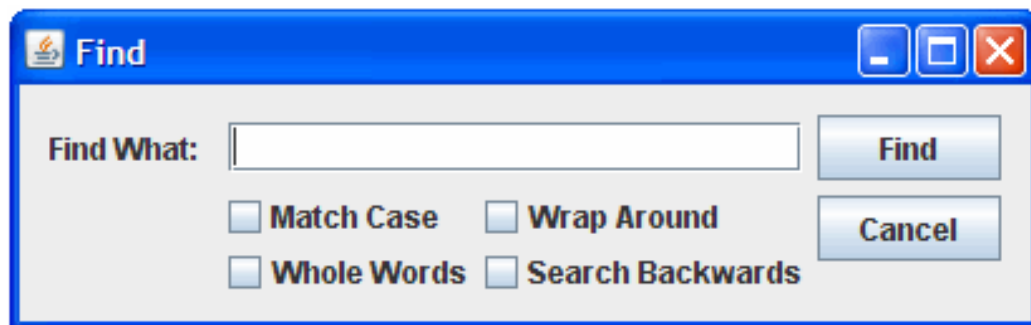
# Card Layout

- CardLayout gestiona varios componentes que comparten el mismo espacio de visualización.  
La decisión de mostrar uno u otro se puede tomar según la acción realizada sobre el control.



# Group Layout

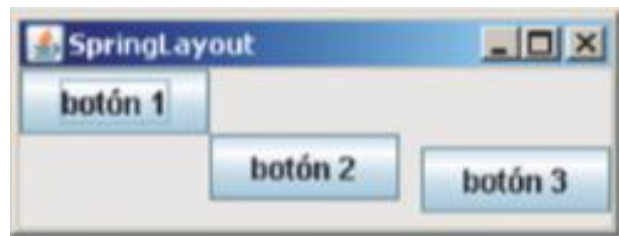
- GroupLayout fue desarrollado para ser utilizado por herramientas de construcción de GUI, pero también puede usarse manualmente. Permite indicar cómo disponer los controles mediante diseños horizontales y verticales totalmente independientes. Como requisito, todos los componentes deben ser definidos dos veces (una por cada dimensión). Dentro de cada dimensión los controles pueden disponerse de forma secuencial o paralela.





# Spring Layout

- SpringLayout también fue desarrollado para ser utilizado por herramientas de construcción de GUI. Permite especificar relaciones de distancia horizontal y vertical entre los bordes de los componentes para realizar un posicionamiento relativo. Para ello, es necesario hacer uso del método *putConstraint* al cual se le pasa como argumento las referencias de los objetos dependiente y ancla, los bordes de esos objetos a tener en cuenta para el posicionamiento y la distancia entre ambos.





# COMPONENTES (CONTROLES)



Enter your name here

TextField

Click Me!

Button

This is Label

Label

Red  
Red  
Green  
Blue

Choice

☒ one ☐ two ☐ three

CheckBox

☒ Alpha ☐ Beta ☐ Charlie

CheckBoxGroup

Mercury  
Venus  
Earth  
Mars  
Jupiter  
Saturn  
Uranus  
Neptune

List

Palette

Swing Containers

Panel

Tabbed Pane

Split Pane

Scroll Pane

Tool Bar

Desktop Pa

Internal Frame

Layered Pane

Swing Controls

Label

Button

ToggleButton

CheckBox

RadioButton

Button Group

ComboBox

List

TextField

TextArea

ScrollBar

Slider

ProgressBar

Formatted Field

Password

Spinner

Separator

Text Pa

Editor Pane

Tree

Table

Swing Menus

Swing Windows

Swing Fillers

AWT

Beans

Java Persistence

Etiqueta

Caja de texto

Elemento 2

Botón

☐ Casilla de verificación

☐ RadioButton 1

☐ RadioButton 2

☐ CheckBox 1 agrupado

☐ CheckBox 2 agrupado

Botón de activación

FlatLaf Demo

File Edit View Font Options Help

Basic Components More Components Data Components Tabs Option Pane Extras

JScrollPane:

JScrollBar:

JSeparator:

JSlider:

JProgressBar:

TitledBorder

HTML:

JLabel HTML

Sample content

text with link

JEditorPane

HTML

Sample content

text with link

JTextPane HTML

Sample content

text with link

Themes:

all

Core Themes

Flat Light

Flat Dark

Flat IntelliJ

Flat Darcula

Current Directory

DemoLaf

IntelliJ Themes

Arc

Arc - Orange

Arc Dark

Arc Dark - Orange

Carbon

Cobalt 2

Cyan light

Flat Light (F1)

right-to-left

enabled

Java 1.8.0\_202; user scale factor 2.0; Segoe UI 24

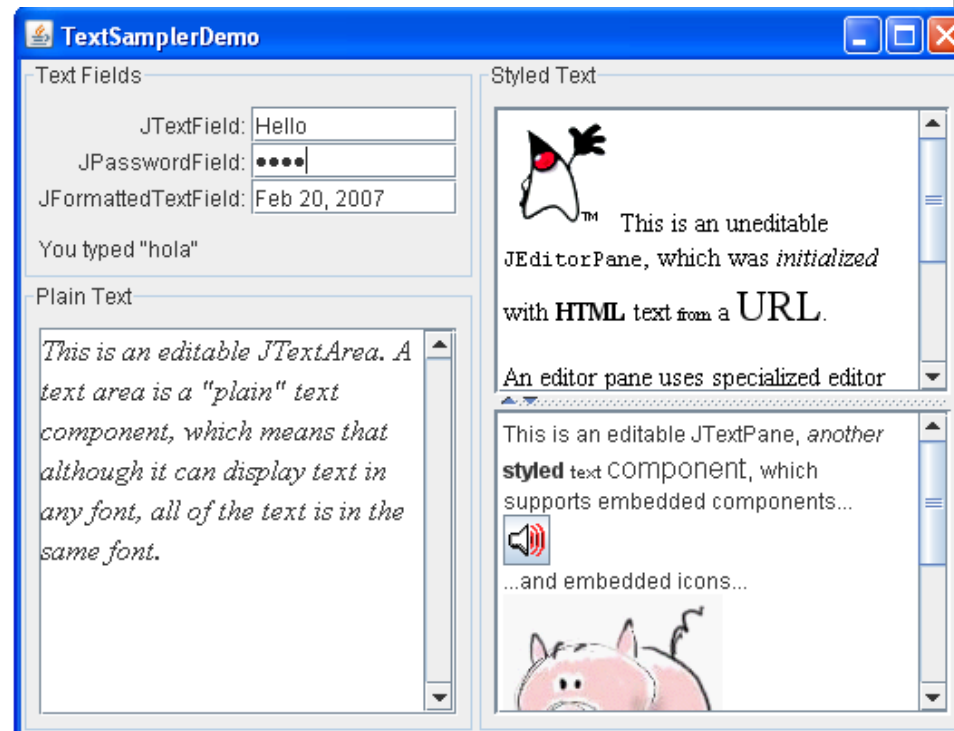
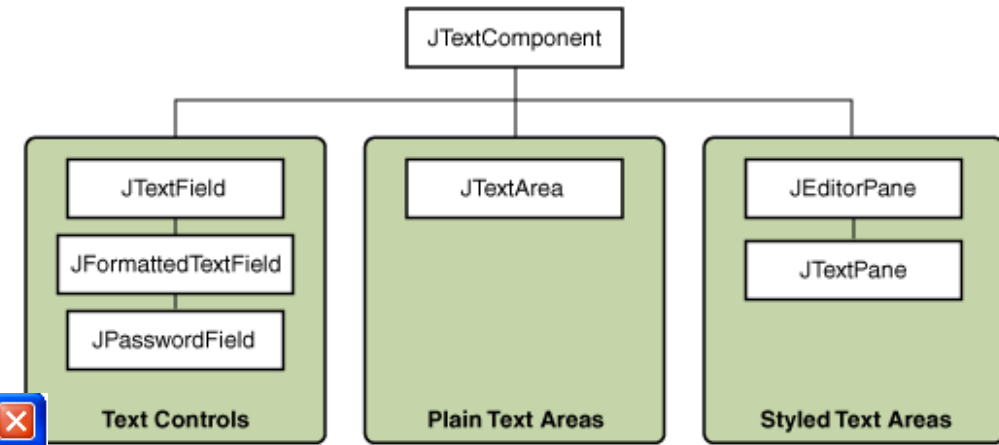
Close

