

TEMA 11: INTERFAZ GRÁFICA DE USUARIO

1.- INTRODUCCIÓN

La interface gráfica de usuario (GUI – *Graphical User Interface*) es el aspecto más importante de cualquier aplicación. Una aplicación sin un interfaz fácil, impide que los usuarios saquen el máximo rendimiento del programa. Java proporciona los elementos básicos para construir interfaces gráficas de usuario a través del **AWT** y **Swing**.

El AWT es la parte de **Java** que se ocupa de construir interfaces gráficas de usuario. Aunque el AWT ha estado presente en **Java** desde la versión 1.0, la versión 1.1 representó un cambio notable, sobre todo en lo que respecta al **modelo de eventos**. La versión 1.2 incorporó un modelo distinto de componentes llamado **Swing**, que también está disponible en la versión 1.1 como package adicional.

La versión del **AWT** que se proporciona con el JDK se desarrolló en sólo dos meses y, aunque se ha retocado posteriormente, es la parte más débil de todo lo que representa Java como lenguaje. El entorno que ofrece es demasiado simple, no se han tenido en cuenta las ideas de entornos gráficos más novedosos.

Por su parte, **Swing**, proporciona una serie de componentes muy bien descritos y especificados de forma que su presentación visual es independiente de la plataforma en que se ejecute el applet o la aplicación que utilice estas clases. **Swing** simplemente extiende el AWT añadiendo un conjunto de componentes, *JComponents*, y sus clases de soporte. Hay un conjunto de componentes de **Swing** que son análogos a los de AWT, aunque **Swing** también proporciona otros nuevos como árboles, pestañas, etc.

1.1. ¿Qué es swing?

Son muchas las ventajas que ofrece el uso de Swing. Por ejemplo, la navegación con el teclado es automática, cualquier aplicación Swing se puede utilizar sin ratón, sin tener que escribir ni una línea de código adicional. Las etiquetas de información, o “*tool tips*”, se pueden crear con una sola línea de código. Además, Swing aprovecha la circunstancia de que sus Componentes no están renderizados sobre la pantalla por el sistema operativo para soportar lo que se llama “*plugable look and feel*”, es decir, que la apariencia de la aplicación se adapta dinámicamente al sistema operativo y plataforma en que esté corriendo.

El paso de AWT a Swing es muy sencillo y no hay que destacar nada de lo que se haya hecho con el AWT. Afortunadamente, los programadores de Swing han tenido compasión y, en la mayoría de los casos es suficiente con añadir una “J” al componente AWT para que se convierta en un componente Swing.

Es muy importante entender y asimilar el hecho de que Swing es una extensión del AWT, y no un sustituto encaminado a reemplazarlo. Aunque esto sea verdad en algunos casos en los que los componentes de Swing se corresponden a componentes del AWT; por ejemplo, el **JButton** de Swing puede considerarse como un sustituto del **Button** del AWT, y una vez que se usen los botones de Swing se puede tomar la decisión de no volver a utilizar jamás un botón de AWT, pero, la funcionalidad básica de Swing descansa sobre el AWT.

El paquete **javax.swing** proporciona un conjunto de componentes “ligeros” (escritos totalmente en lenguaje Java) que presentan automáticamente el estilo de interfaz de cualquier sistema operativo. Los componentes Swing son versiones 100% Java del conjunto de componentes AWT existente, tales como botón, barra de desplazamiento y etiqueta, con un conjunto de componentes adicionales, como vista de árbol, tabla y panel con pestañas. En swing se incluye también la posibilidad de etiquetas y botones que tengan texto e imágenes.

JFC es la abreviatura de Java Foundation Classes, que comprende un grupo de características para ayudar a construir interfaces gráficos de usuario (GUIs). Las tres primeras características del JFC fueron implementadas sin ningún código nativo, tratando sólo con el API definido en el JDK 1.1. Como resultado, se convirtieron en una extensión del JDK 1.1. Esta versión fue liberada como JFC 1.1, que algunas veces es

llamada 'Versión Swing'. El API del JFC 1.1 es conocido como el API Swing. "Swing" era el nombre clave del proyecto que desarrolló los nuevos componentes. Aunque no es un nombre oficial, frecuentemente se usa para referirse a los nuevos componentes y al API relacionado. Está inmortalizado en los nombres de paquete del API Swing, que empiezan con "javax.swing". Resumiendo Swing es una de las API's del JFC.

2. CREACIÓN DE UNA INTERFAZ GRÁFICA DE USUARIO

Para construir una interfaz gráfica de usuario hace falta:

1. Un "contenedor" o **container**, que es la ventana o parte de la ventana donde se situarán los componentes (botones, barras de desplazamiento, etc.) y donde se realizarán los dibujos. Se correspondería con un **formulario** o una **picture box** de **Visual Basic**.
2. Los **componentes**: menús, botones de comando, barras de desplazamiento, cajas y áreas de texto, botones de opción y selección, etc. Se corresponderían con los **controles** de **Visual Basic**.
3. El **modelo de eventos**. El usuario controla la aplicación actuando sobre los componentes, de ordinario con el ratón o con el teclado. Cada vez que el usuario realiza una determinada acción, se produce el **evento** correspondiente, que el Sistema Operativo transmite al AWT o a Swing. AWT o Swing crea un **objeto** de una determinada clase de evento, derivada de **AWTEvent** (y algunas otras que han surgido con *swing*). Este **evento** es transmitido a un determinado **método** para que lo gestione. En **Visual Basic** el entorno de desarrollo crea automáticamente el procedimiento que va a gestionar el evento (uniendo el nombre del control con el tipo del evento mediante el carácter `_`) y el usuario no tiene más que introducir el código. En **Java** esto es un poco más complicado: el componente u objeto que recibe el evento debe "registrar" o indicar previamente qué objeto se va a hacer cargo de gestionar ese evento.

En los siguientes apartados se verán los dos primeros aspectos, dejando para el próximo tema el modelo de eventos, aunque, será inevitable en muchos casos, no hablar de cómo se gestionan estos eventos.

3. CONTRUCCIÓN DE UN GUI.

Vamos a empezar viendo cómo construir una GUI utilizando AWT (paquete **java.awt**), ya qué cómo se ha comentado antes Swing no sustituye a AWT, y posteriormente veremos las diferencias entre AWT y Swing. Por tanto, para la construcción de una interface gráfica de usuario con AWT, deberemos dar los siguientes pasos:

- a. Crear un contenedor.
- b. Seleccionar un *gestor de esquemas* para la inserción de los componentes.
- c. Crear los componentes adecuados.
- d. Agregarlos al contenedor.
- e. Dimensionar el contenedor(opcional):
- f. Pedir el ajuste de los componentes al contenedor.
- g. Mostrar el contenedor.

3.1. Crear un contenedor.

Hay dos clases de contenedores:

- **Frame**. Ventana de nivel superior con bordes y título, cuyos métodos principales son:
`pack()`, `setTitle()`, `getTitle()`, `setIconImage()`.
- **Panel**. No puede utilizarse de forma aislada. Necesita estar contenida en otro contenedor.

Un contenedor puede contener, a su vez, a otros contenedores o a otros componentes mediante la utilización del método `add()` de la clase *Component*.

Un *Frame*(ventana) puede contener a varios paneles y componentes. Un *Panel* puede contener a otros paneles y componentes.

3.2. Seleccionar un gestor de esquemas.

Un ***Layout Managers*** (Gestor de esquemas) es un objeto que controla cómo los *Components* (componentes) se sitúan en un *Container* (contenedor). Determinan, por tanto, como encajan los componentes dentro de los contenedores.

Cada contenedor tiene su propio gestor de esquemas por defecto. Por defecto, un *Frame* tiene un *BorderLayout* y un *Panel* tiene un *FlowLayout*.

Los gestores existentes en AWT son: *FlowLayout*, *BorderLayout*, *GridLayout*, *GridBagLayout* y *CardLayout*.

Para asignar un gestor de esquemas a un *Container* se utiliza el método:

```
setLayout(Layout Managers)
```

por ejemplo,

```
setLayout(new FlowLayout());
```

3.3. Crear componentes.

Cada componente es una clase que se debe instanciar, creando un objeto de dicha clase. Por ejemplo:

```
Button bSi = new Button("SI");  
Label l = new Label("Nombre");
```

3.4. Agregar componentes al contenedor.

Se hace a través del método `add()` de la clase *Container*. Por ejemplo:

```
Frame f = new Frame("Ejemplo de GUI");  
f.setLayout(new FlowLayout());  
Button bSi = new Button("SI");  
Button bNo = new Button("NO");  
Label l = new Label("Nombre");  
f.add(l);  
f.add(bSi);  
f.add(bNo);
```

3.5. Dimensionar el contenedor.

Es opcional. Si no se indica, el ajuste se hace para que quepan todos los componentes (excepto para la clase *Canvas*).

El método a llamar es `setSize()` de la clase *Component*. Por ejemplo:

```
f.setSize(int anchura, int altura);
```

3.6. Ajuste del contenedor.

Calcula la posición de los componentes teniendo en cuenta:

- El gestor de esquemas seleccionado.
- El número y orden de los componentes añadidos.
- La dimensión dada o calculada.

El método a llamar es `pack()` de la clase *Container*. Por ejemplo: `f.pack();`.

3.7. Mostrar el contenedor.

Para hacer visible o invisible un contenedor se utiliza el método `setVisible(boolean)` de la clase `Component`. Este método es válido para mostrar u ocultar componentes y contenedores. Por ejemplo:

```
f.setVisible(true);
```

3.8. Ejemplo de GUI en AWT.

```
import java.awt.*;
class GUI01 {
    public static void main(String args []) {
        Frame f = new Frame("Ejemplo de GUI");
        f.setLayout(new FlowLayout());
        Button bSi = new Button("SI");
        Button bNo = new Button("NO");
        Label l = new Label("Nombre");
        f.add(l);
        f.add(bSi);
        f.add(bNo);
        f.pack();
        f.setVisible(true);
    }
}
```

3.9. Convertir una aplicación AWT a Swing

Aunque en el documento anexo a este tema se puede ver mucho más detalladamente, aquí resumiremos los pasos a dar para convertir un programa hecho con AWT a Swing.

1. Cambiar el paquete de AWT (**java.awt.***) por el de Swing (**javax.swing.***), así el compilador nos irá dando errores, esto nos ayudará a localizar nuestros componentes AWT y los podremos reemplazar por sus equivalentes Swing, por último añadiremos las importaciones para las clases de AWT que realmente necesitemos (como es el caso de los gestores de esquemas).
2. Cambiar cada componente AWT por su equivalente Swing más cercano, en la mayoría de los casos sólo hay que añadirle una J, por ejemplo `Button`, lo cambiaremos por `JButton`.

```
JButton bSi = new JButton("SI");
```

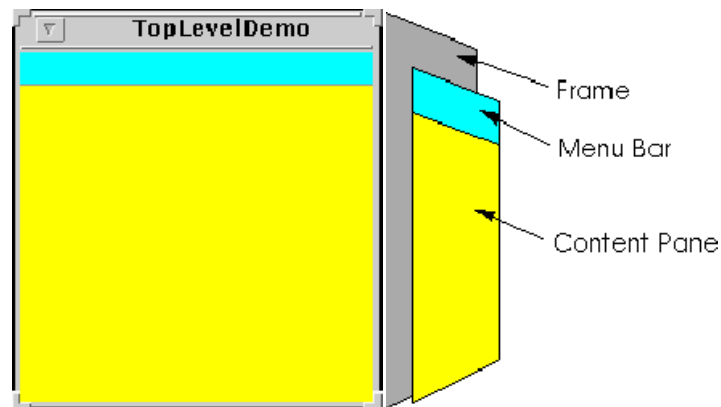
3. Cambiar las llamadas a los métodos `add` y `setLayout`

Los programas AWT añaden componentes y seleccionan el controlador de distribución directamente sobre el contenedor de alto nivel (un frame, dialog, o applet). En contraste, los programas Swing añaden componentes y seleccionan el controlador de distribución sobre el **panel de contenido** del contenedor de alto nivel. La primera fila de la siguiente tabla muestra algún código típico AWT para añadir componentes a un frame y seleccionar su controlador de distribución. La segunda y tercera filas muestran dos diferentes conversiones Swing.