# Componentes de texto



- Existen diversos componentes de texto que podemos utilizar en nuestras aplicaciones, todos ellos en clases que heredan de JTextComponent:

- JTextField
- JFormattedTextField
- JPasswordField
- JTextArea
- JEditorPane
- JTextPane



# JLabel

- JLabel permite crear etiquetas de texto, con html, de imagen o combinaciones de estos elementos.

```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel contentPane = new JPanel();
```

```
BoxLayout gestor = new BoxLayout(contentPane, BoxLayout.Y_AXIS);
contentPane.setLayout(gestor);
```

```
JLabel etiqueta1 = new JLabel();
JLabel etiqueta2 = new JLabel();
JLabel etiqueta3 = new JLabel();
JLabel etiqueta4 = new JLabel();
JLabel etiqueta5 = new JLabel();
```

```
etiqueta1.setText("Etiqueta de texto");
```

```
etiqueta2.setIcon(new ImageIcon("img/supermario.png"));
```

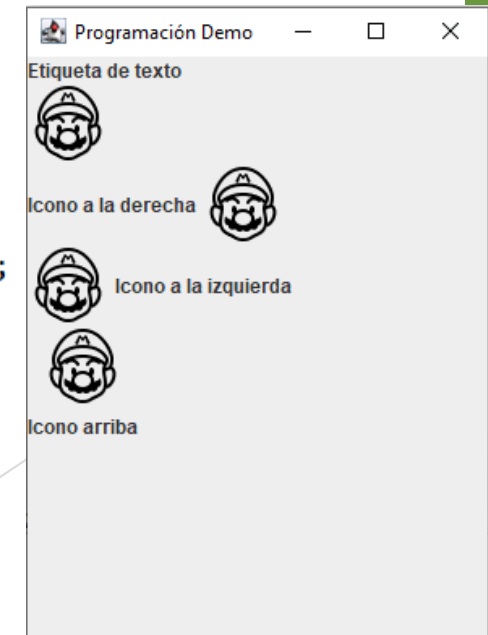
```
etiqueta3.setHorizontalTextPosition(SwingConstants.LEFT);
etiqueta3.setText("Icono a la derecha");
etiqueta3.setIcon(new ImageIcon("img/supermario.png"));
```

```
etiqueta4.setHorizontalTextPosition(JLabel.RIGHT);
etiqueta4.setText("Icono a la izquierda");
etiqueta4.setIcon(new ImageIcon("img/supermario.png"));
```

```
etiqueta5.setText("Icono arriba");
etiqueta5.setIcon(new ImageIcon("img/supermario.png"));
etiqueta5.setHorizontalTextPosition(SwingConstants.CENTER);
etiqueta5.setVerticalTextPosition(JLabel.BOTTOM);
```

```
contentPane.add(etiqueta1);
contentPane.add(etiqueta2);
contentPane.add(etiqueta3);
contentPane.add(etiqueta4);
contentPane.add(etiqueta5);
```

```
frame.setContentPane(contentPane);
frame.setSize(300,400);
frame.setVisible(true);
```



# JButton

- JButton nos permite crear botones con un título o una imagen, o ambos.

```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

ImageIcon leftButtonIcon = new ImageIcon("img/right.gif");
ImageIcon middleButtonIcon = new ImageIcon("img/middle.gif");
ImageIcon rightButtonIcon = new ImageIcon("img/left.gif");

JButton b1 = new JButton("Botón habilitado", leftButtonIcon);
b1.setVerticalTextPosition(AbstractButton.CENTER);
b1.setHorizontalTextPosition(AbstractButton.LEADING);
b1.setMnemonic(KeyEvent.VK_D);

JButton b2 = new JButton("Botón del medio", middleButtonIcon);
b2.setVerticalTextPosition(AbstractButton.BOTTOM);
b2.setHorizontalTextPosition(AbstractButton.CENTER);
b2.setMnemonic(KeyEvent.VK_M);

JButton b3 = new JButton("Botón deshabilitado", rightButtonIcon);
b3.setMnemonic(KeyEvent.VK_E);
b3.setEnabled(false);

b1.setToolTipText("Mensaje de ayuda Botón 1");
b2.setToolTipText("Mensaje de ayuda Botón 2");
b3.setToolTipText("Mensaje de ayuda Botón 3");

JPanel contentPane = new JPanel();
contentPane.add(b1);
contentPane.add(b2);
contentPane.add(b3);
frame.setContentPane(contentPane);

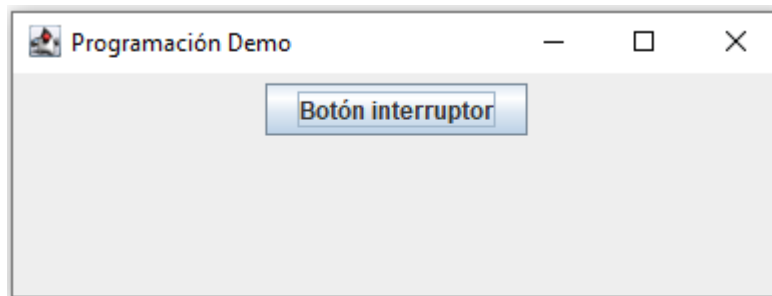
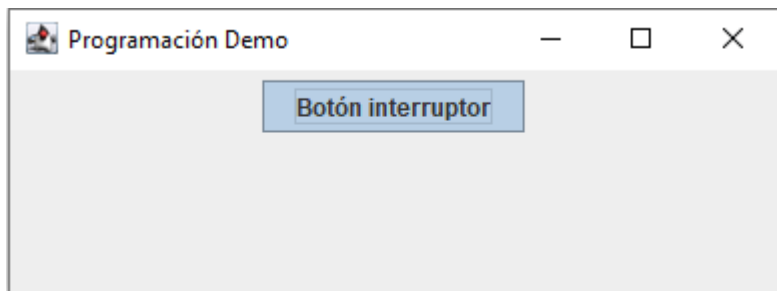
frame.pack();
frame.setVisible(true);
```



# JToggleButton

- JToggleButton nos permite crear botones a modo de interruptor, que están o no pulsados.

```
JFrame frame = new JFrame("Programación Demo");  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
JToggleButton b1 = new JToggleButton("Botón interruptor", true);  
  
JPanel contentPane = new JPanel();  
contentPane.add(b1);  
frame.setContentPane(contentPane);  
  
frame.setSize(400, 150);  
frame.setVisible(true);
```



# JRadioButton

- JRadioButton nos permite crear un grupo de botones donde solo uno de ellos puede estar seleccionado.

```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel contentPane = new JPanel();
contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.Y_AXIS));
contentPane.setPreferredSize(new Dimension(300, 100));

JRadioButton rdbtnOpcion= new JRadioButton("Opcion 1", true);
rdbtnOpcion.setBounds(43, 194, 109, 23);
contentPane.add(rdbtnOpcion);

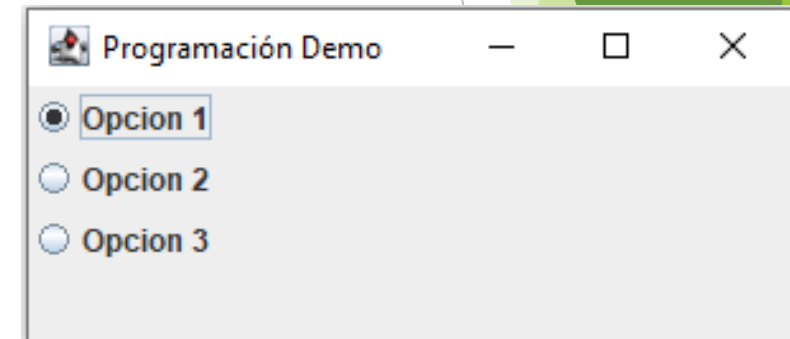
JRadioButton rdbtnOpcion_1 = new JRadioButton("Opcion 2", false);
rdbtnOpcion_1.setBounds(43, 220, 109, 23);
contentPane.add(rdbtnOpcion_1);

JRadioButton rdbtnOpcion_2 = new JRadioButton("Opcion 3", false);
rdbtnOpcion_2.setBounds(43, 246, 109, 23);
contentPane.add(rdbtnOpcion_2);

ButtonGroup bgroup = new ButtonGroup();
bgroup.add(rdbtnOpcion);
bgroup.add(rdbtnOpcion_1);
bgroup.add(rdbtnOpcion_2);

frame.setContentPane(contentPane);

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





# JCheckBox

- JCheckBox nos permite crear un grupo de botones donde cada uno de ellos puede estar seleccionado independientemente de los demás.

```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel contentPane = new JPanel();
contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.Y_AXIS));
contentPane.setPreferredSize(new Dimension(300, 100));
```

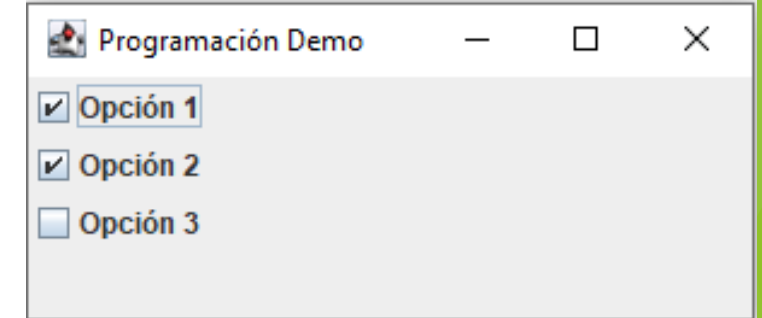
```
JCheckBox chckbxOpcion = new JCheckBox("Opción 1", true);
contentPane.add(chckbxOpcion);
```

```
JCheckBox chckbxNewCheckBox = new JCheckBox("Opción 2", true);
contentPane.add(chckbxNewCheckBox);
```

```
JCheckBox chckbxOpcion_1 = new JCheckBox("Opción 3", false);
contentPane.add(chckbxOpcion_1);
```

```
frame.setContentPane(contentPane);
```

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





# JComboBox

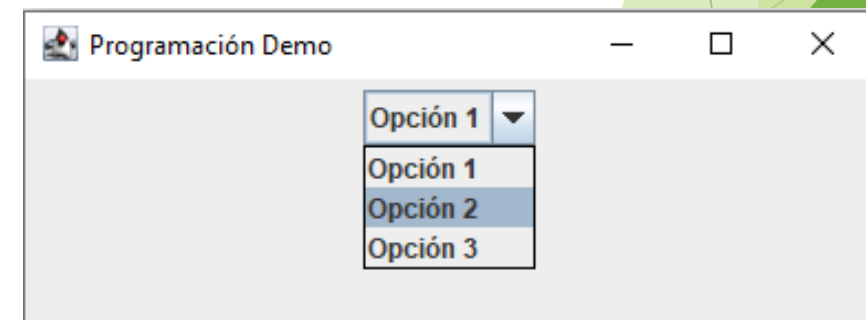
- JComboBox permite crear un elemento que le da al usuario la posibilidad de elegir entre varias opciones.

```
JFrame frame = new JFrame("Programación Demo");  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
JComboBox<String> comboBox = new JComboBox<>();  
comboBox.addItem("Opción 1");  
comboBox.addItem("Opción 2");  
comboBox.addItem("Opción 3");
```

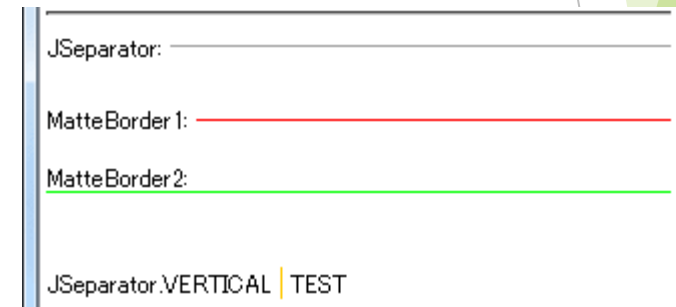
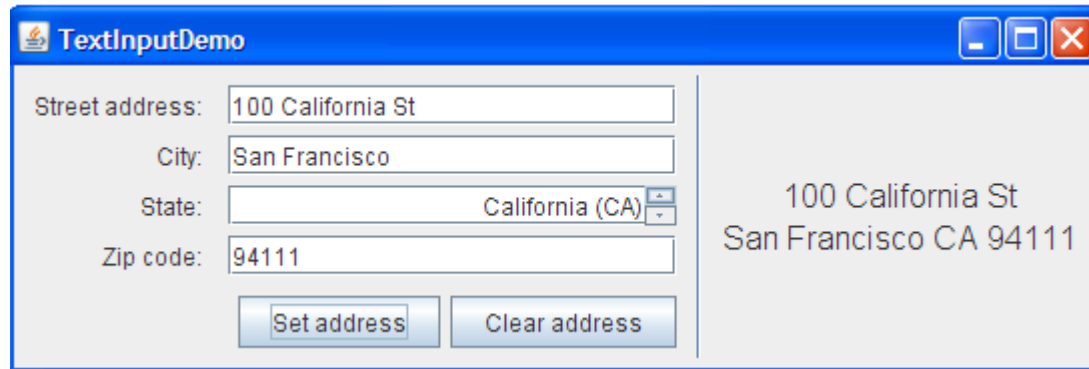
```
JPanel contentPane = new JPanel();  
contentPane.add(comboBox);  
frame.setContentPane(contentPane);
```