Código AWT	<pre>frame.setLayout(new FlowLayout()); frame.add(button); frame.add(label); frame.add(textField);</pre>
Conversión Óbvia Swing (No hagas esto)	<pre>frame.getContentPane().setLayout(new FlowLayout()); frame.getContentPane().add(button); frame.getContentPane().add(label); frame.getContentPane().add(textField);</pre>
Conversión Eficiente Swing (Haz esto)	<pre>Container contentPane = frame.getContentPane(); contentPane.setLayout(new FlowLayout()); contentPane.add(button); contentPane.add(label); contentPane.add(textField);</pre>

Habrás notado que el código Swing de la segunda fila llama a **getContentPane** múltiples veces, lo que es ineficiente si nuestro programa usa muchos componentes. El código Swing de la tercera fila mejora el

código, obteniendo el panel de contenido una sola vez, almacenándolo en una variable, y usando la variable para añadir componentes y seleccionar el controlador de distribución.



3.10. Ejemplo de GUI en Swing.

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

class GUI01Swing {
    public static void main(String[] args)
    {
        JFrame f=new JFrame("Ejemplo de GUI");
        Container gcp = f.getContentPane();

        gcp.setLayout(new FlowLayout());

        JButton bSi = new JButton("SI");
        JButton bNo = new JButton("NO");
        JLabel l = new JLabel("Nombre");

        gcp.add(l);
        gcp.add(bSi);
        gcp.add(bNo);
        f.setSize(200,400);
        f.pack();
        f.setVisible(true);

        // Añado el control al cierre de la ventana
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Aunque los dos ejemplos anteriores, tanto el de AWT como el de Swing, son correctos, es habitual que una aplicación en Swing (también en AWT) se cree definiendo una clase, subclase de JFrame. En el constructor de la nueva clase se definirá todo el interfaz gráfico, y en el método *main*, se creará un objeto de la clase que acabamos de desarrollar. De esta forma el contenedor de nivel superior lo tenemos ya de forma implícita al haber heredado de JFrame, y nos ahorramos de tenerlo que crear, si nos queremos referir a él de forma explícita utilizaremos *this*. El ejemplo anterior, trabajando de esta forma quedaría:

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

@SuppressWarnings("serial")
class GUI01Swing2 extends JFrame
{
    public GUI01Swing2()
    {
        super("Ejemplo de GUI");

        Container contentPane = this.getContentPane();
```

```

contentPane.setLayout(new FlowLayout());

JButton bSi = new JButton("Si");
JButton bNo = new JButton("No");
JLabel l = new JLabel("Nombre");

contentPane.add(l);
contentPane.add(bSi);
contentPane.add(bNo);

this.setSize(200,100);
this.pack();
this.setVisible(true);
// Cierre de la ventana (si no se pone, también funciona,
// pero el proceso sigue estando activo, es como si le
// quitase solo la visibilidad)
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

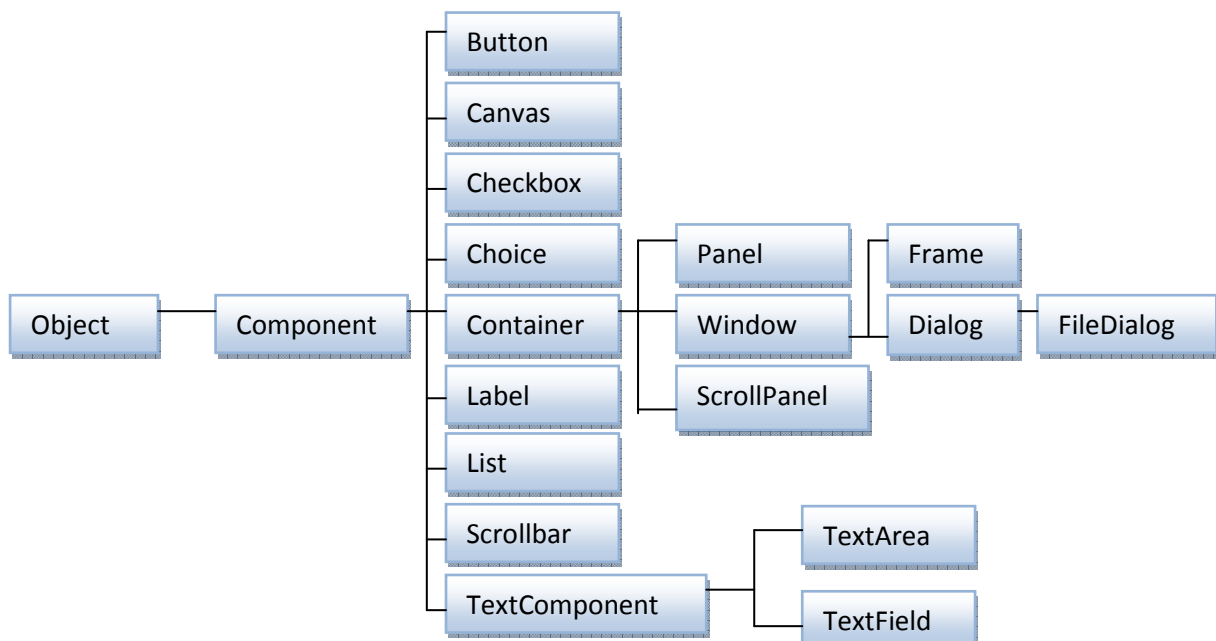
public static void main(String[] args)
{
    new GUI01Swing2();
}

```

4. UN PRIMER VISTAZO A COMPONENTES Y CONTENEDORES.

4.1. Componentes soportados por el AWT de Java

Como todas las clases de **Java**, los componentes utilizados en el AWT pertenecen a una determinada jerarquía de clases, que es muy importante conocer. Esta jerarquía de clases se muestra en la siguiente figura. Todos los componentes descienden de la clase **Component**, de la que pueden ya heredar algunos métodos interesantes. El **package** al que pertenecen estas clases se llama **java.awt**.



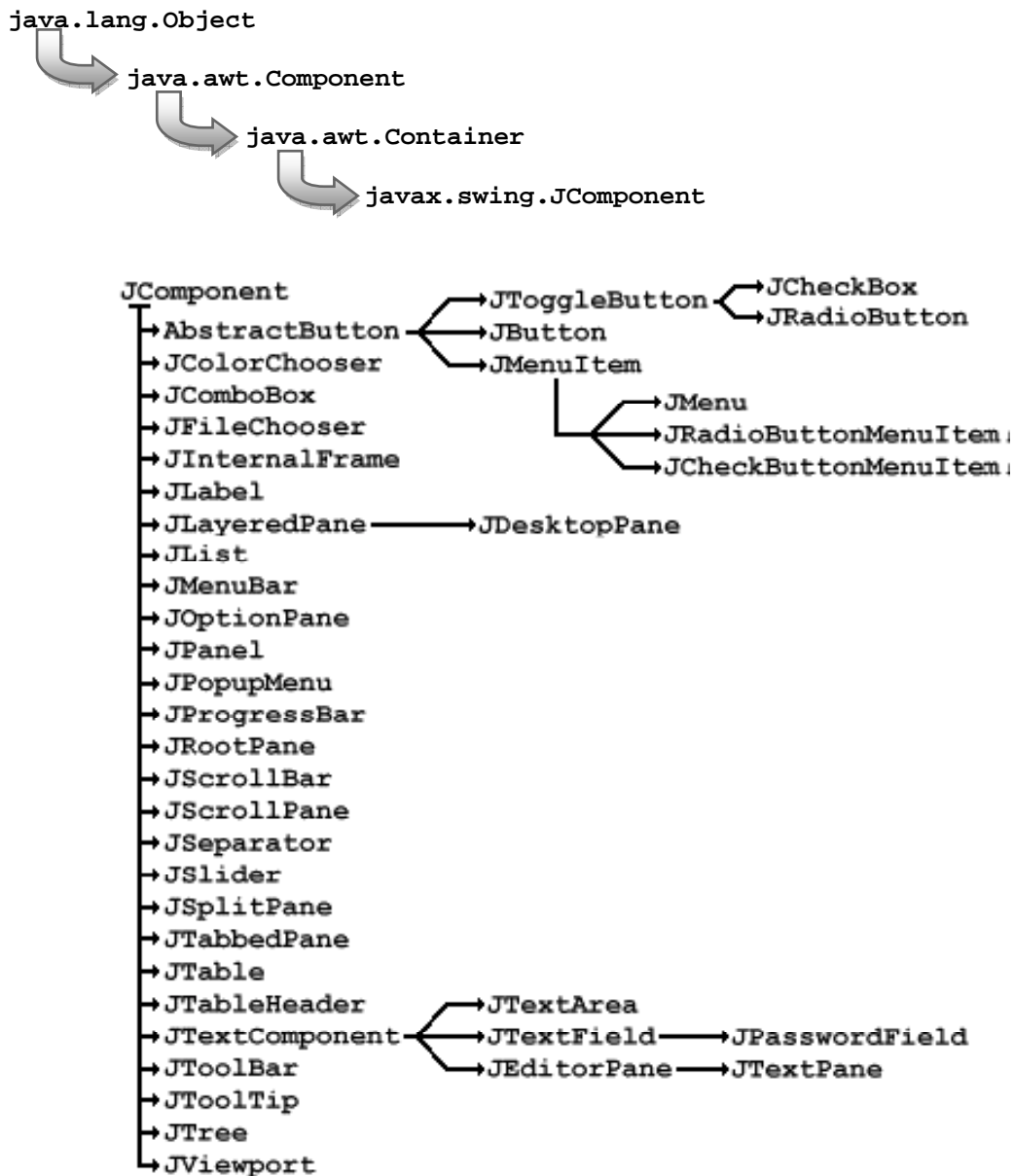
A continuación se resumen algunas características importantes de los componentes mostrados en la figura anterior:

1. Todos los **Components** (excepto **Window** y los que derivan de ella) deben ser añadidos a un **Container**. También un **Container** puede ser añadido a otro **Container**.
2. Para añadir un **Component** a un **Container** se utiliza el método `add()` de la clase **Container**:

```
containerName.add(componentName);
```
3. Los **Containers** de máximo nivel son las **Windows** (**Frames** y **Dialogs**). Los **Panels** y **ScrollPanels** deben estar siempre dentro de otro **Container**.
4. Un **Component** sólo puede estar en un **Container**. Si está en un **Container** y se añade a otro, deja de estar en el primero.
5. La clase **Component** tiene una serie de funcionalidades básicas comunes (variables y métodos) que son heredadas por todas sus **sub-classes**.

4.2. Componentes SWING

Como hemos visto los componentes de AWT heredan de la clase **Component** y esta a su vez de la clase **Object**. En el caso de Swing, la relación de herencia de sus **componentes** es la siguiente:



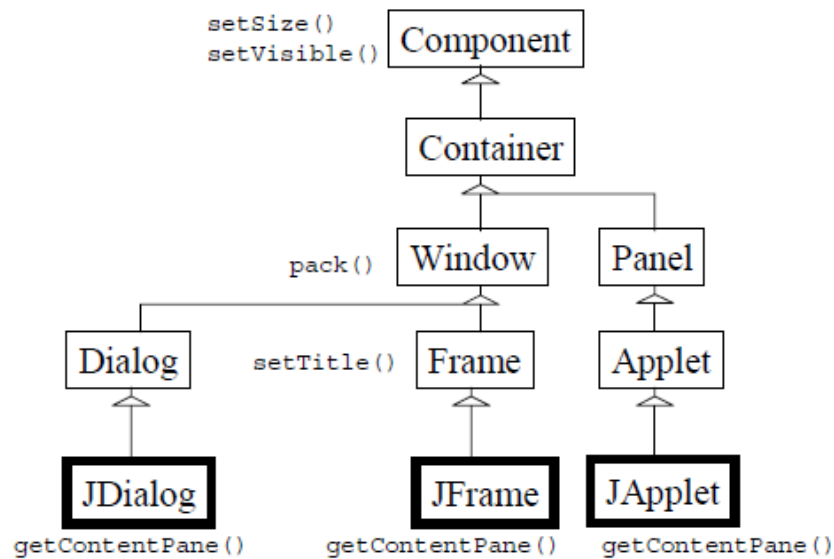
Significado de los componentes anteriores:

Componente	Descripción
JButton	Botón.
JCheckBox	Botón de comprobación.
JCheckBoxMenuItem	Botón de comprobación para usar en menús
JColorChooser	Selector de colores.
JComboBox	Entrada de texto con lista de valores.
JComponent	Raíz de la jerarquía de componentes Swing.
JEditorPane	Editor de texto. Normalmente HTML o RTF.
JFileChooser	Selector de ficheros.
JLabel	Etiqueta.
JList	Lista.
JMenu	Menú dentro de un JMenuBar o dentro de otro menú.
JMenuBar	Barra de Menús.
JMenuItem	Elemento seleccionable en un menú.
JOptionPane	Ventanas de dialogo.
JPasswordField	Entrada de passwords.
JPopupMenu	Ventana con un menú.
JProgressBar	Barra de progreso.
JRadioButton	Botón excluyente.
JRadioButtonMenuItem	Botón excluyente para usar en menús
JScrollBar	Barra de desplazamiento.
JSeparator	Líneas de separación.
JSlider	Deslizador.
JTable	Tabla.
JTextArea	Edición de múltiples líneas de texto plano.
JTextComponent	Raíz de los editores de texto.
TextField	Edición de una línea de texto plano.
JTextPane	Subclase de JEditorPane para hacer procesadores de texto.
JToggleButton	Padre de JCheckBox y JRadioButton.
JToolBar	Barra de herramientas o acciones.
JToolTip	Ventana informativa.
JTree	Árboles.

JMenuBar y JPopupMenu son en realidad contenedores.

En el caso de los **contenedores** Swing:

- JFrame, JDialog, JWindow y JApplet son **contenedores de alto nivel o superior** que extienden sus versiones AWT, y heredan de la clase `java.awt.Container`.



Cuando se crean estos contenedores, se crea además un contenedor `JRootPane` (esto no ocurría en AWT) dentro de ellos.

El método `getContentPane()` de estos contenedores da acceso al contenedor donde deben añadirse los hijos. Por defecto es un `JPanel`.

```

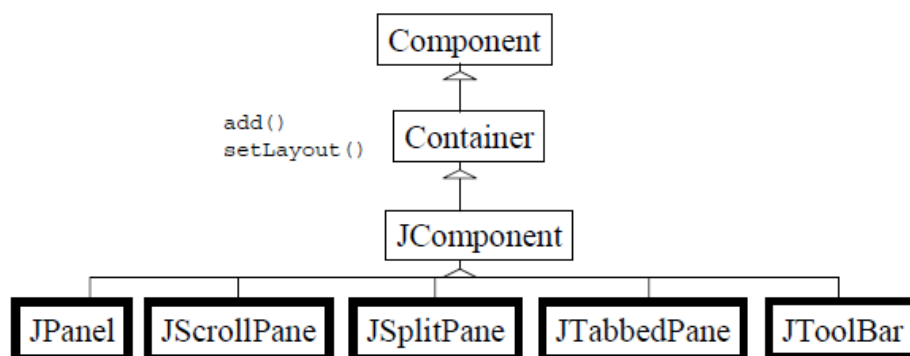
JFrame f = new JFrame("Ejemplo");
JLabel l = new JLabel("Hola");