```
frame.setSize(400, 150);  
frame.setVisible(true);
```



# JSeparator

- JSeparator nos permite introducir divisiones en nuestra interfaz gráfica, a modo de línea horizontal, vertical o espacio vacío.



# JSlider

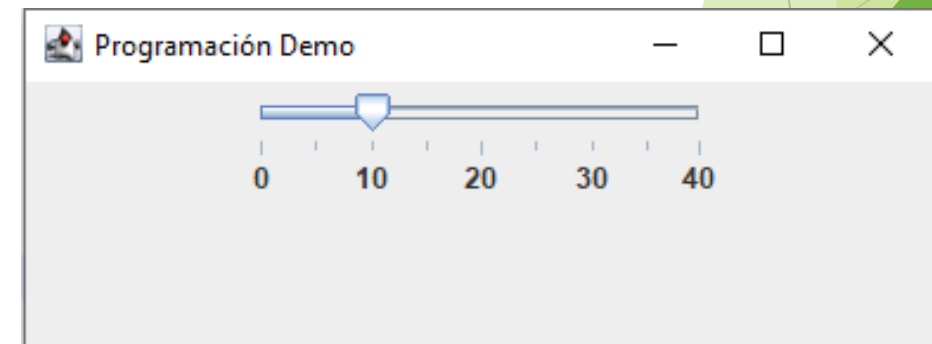
- JSlider proporciona un elemento con el que el usuario puede introducir valores moviendo un indicador en una barra horizontal o vertical, o pulsando un punto de esa barra.

```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 40, 10);
slider.setMinorTickSpacing(5);
slider.setMajorTickSpacing(20);
slider.setPaintTicks(true);
slider.setPaintLabels(true);
slider.setLabelTable(slider.createStandardLabels(10));

JPanel contentPane = new JPanel();
contentPane.add(slider);
frame.setContentPane(contentPane);

frame.setSize(400, 150);
frame.setVisible(true);
```



# JSpinner

- JSpinner proporciona un cuadro de texto que permite al usuario escribir un valor y que contiene dos botones pequeños que posibilitan incrementar o decrementar el valor contenido.

```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

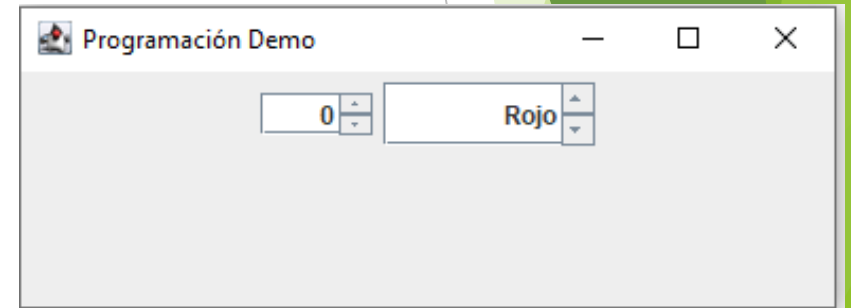
SpinnerNumberModel snm = new SpinnerNumberModel(
    Integer.valueOf(0),
    Integer.valueOf(0),
    Integer.valueOf(100),
    Integer.valueOf(5)
);

JSpinner spnNumber = new JSpinner(snm);

String[] colors = {"Rojo", "Verde", "Azul"};
SpinnerListModel snl = new SpinnerListModel(colors);
JSpinner spnList = new JSpinner(snl);
spnList.setPreferredSize(new Dimension(100, 30));

JPanel contentPane = new JPanel();
contentPane.add(spnNumber);
contentPane.add(spnList);
frame.setContentPane(contentPane);

frame.setSize(400, 150);
frame.setVisible(true);
```



# JList

- JList muestra un listado de objetos permitiendo al usuario seleccionar uno o más elementos.

```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

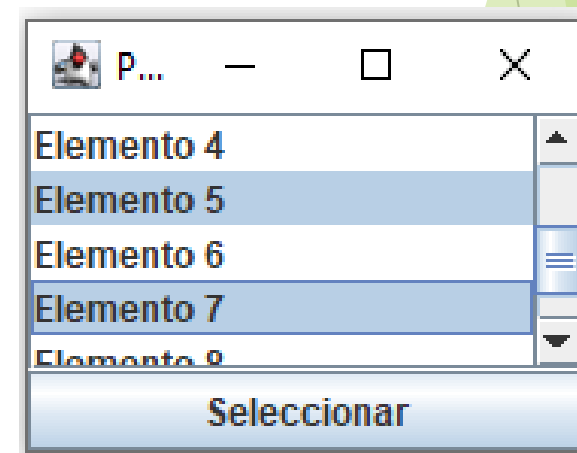
final int MAX = 10;
String[] listElems = new String[MAX];
for (int i = 0; i < MAX; i++) {
    listElems[i] = "Elemento " + i;
}

JList list = new JList(listElems);
JScrollPane scroll = new JScrollPane(list);

JButton btnGet = new JButton("Seleccionar");

JPanel contentPane = new JPanel();
contentPane.setLayout(new BorderLayout());
contentPane.add(scroll, BorderLayout.CENTER);
contentPane.add(btnGet, BorderLayout.SOUTH);
frame.setContentPane(contentPane);

frame.setSize(200, 150);
frame.setVisible(true);
```



# JTree

- JTree nos permite visualizar un árbol de nodos en nuestra aplicación.

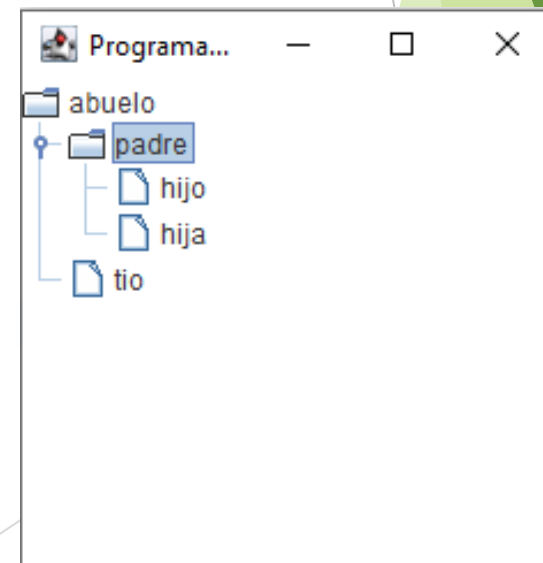
```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

DefaultMutableTreeNode abuelo = new DefaultMutableTreeNode("abuelo");
DefaultTreeModel modelo = new DefaultTreeModel(abuelo);
JTree tree = new JTree(modelo);
DefaultMutableTreeNode padre = new DefaultMutableTreeNode("padre");
DefaultMutableTreeNode tio = new DefaultMutableTreeNode("tio");
DefaultMutableTreeNode hijo = new DefaultMutableTreeNode("hijo");
DefaultMutableTreeNode hija = new DefaultMutableTreeNode("hija");

modelo.insertNodeInto(padre, abuelo, 0);
modelo.insertNodeInto(tio, abuelo, 1);
modelo.insertNodeInto(hijo, padre, 0);
modelo.insertNodeInto(hija, padre, 1);

frame.getContentPane().add(tree);

frame.setSize(250, 250);
frame.setVisible(true);
```





# JTable

- JTable nos permite construir un elemento de tabla con filas y columnas en nuestra aplicación.

```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

String[] columns = {"Code", "Name", "High", "Low",
    "Close", "Volume", "Change", "Change %"};

Object[][] data = {
    {"MBF", "CITYGROUP", 10.16, 10.16, 10.16, 200, 0.08, 0.79},
    {"MBL", "BANK OF AMERICA", 12.66, 12.66, 12.66, 6600, 0.13, 1.04},
    {"MJP", "Morgan Stanley Dean Witter & Co.", 24.97, 24.97, 24.97, 1000, -0.04, -0.16}
};

JTable table = new JTable(data, columns);
JScrollPane scrollPane = new JScrollPane(table);
table.setFillsViewportHeight(true);

JLabel lblHeading = new JLabel("Acciones Dow Jones");
lblHeading.setFont(new Font("Arial", Font.TRUETYPE_FONT, 24));

frame.getContentPane().setLayout(new BorderLayout());
frame.getContentPane().add(lblHeading, BorderLayout.PAGE_START);
frame.getContentPane().add(scrollPane, BorderLayout.CENTER);

frame.setSize(550, 200);
frame.setVisible(true);
```



The screenshot shows a Java Swing window titled "Programación Demo" with standard window controls (minimize, maximize, close). Inside the window, there is a heading "Acciones Dow Jones" and a table with 8 columns: Code, Name, High, Low, Close, Volume, Change, and Change %. The table contains three rows of data:

Code	Name	High	Low	Close	Volume	Change	Change %
MBF	CITYGRO...	10.16	10.16	10.16	200	0.08	0.79
MBL	BANK OF ...	12.66	12.66	12.66	6600	0.13	1.04
MJP	Morgan S...	24.97	24.97	24.97	1000	-0.04	-0.16



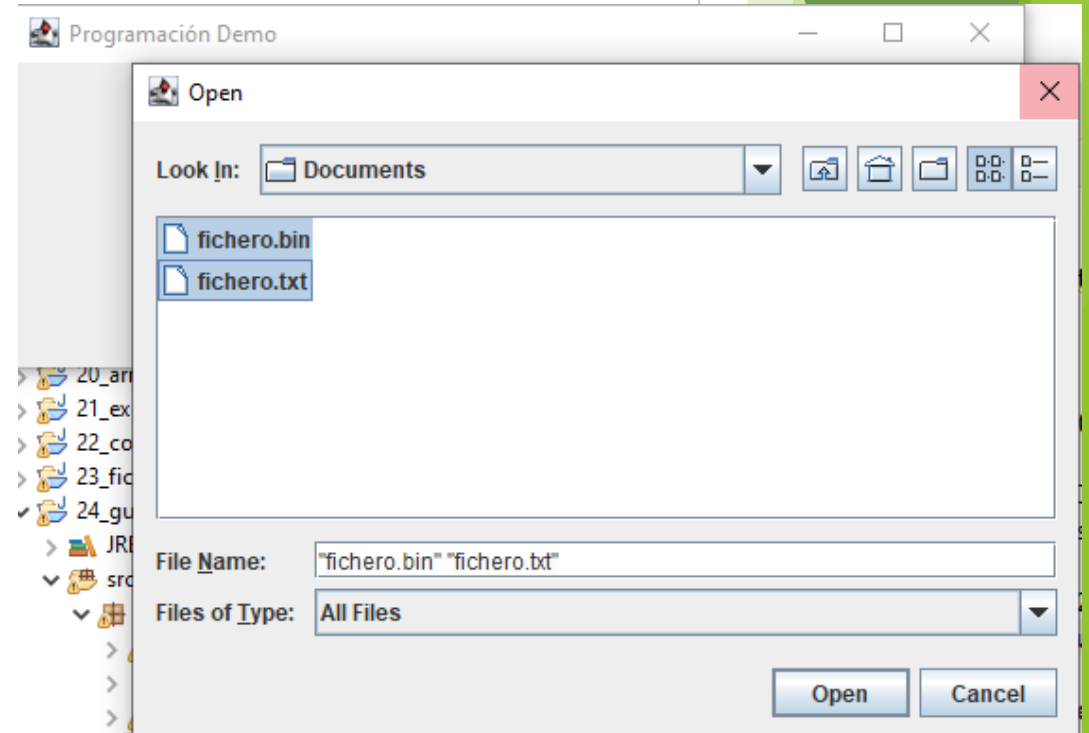
# JFileChooser

- JFileChooser permite mostrar una ventana de diálogo para navegar por el sistema de ficheros y que el usuario pueda seleccionar elementos.

```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

frame.setSize(550, 200);
frame.setVisible(true);

JFileChooser fc = new JFileChooser();
fc.setMultiSelectionEnabled(true);
fc.setCurrentDirectory(new File("C:\\Users\\Héctor\\Documents"));
int retVal = fc.showOpenDialog(frame);
```



# JMenuBar

- JMenuBar nos permite incorporar una barra de menús a la parte superior de la ventana de nuestra aplicación.

```
JFrame frame = new JFrame("Programación Demo");  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
JMenuBar menuBar = new JMenuBar();  
JMenu menu, submenu;  
JMenuItem menuItem;  
JRadioButtonMenuItem rbMenuItem;  
JCheckBoxMenuItem cbMenuItem;
```

```
menu = new JMenu("Primer menú");  
menu.setMnemonic(KeyEvent.VK_P);  
menuBar.add(menu);
```

```
menuItem = new JMenuItem("A text-only menu item", KeyEvent.VK_T);  
menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_1, ActionEvent.ALT_MASK));  
menu.add(menuItem);
```

```
menuItem = new JMenuItem("Both text and icon", new ImageIcon("img/middle.gif"));  
menuItem.setMnemonic(KeyEvent.VK_B);  
menu.add(menuItem);
```

```
menuItem = new JMenuItem(new ImageIcon("img/middle.gif"));  
menuItem.setMnemonic(KeyEvent.VK_D);  
menu.add(menuItem);
```

```
menu.addSeparator();  
ButtonGroup group = new ButtonGroup();  
rbMenuItem = new JRadioButtonMenuItem("A radio button menu item");  
rbMenuItem.setSelected(true);  
rbMenuItem.setMnemonic(KeyEvent.VK_R);  
group.add(rbMenuItem);  
menu.add(rbMenuItem);
```

```
rbMenuItem = new JRadioButtonMenuItem("Another one");  
rbMenuItem.setMnemonic(KeyEvent.VK_O);  
group.add(rbMenuItem);  
menu.add(rbMenuItem);
```