f.getContentPane().add(l);

```

Puede usarse `setContentPane()` para poner el contenedor que se desee.

- Los demás contenedores extienden `JComponent`, son conocidos como **contenedores de nivel intermedio**, y al no ser de alto nivel deben estar contenidos en otro de nivel superior.



La siguiente tabla muestra una breve descripción de los contenedores existentes en Swing:

Contenedor	Descripción
Box	Posiciona hijos usando BoxLayout.
JApplet	Applets.
JDesktopPane	Desktop que contiene JInternalFrame(s).
JDialog	Presentación de diálogos.
JFrame	Ventana.
JInternalFrame	Ventana interna. Suele usarse dentro de un JDesktopPane.
JLayeredPane	Contenedores apilados.
JPanel	Agrupar hijos.
JRootPane	Usado por JApplet, JDialog, JFrame, JInternalFrame y JWindow. Proporciona muchas características.
JScrollPane	Añade barras de desplazamiento a su hijo.
JSplitPane	Muestra dos hijos pudiendo ajustar sus tamaños relativos.
JTabbedPane	Solapas o pestañas para mostrar diferentes hijos.
JViewport	Muestra una parte de sus hijos. Típicamente usado por JScrollPane.
JWindow	Ventana sin decoración.

En el caso de Box automáticamente utiliza un **BoxLayout** para distribuir sus componentes. La ventaja de **Box** es que es de peso superligero, ya que descende directamente de la clase **Container**. Su desventaja es que no es un verdadero componente Swing -- no hereda el API que soporta características como los bordes de la caja, ni la selección sencilla de los tamaños máximo, mínimo y preferido. Por esta razón, es preferible utilizar **JPanel** con **BoxLayout**, en vez **Box**.

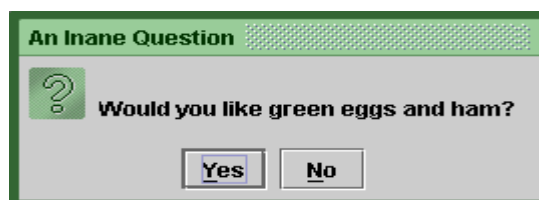
En las imágenes que se muestran a continuación se pueden ver algunos de los contenedores y componentes básicos de Swing. Todos ellos se irán estudiando con más detenimiento en los próximos apartados.

CONTENEDORES

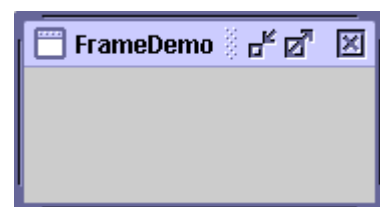
Contenedores de alto nivel



Applet

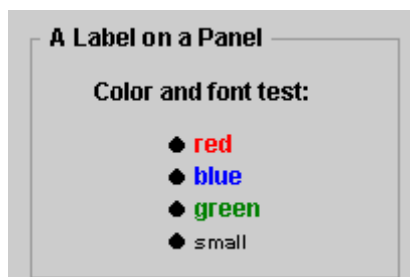


Dialog

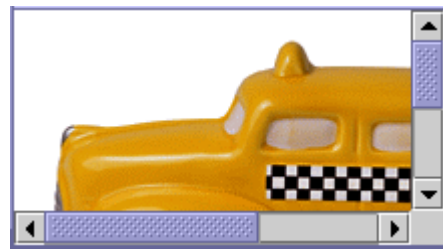


Frame

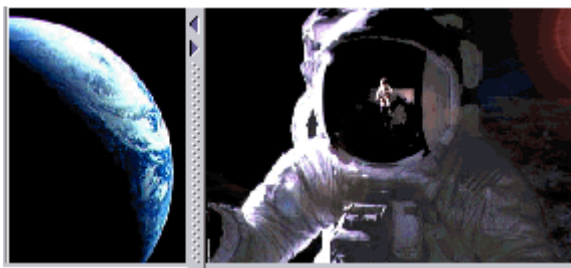
Contenedores de propósito general



Panel



Scroll pane



Split pane

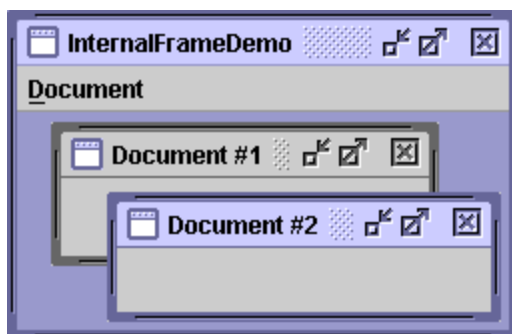


Tabbed pane

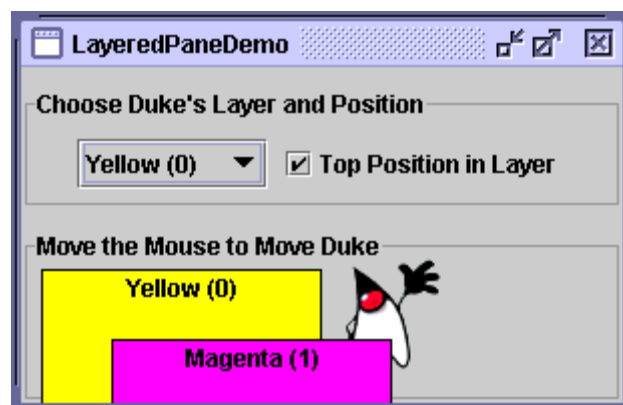


Tool bar

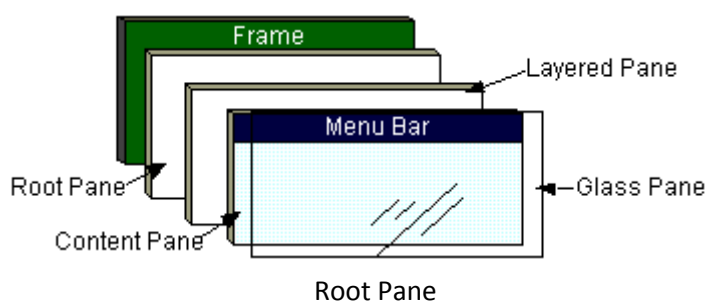
Contenedores de propósito especial



Internal frame



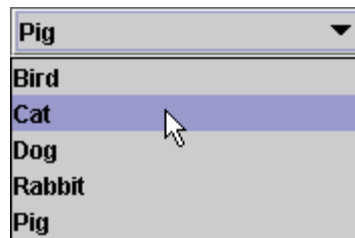
L
Layered pane



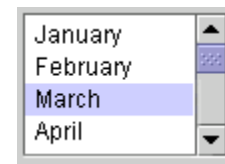
COMPONENTES BÁSICOS



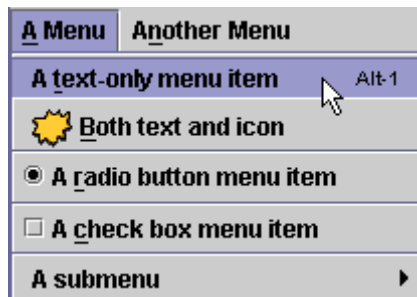
Buttons



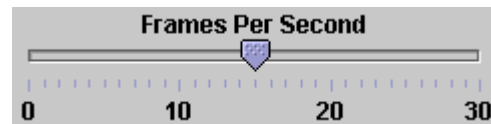
Combo box



List



Menu



Slider



Spinner



Text field or Formatted text field

Información no editable



Label

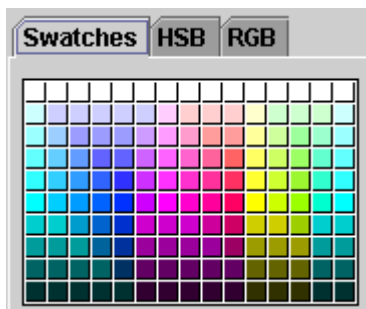


Progress bar

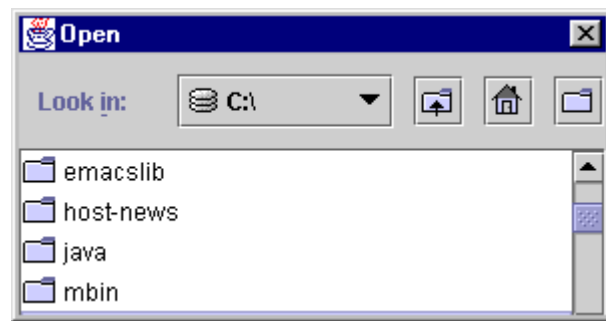


Tool tip

Información destacada



Color chooser



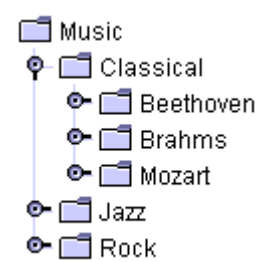
File chooser

First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

Table



Text



Tree

5. COMPONENTES Y CONTENEDORES

Antes de empezar a ver los componentes más importantes de Swing, veremos la **Interfaz SwingConstants**. Esta interfaz define una serie de constantes para la alineación del texto e imágenes en componentes:

- **static int BOTTOM**. Alineación abajo.
- **static int CENTER**. Posición centro
- **static int EAST**. Posición derecha.
- **static int HORIZONTAL**. Orientación horizontal.
- **static int LEADING**. Identifica la esquina leading del texto para usar con lenguajes izq-der y der-izq.
- **static int LEFT**. Alineación izquierda.
- **static int NEXT**. Siguiendo dirección en secuencia.
- **static int NORTH**. Posición arriba.
- **static int NORTH_EAST**. Posición arriba-derecha.
- **static int NORTH_WEST**. Posición arriba-izquierda.
- **static int PREVIOUS**. Previa dirección en la secuencia.
- **static int RIGHT**. Alineación derecha.
- **static int SOUTH**. Posición abajo.
- **static int SOUTH_EAST**. Posición abajo-derecha.
- **static int SOUTH_WEST**. Posición abajo-izquierda.
- **static int TOP**. Alineación arriba.
- **static int TRAILING**. Identifica la esquina trailing del texto usada en lenguajes izq-der y der-izq.
- **static int VERTICAL**. Orientación vertical.
- **static int WEST**. Posición izquierda.

5.1. Clase Component.

La clase **Component** es una clase **abstract** de la que derivan todas las clases del AWT, y a través de la clase `java.awt.Container`, también los componentes de Swing, por medio de la clase `javax.swing.JComponent`, según los diagramas mostrados previamente.

Los métodos de esta clase son importantes porque son heredados por todos los componentes de AWT y Swing. La siguiente tabla muestra algunos de los métodos más utilizados de la clase **Component**. En las declaraciones de los métodos de dicha clase aparecen las clases **Point**, **Dimension** y **Rectangle**.

- La clase **java.awt.Point** tiene dos variables miembro **int** llamadas **x** e **y**.
- La clase **java.awt.Dimension** tiene dos variables miembro **int**: **height** y **width**.
- La clase **java.awt.Rectangle** tiene cuatro variables **int**: **height**, **width**, **x** e **y**.
- Las tres son sub-clases de **Object**.

Métodos de Component	Función que realizan
<code>boolean isVisible(), void setVisible(boolean)</code>	Permiten chequear o establecer la visibilidad de un componente
<code>boolean isShowing()</code>	Permiten saber si un componente se está viendo. Para ello tanto el componente debe ser visible, y su <i>container</i> debe estar mostrándose
<code>boolean isEnabled(), void setEnabled(boolean)</code>	Permiten saber si un componente está activado y activarlo o desactivarlo

Point getLocation(), Point getLocationOnScreen()	Permiten obtener la posición de la esquina superior izquierda de un componente
void setLocation(Point), void setLocation(int x, int y)	Desplazan un componente a la posición especificada respecto al container o respecto a la pantalla
Dimension getSize(), void setSize(int w, int h), void setSize(Dimension d)	Permiten obtener o establecer el tamaño de un componente
Rectangle getBounds(), void setBounds(Rectangle), void setBounds(int x, int y, int width, int height)	Obtienen o establecen la posición y el tamaño de un componente
invalidate(), validate(), doLayout	<i>invalidate()</i> marca un componente y sus contenedores para indicar que se necesita volver a aplicar el Layout Manager. <i>validate()</i> se asegura que el Layout Manager está bien aplicado. <i>doLayout()</i> hace que se aplique el Layout Manager
setBackground(Color), setForeground(Color)	Métodos para establecer los colores por defecto
paint(Graphics), repaint() y update(Graphics)	Métodos gráficos para dibujar en la pantalla

Además de los métodos mostrados en esta tabla, la clase **Component** tiene un gran número de métodos básicos cuya funcionalidad puede estudiarse mediante la documentación on-line de **Java**. Entre otras funciones, permite controlar los **colores**, las **fonts** y los **cursores**.

5.2. Clase Container.

La clase **Container** es también una clase muy general. De ordinario, nunca se crea un objeto de esta clase, pero los métodos de esta clase son heredados por las clases **Frame** y **Panel**, que sí se utilizan con mucha frecuencia para crear objetos. Además de ella hereda la clase **JComponent**.

Los **containers** mantienen una **lista de los objetos** que se les han ido añadiendo. Cuando se añade un nuevo objeto se incorpora al final de la lista, salvo que se especifique una posición determinada. En esta clase tiene mucha importancia todo lo que tiene que ver con los **Layout Managers**, que se explicarán más adelante. La siguiente tabla muestra algunos métodos de la clase **Container**.

Métodos de Container	Función que realizan
add()	Añade un componente al container
doLayout()	Ejecuta el algoritmo de ordenación del layout manager
getComponent(int)	Obtiene el n-ésimo componente en el container
getComponentAt(int, int), getComponentAt(Point)	Obtiene el componente que contiene un determinado punto
getComponentCount()	Obtiene el número de componentes en el container
getComponents()	Obtiene los componentes en este container.
remove(Component), remove(int), removeAll()	Elimina el componente especificado.
setLayout(LayoutManager)	Determina el layout manager para este container

5.3. Clase JComponent

Como hemos visto anteriormente, la clase JComponent representa la clase base de toda la jerarquía de componentes a excepción de los contenedores de alto nivel (JFrame, JDialog y JApplet).

JComponent proporciona el siguiente comportamiento común:

- Es la clase base para los componentes estándar y de usuario en la arquitectura Swing.
- Permite establecer un aspecto para los componentes, lo que se llama “*pluggable look and feel (L&F)*”. El programador puede establecer un aspecto determinado tanto en tiempo de diseño como en tiempo de ejecución. El aspecto para cada componente es proporcionado por un objeto UI (objeto que descende de ComponentUI).
- Manejo de teclas más comprensivo.
- Soporte para las ventanas deslizantes (*Tool Tips*).
- Soporte para propiedades específicas de cliente, con los métodos `putClientProperty(java.lang.Object, java.lang.Object)` and `getClientProperty(java.lang.Object)`.
- Soporte para el pintado que incluye el doble buffering y manejo de bordes.

Algunos de los métodos de esta clase son los siguientes:

<i>Métodos de JComponent</i>	<i>Función que realizan</i>
<code>ActionListener getActionForKeyStroke (KeyStroke aKeyStroke)</code>	Devuelve la acción asignada a una combinación de teclas.
<code>float getAlignmentX(), float getAlignmentY()</code>	Devuelve la alineación vertical-horizontal del componente.
<code>boolean getAutoscrolls(), void setAutoscrolls(boolean autoscrolls)</code>	Devuelve o establece la propiedad autoscroll.
<code>Border getBorder(), void setBorder(Border border)</code>	Devuelve o establece el borde del component.
<code>Rectangle getBounds(Rectangle rv), int getHeight(), int getWidth(), Dimension getSize(Dimension rv)</code>	Devuelve los límites del componente.
<code>Object getClientProperty(Object key)</code>	Devuelve la propiedad para la clave indicada.
<code>int getConditionForKeyStroke (KeyStroke aKeyStroke)</code>	Devuelve la condición que determina si una acción registrada ocurre en respuesta a un KeyStroke determinado.
<code>Graphics getGraphics()</code>	Devuelve el entorno gráfico del componente para poder dibujar en él.
<code>Insets getInsets()</code>	Devuelve la configuración de los bordes del componente si está definido o los del componente si no tiene.