```
menu.addSeparator();  
cbMenuItem = new JCheckBoxMenuItem("A check box menu item");  
cbMenuItem.setMnemonic(KeyEvent.VK_C);  
menu.add(cbMenuItem);
```



# JMenuBar

```
cbMenuItem = new JCheckBoxMenuItem("Another one");
cbMenuItem.setMnemonic(KeyEvent.VK_H);
menu.add(cbMenuItem);

menu.addSeparator();
submenu = new JMenu("A submenu");
submenu.setMnemonic(KeyEvent.VK_S);

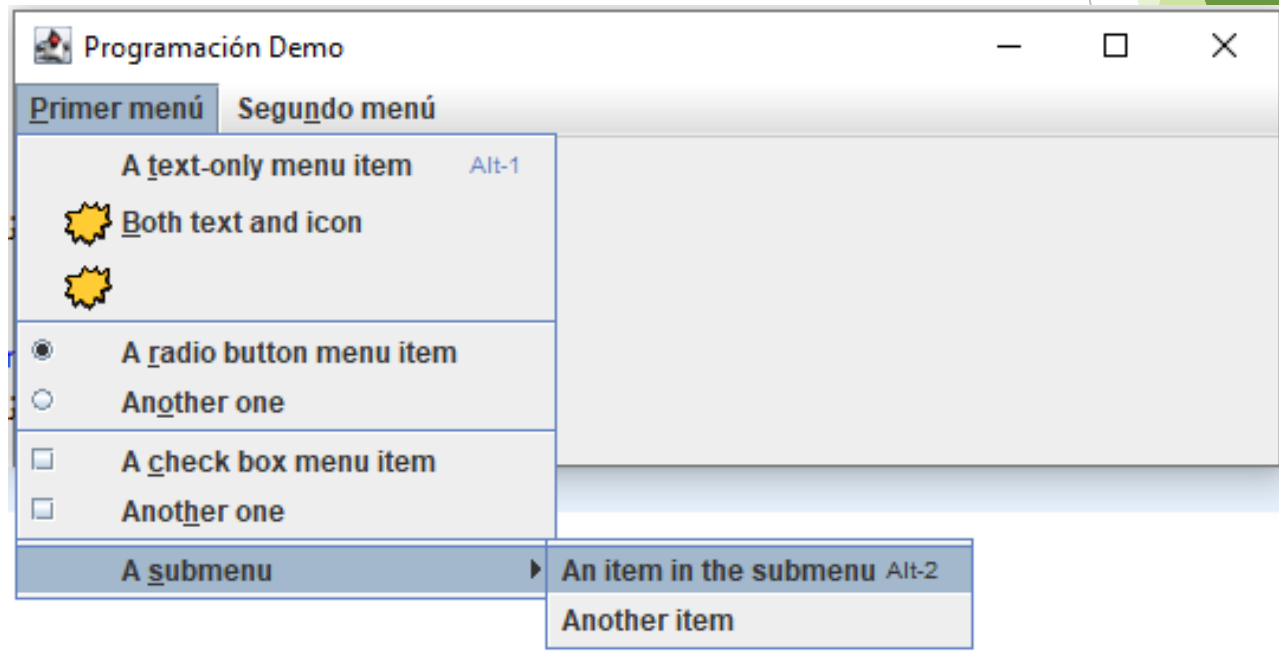
menuItem = new JMenuItem("An item in the submenu");
menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_2, ActionEvent.ALT_MASK));
submenu.add(menuItem);

menuItem = new JMenuItem("Another item");
submenu.add(menuItem);
menu.add(submenu);

menu = new JMenu("Segundo menú");
menu.setMnemonic(KeyEvent.VK_N);
menuBar.add(menu);

frame.setJMenuBar(menuBar);

frame.setSize(550, 200);
frame.setVisible(true);
```



# JTabbedPane

- JTabbedPane permite al usuario seleccionar entre un grupo de componentes haciendo clic en una pestaña.

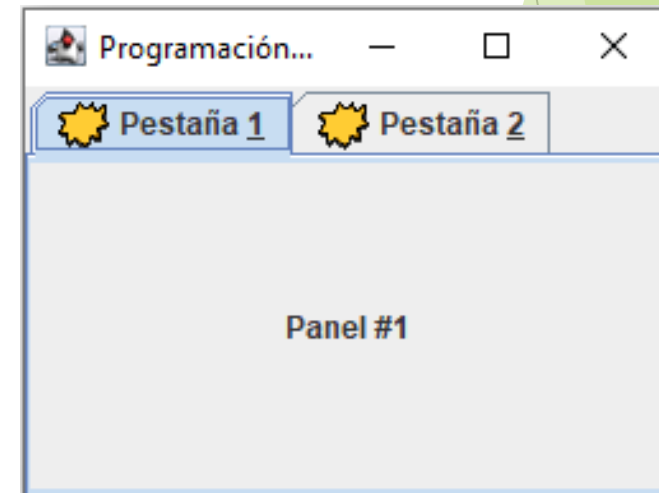
```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JTabbedPane tabbedPane = new JTabbedPane();
ImageIcon icon = new ImageIcon("img/middle.gif");

JPanel panel1 = new JPanel(false);
JLabel filler1 = new JLabel("Panel #1");
filler1.setHorizontalAlignment(JLabel.CENTER);
panel1.setLayout(new GridLayout(1, 1));
panel1.add(filler1);
tabbedPane.addTab("Pestaña 1", icon, panel1, "Primera pestaña");
tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);

JPanel panel2 = new JPanel(false);
JLabel filler2 = new JLabel("Panel #2");
filler2.setHorizontalAlignment(JLabel.CENTER);
panel2.setLayout(new GridLayout(1, 1));
panel2.add(filler2);
tabbedPane.addTab("Pestaña 2", icon, panel2, "Segunda pestaña");
tabbedPane.setMnemonicAt(1, KeyEvent.VK_2);

frame.getContentPane().add(tabbedPane);
frame.setSize(270, 200);
frame.setVisible(true);
```



# JScrollPane

- JScrollPane proporciona una vista con scroll de un componente.

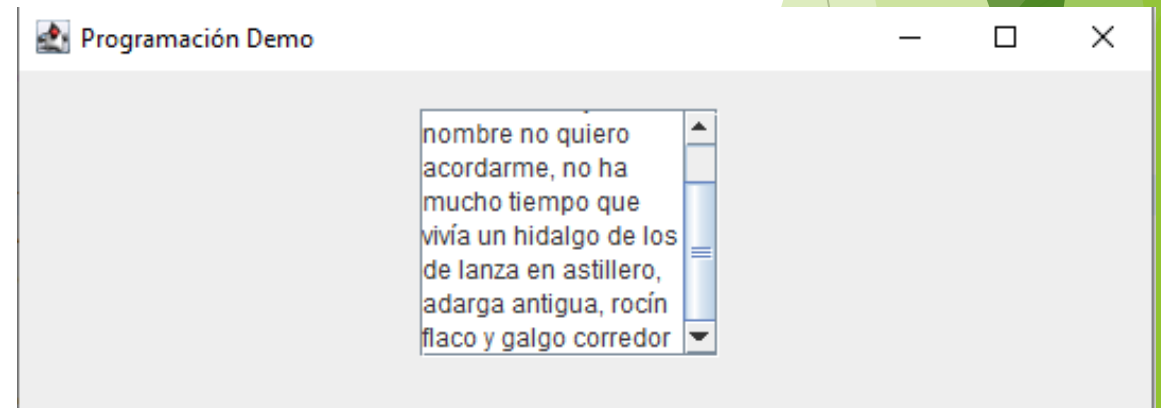
```
JFrame frame = new JFrame("Programación Demo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JTextArea textArea = new JTextArea("En un lugar de la Mancha");
textArea.setBounds(189, 18, 141, 117);
textArea.setWrapStyleWord(true);
textArea.setLineWrap(true);

JScrollPane scroll = new JScrollPane(textArea);
scroll.setBounds(189, 18, 141, 117);

frame.getContentPane().setLayout(null);
frame.getContentPane().add(scroll);

frame.setSize(550, 200);
frame.setVisible(true);
```

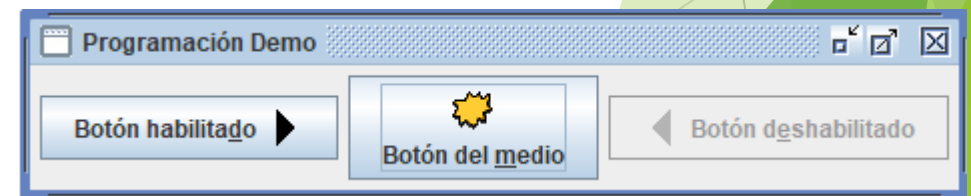
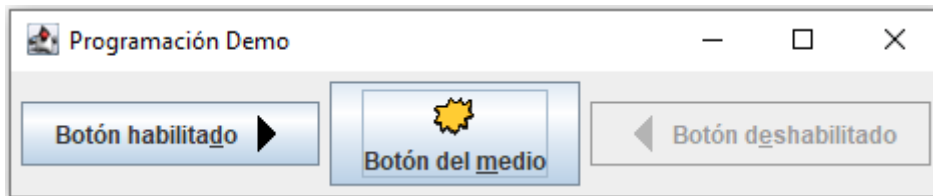
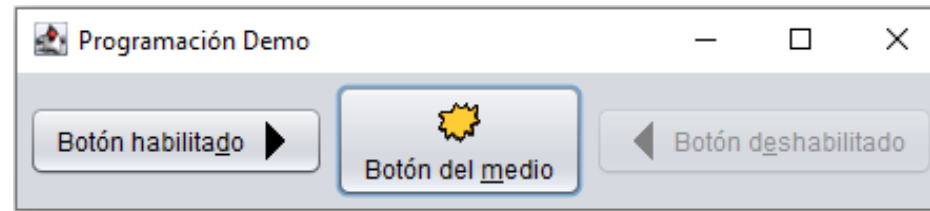




# Look and Feel

- La arquitectura de Java Swing permite cambiar con facilidad el “look and feel” de la aplicación, es decir, la apariencia de los componentes, utilizando la clase *UIManager*.

```
try {  
    JFrame.setDefaultLookAndFeelDecorated(true);  
    UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");  
    //UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");  
} catch (Exception e) {  
    e.printStackTrace();  
}
```



# EVENTOS

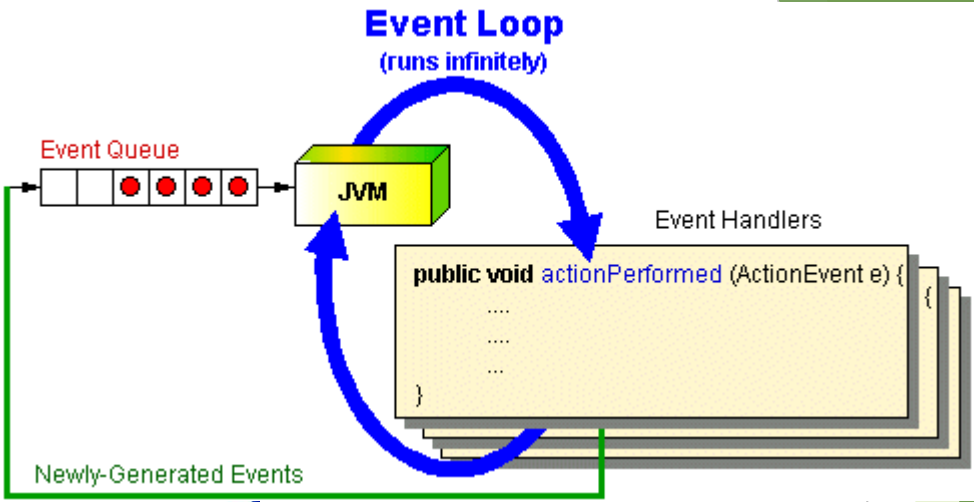


Examples

Control in action: Button



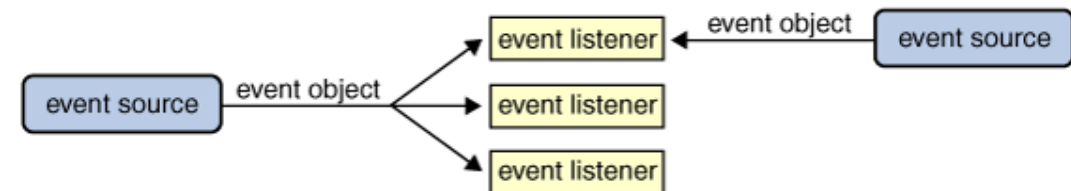
Ok Button clicked.



# Eventos y listeners

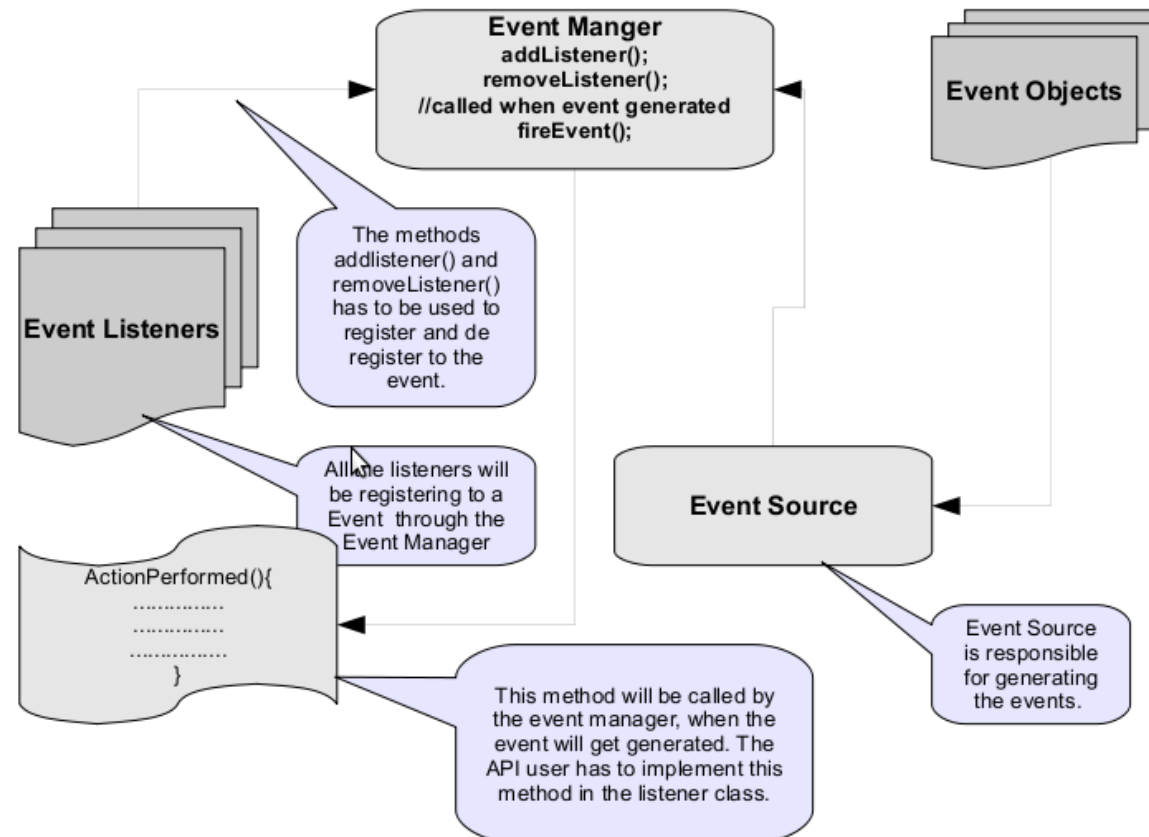


- Llamamos eventos a las interacciones que puede tener un usuario con la GUI. Cuando un usuario efectúa una acción sobre un componentes (por ejemplo, hacer clic en un botón) se genera un objeto de una de las clases de eventos que lo identifica y que contiene información relativa al mismo: tipo de evento, objeto en el que fue generado, momento en que se ha producido, tecla pulsada (si es un evento de teclado)... este evento finalmente provocará la ejecución de un método en nuestro programa.
- El tratamiento de un evento que ocurre en un objeto (por ejemplo en un botón) no se realiza en ese mismo objeto, sino que se delega en otro objeto diferente (listener). Un listener es un objeto que espera “escuchando” hasta que recibe un evento, cuando lo recibe, lo procesa y genera una respuesta a ese evento (por ejemplo, cuando un usuario pulsa un botón cambia el texto de una etiqueta).



# Clases Evento y Listener

- Las clases que representan a los eventos son del tipo *XXXEvent*, en donde XXX hace referencia la tipo de evento: *ActionEvent*, *ComponentEvent*, *KeyEvent*, *MouseEvent*, *InputEvent*, *TextEvent*... Las clases que representan a los listeners son del tipo *XXXListener*, en donde XXX hace referencia al tipo de listener: *ActionListener*, *KeyListener*, *FocusListener*, *MouseListener*...



# Manejo de eventos con listeners



- ▶ Para crear un mecanismo de manejo y reacción a eventos en un componente debemos:

- ▶ Declarar una clase que implemente uno de los listener para que trate eventos de ese tipo (por ejemplo, clase que implemente ActionListener para que “escuche” clics en botones)