<code>Point getLocation(Point rv), int getX(), int getY()</code>	Devuelve la posición en la que se sitúa el componente
<code>Dimension getMaximumSize(), Dimension getMinimumSize(), void setMaximumSize (Dimension maximumSize), void setMinimumSize (Dimension minimumSize)</code>	Devuelve el tamaño máximo y mínimo posible para el componente.
<code>Dimension getPreferredSize(), void setPreferredSize (Dimension preferredSize)</code>	Devuelve o establece el tamaño aconsejado para el componente.

<code>JrootPane getRootPane()</code>	Devuelve el JrootPane en el que esta contenido el componente.
<code>Point getToolTipLocation(MouseEvent event)</code>	Devuelve la posición de la etiqueta flotante en coordenadas del componente.
<code>String getToolTipText(), void setToolTipText(String text)</code>	Devuelve o establece el texto de las etiquetas.
<code>Container getTopLevelAncestor()</code>	Devuelve el contenedor ancestro (frame, dialog, applet).
<code>void grabFocus(), void requestFocus(), boolean requestFocusInWindow()</code>	Solicita el focus para el componente.
<code>boolean isDoubleBuffered(), void setDoubleBuffered(boolean aFlag)</code>	Devuelve o establece si se aplica el doble bufferin
<code>boolean isMaximumSizeSet(), boolean isMinimumSizeSet(), boolean isPreferredSizeSet()</code>	Indica si se ha asignado un tamaño máximo, mínimo y aconsejado.
<code>boolean isOpaque(), void setOpaque(boolean isOpaque)</code>	Devuelve o establece si el componente es transparente.
<code>void paint(Graphics g)</code>	Llamado cuando se necesita pintar el componente.
<code>void paintImmediately (int x, int y, int w, int h), void paintImmediately(Rectangle r)</code>	Hace que se redibuje una zona.
<code>void print(Graphics g), void printAll(Graphics g).</code>	Pinta el componente.
<code>void putClientProperty (Object key, Object value)</code>	Establece una propiedad para el componente.
<code>void repaint(long tm, int x, int y, int width, int height), void repaint(Rectangle r)</code>	Hace que se repinte una zona.
<code>void reshape(int x, int y, int w, int h)</code>	Mueve y redimensiona el componente.
<code>void scrollRectToVisible(Rectangle aRect)</code>	Hace que se visualice una zona del componente.
<code>void setBackground(Color bg)</code>	Establece el color de fondo.
<code>void setEnabled(boolean enabled)</code>	Habilita el componente.
<code>void setFont(Font font)</code>	Establece la fuente para el componente.
<code>void setVisible(boolean aFlag)</code>	Hace visible el componente.

5.4.- Contenedores de alto nivel.

Entendemos por contenedores de alto nivel a **JFrame**, **JDialog** y **JApplet**. Al derivar estos de las clases correspondientes en AWT: **Frame**, **Dialog** y **Applet**, tienen todos sus métodos añadiendo otros específicos. También es un contenedor de alto nivel **Window**, pero no se suelen crear objetos de esta clase.

5.4.1. Window.

Los objetos de la clase **Window** son ventanas de máximo nivel, pero *sin bordes* y *sin barra de menús*. En realidad son más interesantes las clases que derivan de ella: **Frame** y **Dialog**. Los métodos más útiles, por ser heredados por las clases **Frame** y **Dialog**, se muestran en la siguiente tabla.

<i>Métodos de Window</i>	<i>Función que realizan</i>
<code>toFront()</code> y <code>toBack()</code>	Para desplazar la ventana hacia adelante y hacia atrás en la pantalla
<code>show()</code>	Muestra la ventana y la trae a primer plano
<code>pack()</code>	Hace que los componentes se reajusten al tamaño preferido

5.4.2. JFrame

Es una ventana *con un borde* y que puede tener una *barra de menús*. Si una ventana depende de otra ventana, es mejor utilizar una **Window** (ventana sin borde ni barra de menús) que un **JFrame**. La siguiente tabla muestra algunos métodos más utilizados de la clase **JFrame**, la mayoría heredados de **Frame**.

<i>Métodos de JFrame</i>	<i>Función que realiza</i>
<code>JFrame()</code> , <code>JFrame(String title)</code> , <code>JFrame(GraphicsConfiguration gc)</code> <code>JFrame(String title, GraphicsConfiguration gc)</code>	Constructores de JFrame
<code>String getTitle()</code> , <code>setTitle(String)</code>	Obtienen o determinan el título de la ventana
<code>MenuBar getMenuBar()</code> , <code>setMenuBar(MenuBar)</code> , <code>remove(MenuComponent)</code>	Permite obtener, establecer o eliminar la barra de menús
<code>Image getIconImage()</code> , <code>setIconImage(Image)</code>	Obtienen o determinan el icono que aparecerá en la barra de títulos
<code>setResizable(boolean)</code> , <code>boolean isResizable()</code>	Determinan o chequean si se puede cambiar el tamaño
<code>dispose()</code>	Método que libera los recursos utilizados en una ventana. Todos los componentes de la ventana son destruidos.
<code>Container getContentPane()</code>	Los componentes que contiene un JFrame se insertan, no directamente en el, sino en un <code>ContentPane</code> , este método nos lo devuelve.

Además de los métodos citados, se utilizan mucho los métodos *show()*, *pack()*, *toFront()* y *toBack()*, heredados de la super-clase **Window**.

5.4.3. JDialog

Un **JDialog** es una ventana que normalmente depende de otra ventana (de una **JFrame**). Es un elemento de visualización al igual que un **Frame**. Si una **JFrame** se cierra, se cierran también los **Dialog** que dependen de ella; si se iconifica, sus **Dialog** desaparecen; si se restablece, sus **Dialog** aparecen de nuevo. Este comportamiento se obtiene de forma automática.

Normalmente se suele crear y no visualizar hasta que sea necesario, para visualizarlo `setVisible(true)`, para ocultarla `setVisible(false)` y para eliminarla `dispose()`.

Las **Applets** estándar no soportan **Dialogs** porque no son **Frames** de **Java**. Las **Applets** que abren **Frames** sí pueden soportar **Dialogs**.

Un **Dialog modal** requiere la atención inmediata del usuario: no se puede hacer ninguna otra cosa hasta no haber cerrado el **Dialog**. Por defecto, los **Dialogs** son **no modales**. La siguiente tabla muestra los métodos más importantes de la clase **Dialog**. Se pueden utilizar también los métodos heredados de sus super-clases.

Métodos de Dialog	Función que realiza
<code>JDialog(Frame fr),</code> <code>JDialog(Frame fr, boolean mod)</code> <code>JDialog(Frame fr, String title),</code> <code>JDialog(Frame fr, String title, boolean mod)</code> <code>...</code>	Constructores
<code>String getTitle(), setTitle(String)</code>	Permite obtener o determinar el título
<code>boolean isModal(), setModal(boolean)</code>	Pregunta o determina si el Dialog es modal o no
<code>boolean isResizable(), setResizable(boolean)</code>	Pregunta o determina si se puede cambiar el tamaño
<code>show()</code>	Muestra y trae a primer plano el Dialog
<code>Container getContentPane().</code>	
<code>JMenuBar getJMenuBar()</code> <code>void setJMenuBar(JMenuBar menubar)</code>	

Ejemplo: `JFrame f = new JFrame();`
`JDialog d = new JDialog(f, "Ventana modal", true);`

5.4.4. JApplet

Los applets son programas que se ejecutan en el navegador. Son aplicaciones accesibles en un servidor web, que se transportan por la red. La clase **JApplet** permite extenderse para crear applets.

Ciclo de vida de un applet: Un applet se carga, se instancia y ejecuta dentro de un navegador web.

Métodos básicos:

1. **init()** : Lo invoca el navegador cuando se carga el applet. // se usa como un constructor //
2. **start()** : Se llama después del `init()` y cada vez que el applet vuelve al área visible del navegador.
3. **stop()** : Se llama cuando se sale del área visible del navegador, y también antes del `destroy()`. Detiene la ejecución pero no lo saca de memoria.
4. **destroy()** : Se llama cuando se descarga el applet del navegador. Lo saca de memoria.

Todos los applets que hacen algo después de la inicialización (excepto la respuesta a una acción al usuario) deben sobrescribir el método `start()`.

5.5. Contenedores básicos o de nivel intermedio (JPanel, JScrollPane, JTabbedPane, JSplitPane, JToolBar)

5.5.1. Clase JPanel.

Un *panel* es un **Container** de propósito general. Se puede utilizar tal cual para contener otras componentes, y también crear una sub-clase para alguna finalidad más específica. Por defecto, el **Layout Manager** (gestor de esquemas) de **JPanel** es **FlowLayout**. Los **Applets** son sub-clases de **Panel**. La siguiente tabla muestra los métodos más importantes que se utilizan con la clase **Panel**, que son algunos métodos heredados de **Component** y **Container**, pues la clase **Panel** no tiene métodos propios.

Métodos de Panel	Función que realiza
JPanel(), JPanel(LayoutManager miLM)	Constructores de Panel
add(Component), add(Component, int)	Añade componentes al panel
setLayout(), getLayout()	Establece o permite obtener el layout manager utilizado
validate(), doLayout()	Para reorganizar los componentes después de algún cambio. Es mejor utilizar validate()
remove(int), remove(Component), removeAll()	Para eliminar componentes
GetMaximumSize(), gettMinimumSize(), getPreferredSize()	Permite obtener los tamaños máximo, mínimo y preferido
Insets getInsets()	

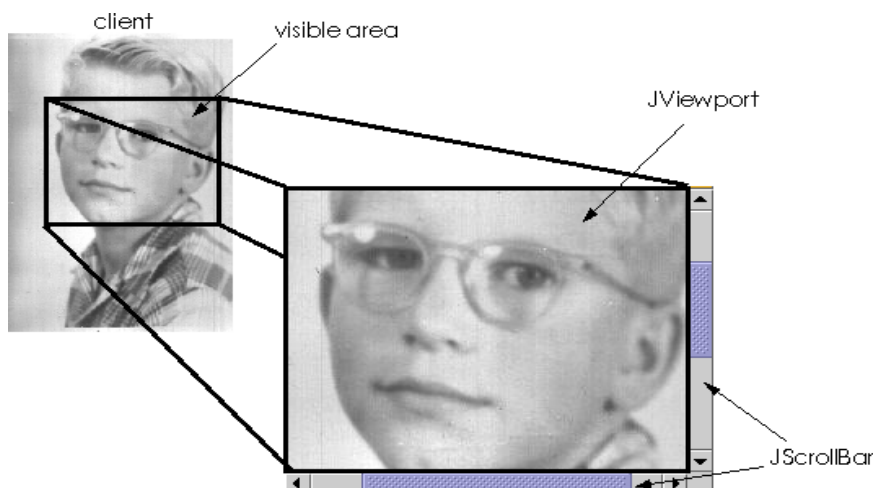
Un **JPanel** puede contener otros **JPanel**. Esto es una gran ventaja respecto a los demás tipos de containers, que son containers de máximo nivel y no pueden introducirse en otros containers.

Insets es una clase que deriva de **Object**. Sus variables son **top**, **left**, **bottom**, **right**. Representa el espacio que se deja libre en los bordes de un **Container**. Se establece mediante la llamada al adecuado constructor del **Layout Manager**.

Ejemplo: `JPanel p = new JPanel();`

5.5.2. JScrollPane

Permite hacer scroll a un componente (u otro contenedor intermedio), por ejemplo a una imagen, un área de texto, una lista, ... Dicho en otras palabras, un **JScrollPane** es un panel que aporta las características de desplazamiento a cualquier componente que así lo requiera. El **JScrollPane** define un objeto de tipo **JViewport** que es el encargado de traducir las posiciones de la barra de desplazamiento con respecto del área de cliente.



A continuación se muestra un pequeño ejemplo de cómo poner un área de texto dentro de un JScrollPane:

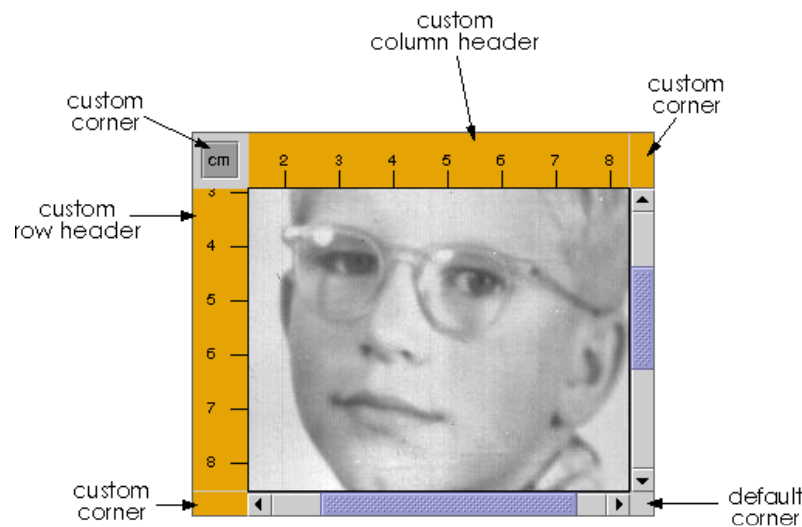
```
JTextArea area = new JTextArea("Area de texto",5,20);
JScrollPane spa = new JScrollPane(area);
// Lo que añadiremos al contenedor donde vaya el área de texto será el
// JScrollPane que la contiene no el área de texto
```

Otro ejemplo sencillo, en este caso para una imagen es el siguiente:

```
import java.awt.*;
import javax.swing.*;
class GUIP {
    public static void main(String [] args) {
        JFrame frame = new JFrame("Un ejemplo");
        ImageIcon ii = new ImageIcon("carapin.gif");
        JLabel label = new JLabel(ii);
        JScrollPane scrollPane = new JScrollPane(label);
        frame.getContentPane().add(scrollPane);
        frame.pack();
        frame.setVisible(true);
    }
}
```



Un JScrollPane permite indicar una cabecera de columna o fila y diferentes componentes para las esquinas. Así para crear el siguiente ejemplo:



```
private Rule columnView;
private Rule rowView;
...
//Where the GUI is initialized:
ImageIcon david = createImageIcon("images/youngdad.jpeg");
...
//Create the row and column headers.
columnView = new Rule(Rule.HORIZONTAL, true);
rowView = new Rule(Rule.VERTICAL, true);

if (david != null) {
    columnView.setPreferredWidth(david.getIconWidth());
    rowView.setPreferredHeight(david.getIconHeight());
}
JLabel jLabel3=new JLabel();
jLabel3.setIcon(david);
JScrollPane.getViewport().add(jLabel3, null);
//o tambien jScrollPane=new JScrollPane(jLabel3);
```

```

pictureScrollPane.setColumnHeaderView(columnView);
pictureScrollPane.setRowHeaderView(rowView);

//Create the corners.
JPanel buttonCorner = new JPanel(); //use FlowLayout
isMetric = new JToggleButton("cm", true);
isMetric.setFont(new Font("SansSerif", Font.PLAIN, 11));
isMetric.setMargin(new Insets(2,2,2,2));
isMetric.addItemListener(this);
buttonCorner.add(isMetric);
...
//Set the corners.
pictureScrollPane.setCorner(JScrollPane.UPPER_LEFT_CORNER, buttonCorner);
pictureScrollPane.setCorner(JScrollPane.LOWER_LEFT_CORNER, new Corner());
pictureScrollPane.setCorner(JScrollPane.UPPER_RIGHT_CORNER, new Corner());

```

Define los siguientes métodos:

<i>Métodos de JScrollPane</i>	<i>Función que realiza</i>
JScrollPane(), JScrollPane(Component view)	Constructores.
JViewport getColumnHeader(), void setColumnHeaderView(Component view).	Devuelve o establece el componente para la cabecera de columna.
JViewport getRowHeader(), void setRowHeaderView(Component view).	Devuelve o establece el componente para la cabecera de fila.
Component getCorner(String key), void setCorner(String key, Component corner).	Devuelve o establece las esquinas.
JScrollBar getHorizontalScrollBar(), JScrollBar getVerticalScrollBar().	Devuelve la barra de desplazamiento horizontal/vertical.
int getHorizontalScrollBarPolicy(), int getVerticalScrollBarPolicy(), void setHorizontalScrollBarPolicy(int policy), void setVerticalScrollBarPolicy(int policy).	Devuelve o establece como deben aparecer las barras de desplazamiento.
JViewport getViewPort(), void setViewport(JViewport viewport).	Devuelve o establece el ViewPort.
void setViewportView(Component view).	Crea un ViewPort para el componente.

5.5.3. JTabbedPane

Con esta clase, podemos tener varios componentes (normalmente objetos **JPanel**) compartiendo el mismo espacio. El usuario puede elegir qué componente ver seleccionando la pestaña del componente deseado (TAB). Cada TAB tiene: color de fondo, de primer plano, icono, icono deshabilitado, título, tooltip.

Para crear un JTabbedPane, simplemente se ejemplariza un **JTabbedPane**, se crean los componentes que deseemos mostrar, y luego los añadimos al TabbedPane utilizando el método **addTab**.

Define los siguientes métodos:

<i>Métodos de JTabbedPane</i>	<i>Función que realiza</i>
<code>JTabbedPane()</code> , <code>JTabbedPane(int tabPlacement)</code>	Constructores.
<code>Component add(Component component)</code> <code>void add(Component component, Object constraints)</code> <code>Component add(String title, Component component)</code>	Añade un nuevo componente.
<code>Component component, int index)</code> , <code>void add(Component component, Object constraints, int index).</code>	Añade un componente en un TAB
<code>void addTab(String title, Component component)</code> , <code>void addTab(String title, Icon icon, Component component)</code> , <code>void addTab(String title, Icon icon, Component component, String tip).</code>	Añade un nuevo TAB.
<code>Component getComponentAt(int index)</code> <code>Color getBackgroundAt(int index)</code> <code>Color getForegroundAt(int index)</code> <code>Icon getIconAt(int index)</code> <code>String getTitleAt(int index)</code>	Devuelve los valores para el TAB.
<code>Component getSelectedComponent()</code> <code>int getSelectedIndex()</code> <code>int getMnemonicAt(int tabIndex)</code> <code>void setSelectedComponent(Component c)</code> <code>void setSelectedIndex(int index)</code>	Devuelve o establece el TAB seleccionado.
<code>int getTabCount()</code>	Numero de tabs.
<code>int getTabPlacement()</code> , <code>void setTabPlacement(int tabPlacement)</code>	Devuelve o establece donde se colocaran las etiquetas.
<code>int indexOfTab(String title)</code> <code>int indexOfComponent(Component component)</code>	Busca un tab determinado.
<code>void insertTab(String title, Icon icon, Component component, String tip, int index)</code>	Añade un TAB en una posición.
<code>void remove(Component component)</code> <code>void remove(int index)</code> , <code>void removeAll()</code> <code>void removeTabAt(int index)</code>	Borra TABS
<code>void setBackgroundAt(int index, Color background)</code> <code>void setComponentAt(int index, Component component)</code> <code>void setForegroundAt(int index, Color foreground)</code> <code>void setIconAt(int index, Icon icon)</code> <code>void setTitleAt(int index, String title)</code> <code>void setToolTipTextAt(int index, String toolTipText)</code>	Establece las propiedades para un tab.
<code>void setEnabledAt(int index, boolean enabled).</code>	Habilita un tab determinado.

A continuación se muestra un ejemplo:

```
import javax.swing.JTabbedPane;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JFrame;

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

public class TabbedPaneDemo extends JPanel {
    public TabbedPaneDemo() {
        ImageIcon icon = new ImageIcon("images/middle.gif");
        JTabbedPane tabbedPane = new JTabbedPane();

        Component panel1 = makeTextPanel("Blah");
        tabbedPane.addTab("One", icon, panel1, "Does nothing");
        tabbedPane.setSelectedIndex(0);

        Component panel2 = makeTextPanel("Blah blah");
        tabbedPane.addTab("Two", icon, panel2, "Does twice as much nothing");

        Component panel3 = makeTextPanel("Blah blah blah");
        tabbedPane.addTab("Three", icon, panel3, "Still does nothing");

        Component panel4 = makeTextPanel("Blah blah blah blah");
        tabbedPane.addTab("Four", icon, panel4, "Does nothing at all");

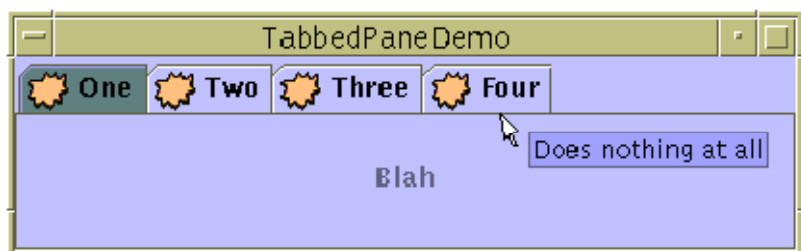
        //Add the tabbed pane to this panel.
        setLayout(new GridLayout(1, 1));
        add(tabbedPane);
    }

    protected Component makeTextPanel(String text) {
        JPanel panel = new JPanel(false);
        JLabel filler = new JLabel(text);
        filler.setHorizontalAlignment(JLabel.CENTER);
        panel.setLayout(new GridLayout(1, 1));
        panel.add(filler);
        return panel;
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("TabbedPaneDemo");

        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {System.exit(0);}
        });

        frame.getContentPane().add(new TabbedPaneDemo(),
                                   BorderLayout.CENTER);
        frame.setSize(400, 125);
        frame.setVisible(true);
    }
}
```



5.5.4. JSplitPane

Un **JSplitPane** contiene dos componentes de peso ligero, separados por un divisor. Arrastrando el divisor, el usuario puede especificar qué cantidad de área pertenece a cada componente. Un **SplitPane** se utiliza cuando dos componentes contienen información relacionada y queremos que el usuario pueda cambiar el tamaño de los componentes en relación a uno o a otro. Un uso común de un **SplitPane** es para contener listas de elecciones y una visión de la elección actual. Un ejemplo sería un programa de correo que muestra una lista con los mensajes y el contenido del mensaje actualmente seleccionado de la lista.

Define las constantes:

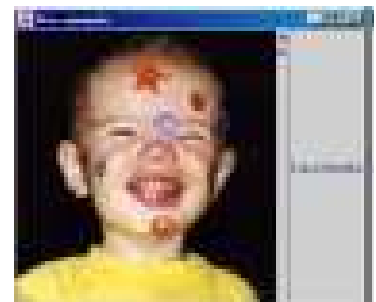
```
static int HORIZONTAL_SPLIT.  
static int VERTICAL_SPLIT
```

Alguno de los métodos que define son:

<i>Métodos de JSplitPane</i>	<i>Función que realiza</i>
<code>JSplitPane()</code> , <code>JSplitPane(int newOrientation)</code> , <code>JSplitPane(int newOrientation, Component newLeftComponent, Component newRightComponent)</code>	Constructores.
<code>Component getBottomComponent()</code> <code>Component getLeftComponent()</code> <code>Component getRightComponent()</code> <code>Component getTopComponent()</code> <code>void setBottomComponent(Component comp)</code> <code>void setLeftComponent(Component comp)</code> <code>void setRightComponent(Component comp)</code> <code>void setTopComponent(Component comp)</code>	Devuelve o establece el componente.
<code>int getDividerLocation()</code> <code>void setDividerLocation(int location)</code>	Posición de la barra de división.
<code>int getDividerSize()</code> <code>void setDividerSize(int newSize)</code>	Devuelve o establece el tamaño de la barra de división.
<code>int getOrientation()</code> , <code>void setOrientation(int orientation).</code>	Devuelve o establece la orientación.
<code>boolean isOneTouchExpandable()</code> , <code>void setOneTouchExpandable(boolean newValue)</code>	Devuelve o establece si es de tipo TouchExpandable .

A continuación se muestra un ejemplo:

```
import java.awt.*;  
import javax.swing.*;  
class GUISiP {  
    public static void main(String [] args) {  
        JFrame frame = new JFrame("Un ejemplo");  
        ImageIcon ii = new ImageIcon("carapin.gif");  
        JLabel label1 = new JLabel(ii);  
        JLabel label2 = new JLabel("Cara bonita");  
        JSplitPane splitPane = new  
        JSplitPane(JSplitPane.HORIZONTAL_SPLIT,  
        label1, label2);  
        splitPane.setOneTouchExpandable(true);  
        frame.getContentPane().add(splitPane);  
        frame.pack();  
        frame.setVisible(true);  
    }  
}
```



5.5.5. JToolBar

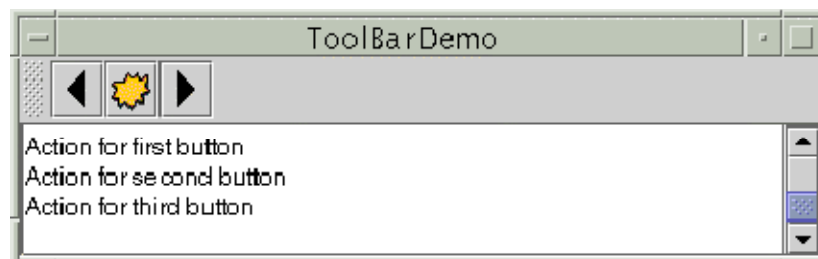
Representa una barra de herramientas en la que se insertan botones. Debe incluirse en un contenedor con BorderLayout. Usualmente contiene botones con iconos.

Define los siguientes métodos:

<i>Métodos de JToolBar</i>	<i>Función que realiza</i>
JToolBar(), JToolBar(int orientation)	Constructores. La orientación puede ser horizontal o vertical
JButton add(Action a)	Añade una acción a la barra.
void addSeparator(), void addSeparator(Dimension size)	Añade un separador.
Component getComponentAtIndex(int i), int getComponentIndex(Component c)	Localiza un componente.
setFloatable(boolean)	Por defecto flota.

Veamos a continuación un ejemplo:

```
JToolBar toolBar = new JToolBar();
JButton button = null;
button = new JButton(new ImageIcon("images/left.gif"));
toolBar.add(button);
button = new JButton(new ImageIcon("images/middle.gif"));
toolBar.add(button);
button = new JButton(new ImageIcon("images/right.gif"));
toolBar.add(button);
pane.add(toolBar, BorderLayout.NORTH)
```



5.6. Clase JLabel

La clase **JLabel** representa una etiqueta que puede contener un texto (no seleccionable y no editable), una imagen o ambas, que por defecto se alinea por la izquierda. La clase **JLabel** define las constantes JLabel.CENTER, JLabel.LEFT y JLabel.RIGHT para determinar la alineación del texto.

Además como esta clase deriva de **JComponent** implementa todas las características inherentes a los componentes Swing, como pueden ser los aceleradores de teclado, bordes, y demás. La siguiente tabla muestra algunos métodos de esta clase.

<i>Métodos de JLabel</i>	<i>Función que realiza</i>
JLabel(), JLabel(Icon image) JLabel(Icon image, int horizontalAlignment) JLabel(String text) JLabel(String text, Icon icon, int horizontalAlignment) JLabel(String text, int horizontalAlignment)	Constructores.
Icon getDisabledIcon() void setDisabledIcon(Icon disabledIcon)	Devuelve o establece la imagen que aparece cuando la etiqueta esta deshabilitada

int getHorizontalAlignment() void setHorizontalAlignment(int alignment)	Devuelve o establece la alineación horizontal.
int getHorizontalTextPosition() void setHorizontalTextPosition(int textPosition)	Devuelve o establece la alineación del texto respecto a la imagen.
Icon getIcon(), void setIcon(Icon icon)	Devuelve o establece la imagen.
int getIconTextGap(), void setIconTextGap(int iconTextGap)	Devuelve o establece los píxeles de separación entre la imagen y el texto.
String getText(), void setText(String text)	Devuelve o establece el texto.
int getVerticalAlignment(), void setVerticalAlignment(int alignment)	Alineación vertical.
int getVerticalTextPosition(), void setVerticalTextPosition(int textPosition)	Alineación vertical del texto respecto de la imagen.

Si se echa un vistazo a los constructores de la clase, se puede observar que Swing ofrece un API mucho más rico, presentando constructores con habilidades no disponibles en el AWT. El siguiente ejemplo crea varias instancias de la clase **JLabel** utilizando algunas características que permite Swing, como por ejemplo la utilización de gráficos, que resulta extremadamente simple.

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

```
@SuppressWarnings("serial")
class PruebaLabel extends JFrame
{
    // Constructor
    public PruebaLabel()
    {
        super("Ejemplo de Etiquetas");
        JPanel p = new JPanel();
        p.setLayout(new GridLayout(2,2));

        JLabel etiq1 = new JLabel();
        etiq1.setText("Etiqueta1");
        p.add(etiq1);

        JLabel etiq2 = new JLabel("Etiqueta2");
        etiq2.setFont(new Font("Helvetica", Font.BOLD, 18));
        p.add(etiq2);

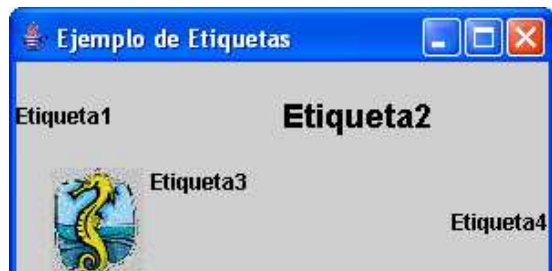
        Icon imagen = new ImageIcon("caballito.gif");
        JLabel etiq3 = new JLabel("Etiqueta3", imagen, SwingConstants.CENTER);
        etiq3.setVerticalTextPosition(SwingConstants.TOP);
        p.add(etiq3);

        JLabel etiq4 = new JLabel("Etiqueta4", JLabel.RIGHT);
        p.add(etiq4);

        // Añado el panel al contenedor principal (frame)
        getContentPane().add(p, BorderLayout.CENTER);

        // Control de Ventana
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        PruebaLabel ventana = new PruebaLabel();
        ventana.setSize(300,150);
        ventana.setVisible(true);
    }
}
```



Resultado del programa

5.7. Clases JButton y JToggleButton (JCheckBox y JRadioButton)

Swing aporta dos clases de botones JButton y JToggleButton. Estas clases derivan de una clase base AbstractButton. Algunos de los métodos definidos en esta clase son.

<i>Métodos de AbstractButton</i>	<i>Función que realiza</i>
void addActionListener(ActionListener l) void addChangeListener(ChangeListener l) void addItemListener(ItemListener l)	Métodos para el control de eventos (enlazan el componente con el oyente).
void doClick(), void doClick(int pressTime)	Lanza el evento clic.
String getActionCommand() void setActionCommand(String actionCommand)	Devuelve o establece la acción (<i>alias</i>).
Icon getDisabledIcon() Icon getDisabledSelectedIcon() Icon getIcon() Icon getPressedIcon() Icon getRolloverIcon() Icon getSelectedIcon() void setDisabledIcon(Icon disabledIcon) void setDisabledSelectedIcon(Icon disabledSelectedIcon) void setIcon(Icon defaultIcon) void setPressedIcon(Icon pressedIcon) void setRolloverEnabled(boolean b) void setSelectedIcon(Icon selectedIcon)	Devuelve o establece la imagen para el estado deshabilitado, deshabilitado-seleccionado, normal, pulsado, ratón sobre el botón y seleccionado.
int getHorizontalAlignment() void setHorizontalAlignment(int alignment) int getVerticalAlignment() void setVerticalAlignment(int alignment)	Alineación del texto.
int getHorizontalTextPosition() void setHorizontalTextPosition(int textPosition) int getVerticalTextPosition() void setVerticalTextPosition(int textPosition)	Alineación del texto respecto de la imagen.
int getIconTextGap() void setIconTextGap(int iconTextGap)	Devuelve o establece la separación entre imagen y texto.
Insets getMargin(), void setMargin(Insets m)	Devuelve o establece los márgenes.
String getText(), void setText(String text)	Devuelve o establece el texto.
boolean isSelected()	Indica si está seleccionado.

El aspecto de un **JButton** depende de la plataforma (PC, Mac, Unix), pero la funcionalidad siempre es la misma. Entre otras cosas, a un botón se le puede cambiar el *texto* y la *font*, así como el *foreground* y *background color*, establecer que esté activado o no. etc.

Ejemplo: JButton b1 = new JButton("Aceptar");

La clase **JToggleButton** representa un botón con dos estados posibles. Esta clase es usada para derivar de ella las clases **JCheckBox** y **JRadioButton**. Esta clase no aporta ningún método adicional.

Ejemplo: JToggleButton tb1 = new JToggleButton("JToggleButton");

Los Botones de Radio son grupos de botones en los que, por convención, sólo uno de ellos puede estar seleccionado. Swing soporta botones de radio con las clases **JRadioButton** y **ButtonGroup**. Los **JCheckbox** son similares a los **JRadioButton**, pero su modelo de selección es diferente, por convención. Cualquier número de checkboxes en un grupo, ninguno, alguno o todos, pueden ser seleccionados.

Cuando tengamos varias opciones que sean autoexcluyentes se usa la clase **ButtonGroup**. Se puede usar con cualquier clase que derive de **AbstractButton**. Inicialmente todos los botones están deseleccionados. Una vez seleccionado uno, siempre habrá uno seleccionado. Tiene como métodos:

Métodos de la clase ButtonGroup	Función que realiza
<code>ButtonGroup()</code>	Constructor.
<code>void add(AbstractButton b)</code>	Añade un botón al grupo.
<code>void remove(AbstractButton b)</code>	Elimina un botón del grupo

Los objetos de las clases **JCheckBox** y **JRadioButton** son *botones de selección* con dos posibles valores: **on** y **off**. Al cambiar la selección de un **JCheckBox** o un **JRadioButton** se produce un **ItemEvent**.

Los constructores de la clase **JCheckBox** son:

Métodos de JCheckBox	Función que realiza
<code>JCheckBox(String)</code> <code>JCheckBox(String, boolean)</code> <code>JCheckBox(Icon)</code> <code>JCheckBox(Icon, boolean)</code> <code>JCheckBox(String, Icon)</code> <code>JCheckBox(String, Icon, boolean)</code> <code>JCheckBox()</code>	Constructores. Crea un ejemplar de JCheckBox . El argumento string especifica el texto, si existe, que el checkbox debería mostrar. De forma similar, el argumento Icon especifica la imagen que debería utilizarse en vez de la imagen por defecto del aspecto y comportamiento. Especificando el argumento booleano como true se inicializa el checkbox como seleccionado. Si el argumento booleano no existe o es false , el checkbox estará inicialmente desactivado.

Ejemplo: `JCheckBox ch1 = new JCheckBox("Manzana", true);`
`JCheckBox ch2 = new JCheckBox("Melón", false);`

Los constructores de la clase **JRadioButton** son:

Métodos de JRadioButton	Función que realiza
<code>JRadioButton(String)</code> <code>JRadioButton(String, boolean)</code> <code>JRadioButton(Icon)</code> <code>JRadioButton(Icon, boolean)</code> <code>JRadioButton(String, Icon)</code> <code>JRadioButton(String, Icon, boolean)</code> <code>JRadioButton()</code>	Constructores. Crea un ejemplar de JRadioButton . El argumento string especifica el texto, si existe, que debe mostrar el botón de radio. Similarmente, el argumento, Icon especifica la imagen que debe usar en vez la imagen por defecto de un botón de radio para el aspecto y comportamiento. Si se especifica true en el argumento booleano, inicializa el botón de radio como seleccionado, sujeto a la aprobación del objeto ButtonGroup . Si el argumento booleano esta ausente o es false , el botón de radio está inicialmente deseleccionado.

Ejemplo: `ButtonGroup gcb = new ButtonGroup();`
`JRadioButton rb1 = new JRadioButton("Hombre", false);`
`JRadioButton rb2 = new JRadioButton("Mujer", true);`
`gcb.add(rb1);`
`gcb.add(rb2);`

Todos los botones tienen la posibilidad de incorporar imágenes a través del objeto **Icon**, que se puede asignar a cualquier tipo de botón. E incluso se pueden asignar varios iconos a un mismo botón para visualizar los diferentes estados en que pueda encontrarse dicho botón.

5.8. Clase JTextComponent (JTextField, JTextArea, JPasswordField)

Todos los componentes de texto derivan de la clase **JTextComponent** y muestran texto seleccionable y editable. Para controlar el cursor de inserción se utiliza la clase **Caret**. Esta clase permite establecer tanto el punto (dot) como el lugar hasta el que esta la selección (mark). Además se pueden usar las funciones ya conocidas como `setSelectionStart`, etc.

En todos los componentes de texto se utiliza la interfaz **Document**. Un document representa una plantilla para representar el texto (por ejemplo texto plano, html, rtf) permitiendo establecer cómo se actúa ante el texto introducido por teclado.

Es a través de **Document** como se puede controlar el cambio en el valor del texto introducido.