```
public class BotonListener implements ActionListener {
```

- ▶ Registrar el listener para el componente que queremos que responda a sus eventos.

```
boton.addActionListener(new BotonListener());
```

- ▶ Incluir código fuente para implementar los métodos del listener con lo que queremos que suceda.

```
@Override  
public void actionPerformed(ActionEvent e) {  
    // Código de respuesta a la acción  
}
```

# Clases listener



- ▶ Para no tener que crear una clase nueva para el listener, se puede implementar la interfaz listener en la misma clase en la que estamos trabajando, o mediante una clase anónima.
- ▶ Las clases anónimas son un mecanismo en java que permite que el código sea más conciso y fácil de leer. Permiten declarar y instanciar una clase al mismo tiempo, sin asignarle un nombre a la clase y sin que se pueda reutilizar. El formato de una clase anónima es el operador *new* seguido del nombre de la interfaz a implementar o la clase a extender, parámetros del constructor y cuerpo de la clase entre llaves.

```
JTextArea areaTexto = new JTextArea("Evento:");
areaTexto.setEditable(false);
boton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        areaTexto.setText(areaTexto.getText() + " Botón pulsado\n");
    }
});
```



# Principales Clases listener



ActionListener	Se produce al hacer click en un componente, también si se pulsa Enter teniendo el foco en el componente.	<code>public void actionPerformed(ActionEvent e)</code>	<ul style="list-style-type: none"><li>▪ <b>JButton</b>: click o pulsar Enter con el foco activado en él.</li><li>▪ <b>JList</b>: doble click en un elemento de la lista.</li><li>▪ <b>JMenuItem</b>: selecciona una opción del menú.</li><li>▪ <b>TextField</b>: al pulsar Enter con el foco activado.</li></ul>
KeyListener	Se produce al pulsar una tecla. según el método cambiara la forma de pulsar la tecla.	<code>public void keyTyped(KeyEvent e)</code> <code>public void keyPressed(KeyEvent e)</code> <code>public void keyReleased(KeyEvent e)</code>	Cuando pulsamos una tecla, según el Listener: <ul style="list-style-type: none"><li>▪ <b>keyTyped</b>: al pulsar y soltar la tecla.</li><li>▪ <b>keyPressed</b>: al pulsar la tecla.</li><li>▪ <b>keyReleased</b>: al soltar la tecla.</li></ul>
FocusListener	Se produce cuando un componente gana o pierde el foco, es decir, que esta seleccionado.	<code>public void focusGained(FocusEvent e)</code> <code>public void focusLost(FocusEvent e)</code>	Recibir o perder el foco.

# Principales Clases listener



MouseListener	Se produce cuando realizamos una acción con el ratón.	<pre>public void mouseClicked(MouseEvent e)  public void mouseEntered(MouseEvent e)  public void mouseExited(MouseEvent e)  public void mousePressed(MouseEvent e)  public void mouseReleased(MouseEvent e)</pre>	Según el Listener: <ul style="list-style-type: none"><li>■ mouseClicked: pinchar y soltar.</li><li>■ mouseEntered: entrar en un componente con el puntero.</li><li>■ mouseExited: salir de un componente con el puntero</li><li>■ mousePressed: presionar el botón.</li><li>■ mouseReleased: soltar el botón.</li></ul>
MouseMotionListener	Se produce con el movimiento del mouse.	<pre>public void mouseDragged(MouseEvent e)  public void mouseMoved(MouseEvent e)</pre>	Según el Listener: <ul style="list-style-type: none"><li>■ mouseDragged: click y arrastrar un componente.</li><li>■ mouseMoved: al mover el puntero sobre un elemento</li></ul>

► Para un listado más detallado de qué componentes soportan un determinado tipo de eventos

► <https://docs.oracle.com/javase/tutorial/uiswing/events/eventsandcomponents.html>

# Ejemplo eventos / listeners

GUI APP - Programación 1º DAM

Pulsa el botón

Elige una opción:

Nombre Elegido

☐ Opción 1

☐ Opción 2

☐ Opción 3

Estás sobre la

Escribe el nombre de una persona

Solo se puede escribir dígitos

GUI APP - Programación 1º DAM

Pantalla 2 de la aplicación