```
DocumentListener myListener = ??;
JTextArea myArea = ??;
myArea.getDocument().addDocumentListener(myListener);
```

<i>Métodos de JTextComponent</i>	<i>Función que realiza</i>
<code>void copy(), void cut(), void paste()</code>	Copia, corta o pega del portapapeles.
<code>Caret getCaret(), Color getCaretColor(), int getCaretPosition(), void setCaret(Caret c), void setCaretColor(Color c), void setCaretPosition(int position)</code>	Devuelve o establece el cursor, el color y la posición del mismo
<code>String getSelectedText()</code>	Devuelve el texto seleccionado.
<code>int getSelectionStart(), int getSelectionEnd(), void setSelectionStart(int selectionStart), void setSelectionEnd(int selectionEnd)</code>	Devuelve o establece los puntos de inicio y fin de la selección.
<code>String getText(), String getText(int offs, int len), void setText(String t)</code>	Devuelve o establece el texto.
<code>String getToolTipText(MouseEvent event)</code>	Devuelve el texto desplegable.
<code>void setMargin(Insets m)</code>	Establece los márgenes para el componente.
<code>boolean isEditable(), void setEditable(boolean b)</code>	Devuelve o establece si es modificable el contenido.
<code>void moveCaretPosition(int pos)</code>	Desplaza el cursor seleccionando texto.
<code>void select(int selectionStart,int selectionEnd) void selectAll()</code>	Selecciona texto.

Para la edición de una línea de texto se definen en swing dos componentes **JTextField** y **JPasswordField**. El primero es equivalente a **TextField** mientras que el segundo se utiliza cuando se quiere leer un texto con un carácter de eco.

<i>Métodos de JTextField</i>	<i>Función que realiza</i>
<code>JTextField(), JTextField(int columns), JTextField(String text), JTextField(String text, int columns)</code>	Constructores.
<code>int getColumns(), void setColumns(int columns)</code>	Establece el numero de columnas
<code>int getHorizontalAlignment(), void setHorizontalAlignment(int alignment)</code>	Devuelve o establece la alineación del texto.
<code>Dimension getPreferredSize()</code>	Devuelve el tamaño aconsejado
<code>void setFont(Font f)</code>	Establece la fuente.

<i>Métodos de JPasswordField</i>	<i>Función que realiza</i>
JPasswordField(), JPasswordField(int columns), JPasswordField(String text), JPasswordField(String text, int columns)	Constructores.
char getEchoChar(), void setEchoChar(char c)	Devuelve o establece el carácter de eco.
char[] getPassword()	Devuelve el texto en claro.

Para la edición de varias líneas de texto, swing define el componente **JTextArea**. Este componente es equivalente a un `TextArea` con la salvedad de que no es autoscrollable por lo que si queremos que automáticamente se realice un desplazamiento deberemos situar el componente dentro de un `JScrollPane`.

La diferencia principal es que **JTextField** sólo puede tener una línea, mientras que **JTextArea** puede tener varias líneas. Además, **JTextArea** ofrece posibilidades de edición de texto adicionales.

Ejemplo:

```
JTextField tf = new JTextField(15);
JPasswordField tfPassword = new JPasswordField(8);
JTextArea areaEdicion = new JTextArea("Hola amigo");

// Si queremos que el JTextArea tenga barras de desplamiento hay que
// colocarlo en un JScrollPane
JScrollPane sp = new JScrollPane(areaEdicion);
```

La clase **TextComponent** recibe eventos **TextEvent**, y por lo tanto también los reciben sus clases derivadas. Este evento se produce cada vez que se modifica el texto del componente. La clase **TextField** soporta también el evento **ActionEvent**, que se produce cada vez que el usuario termina de editar la única línea de texto pulsando **Intro**.

Como es natural, las cajas de texto pueden recibir también los eventos de sus *super-clases*, y más en concreto los eventos de **Component**: **FocusEvent**, **MouseEvent** y sobre todo **KeyEvent**. Estos eventos permiten capturar las teclas pulsadas por el usuario y tomar las medidas adecuadas. Por ejemplo, si el usuario debe teclear un número en un **TextField**, se puede crear una función que vaya capturando los caracteres tecleados y que rechace los que no sean numéricos.

Cuando se cambia desde programa el número de filas y de columnas de un `TextField` o `TextArea`, hay que llamar al método `validate()` de la clase `Component`, para que vuelva a aplicar el `LayoutManager` correspondiente. De todas formas, los tamaños fijados por el usuario tienen el carácter de “recomendaciones” o tamaños “preferidos”, que el `LayoutManager` puede cambiar si es necesario.

5.9. Clases de selección (**JComboBox**, **JList**, **JSpinner** y **JSlider**)

Swing define varios componentes para seleccionar elementos de una lista: **JComboBox**, **JList**, **JSpinner** y **JSlider**.

Un **JComboBox** es un componente que combina campo editable y una lista desplegable, dicho en otras palabras es una entrada de texto con lista de valores.

Con un **JComboBox** editable, una lista desplegable, y un text field, el usuario puede teclear un valor o elegirlo desde una lista. Un `ComboBox` editable ahorra tiempo de entrada proporcionando atajos para los valores más comúnmente introducidos.

Un **JComboBox** no editable desactiva el tecleo pero aún así permite al usuario seleccionar un valor desde una lista. Esto proporciona un espacio alternativo a un grupo de radio buttons o una list.

<i>Métodos de JComboBox</i>	<i>Función que realiza</i>
JComboBox(), JComboBox(ComboBoxModel aModel), JComboBox(Object[] items), JComboBox(Vector items)	Constructores.
void addActionListener(ActionListener l)	Controlador para el evento(ActionEvent)
void addItem(Object anObject)	Añade un ítem a la lista.
void addItemListener(ItemListener aLis)	Controlador para el ItemEvent
Action getAction(), void setAction(Action a)	Devuelve o establece una acción para el componente.
String getActionCommand(), void setActionCommand(String aCommand)	Devuelve o establece el texto para la acción.
Object getItemAt(int index), void insertItemAt(Object anObject, int index)	Devuelve o inserta un objeto en la posición indicada.
int getItemCount()	Devuelve el número de elementos.
int getMaximumRowCount(), void setMaximumRowCount(int count)	Devuelve o establece el número de filas que se despliegan en el popup.
int getSelectedIndex(), void setSelectedIndex(int anIndex)	Devuelve o establece el elemento seleccionado
Object getSelectedItem(), v oid setSelectedItem(Object anObject)	Devuelve o establece el elemento seleccionado.
boolean isEditable(), void setEditable(boolean aFlag)	Devuelve o establece si es editable.
boolean isPopupVisible(), void setPopupVisible(boolean v), void showPopup(), void hidePopup()	Muestra u oculta el popup.
void removeAllItems(), void removeItemAt(int anIndex)	Elimina elementos.

Ejemplo: JComboBox c = new JComboBox();
c.addItem("Elección 1");
c.addItem("Elección 2");
c.addItem("Elección 3");

Un **JList** es un componente que permite mostrar un conjunto de posibles valores y seleccionar uno o más elementos. JList se apoya en varias interfaces para realizar determinadas operaciones:

- **ListModel.** Los elementos de un JList realmente son gestionados por objetos de esta interfaz. De esta forma podemos mostrar en un JList prácticamente toda clase de objetos (imágenes, etiquetas, etc). Tiene como métodos principales:
 - **Object getElementAt(int index) y int getSize().** Normalmente se usan objetos de una clase ya existente javax.swing.DefaultListModel.
- **ListSelectionModel.** Permite gestionar como se seleccionan los elementos.
 - Define una serie de constantes: static intMULTIPLE_INTERVAL_SELECTION, static intSINGLE_INTERVAL_SELECTION, static intSINGLE_SELECTION, que permiten respectivamente indicar que se puedan seleccionar diferentes elementos en diferentes posiciones, diferentes elementos en posiciones consecutivas o un solo elemento.

Así nos podríamos encontrar ejemplos como los que siguen:

```
String [] arrayOpciones = new String[10];
for (int i=0; i<10; i++)
    arrayOpciones[i]="Lista item #" + i;
JList lista=new JList(arrayOpciones);
JScrollPane spLista = new JScrollPane(lista);
```



```
String[] datos = {"one", "two", "three", "four"};
JList lista = new JList(datos);
lista.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
lista.setLayoutOrientation(JList.HORIZONTAL_WRAP);
lista.setVisibleRowCount(-1);

for(int i = 0; i < lista.getModel().getSize(); i++) {
    System.out.println(lista.getModel().getElementAt(i));
}

// Si se desea añadir en cualquier momento podríamos haber declarado
DefaultListModel modelo = new DefaultListModel();
JList lista=new JList(modelo);

//se añade un elemento nuevo
modelo.addElement("opcion nueva");
```

Define las siguientes constantes:

- **static int HORIZONTAL_WRAP**. Los elementos se distribuyen por filas.
- **static int VERTICAL**. Una sola columna de celdas.
- **static int VERTICAL_WRAP**. Los elementos se distribuyen en columnas.

Y los métodos:

Métodos de JList	Función que realiza
JList(), JList(ListModel dataModel), JList(Object[] listData), JList(Vector listData)	Constructores.
void addSelectionInterval(int anchor, int lead), void setSelectionInterval(int anchor, int lead), void removeSelectionInterval(int index0, int index1)	Añade o quita un intervalo de selección.
void clearSelection()	Elimina la selección.
void ensureIndexIsVisible(int index)	Se asegura que un elemento es visible.
int getFirstVisibleIndex(), int getLastVisibleIndex()	Devuelve el primer o el último elemento visible.
int getLayoutOrientation(), void setLayoutOrientation(int layoutOrientation)	Establece o devuelve como se colocan las celdas.
int getMaxSelectionIndex(), int getMinSelectionIndex()	Devuelve el mayor y el menor índice seleccionado.
ListModel getModel(), void setModel(ListModel model)	Devuelve o establece el modelo con el que trabaja (la lista de elementos).
int getSelectedIndex(), int[] getSelectedIndices(), void setSelectedIndex(int index), void setSelectedIndices(int[] indices)	Devuelve o establece selecciones.
Object getSelectedValue(), Object[] getSelectedValues(), void setSelectedValue(Object anObject, boolean shouldScroll)	Devuelve o establece selecciones.
int getSelectionMode(), void setSelectionMode(int selectionMode)	Devuelve o establece el criterio de selección.
ListSelectionModel getSelectionModel(), void setSelectionModel(ListSelectionModel selectionModel)	Devuelve o establece el modelo de selección.

<code>int getVisibleRowCount(),</code> <code>void setVisibleRowCount(int visibleRowCount)</code>	Devuelve o establece las filas visibles.
<code>Point indexToLocation(int index),</code> <code>int locationToIndex(Point location)</code>	Conversión entre punto e índice.
<code>boolean isSelectedIndex(int index)</code>	Indica si un índice esta seleccionado.
<code>boolean isSelectionEmpty()</code>	Indica si no hay nada seleccionado.
<code>void setListData(Object[] listData),</code> <code>void setListData(Vector listData)</code>	Establece los elementos de la lista.

Un **JSpinner** es un componente java en el que se presenta una caja de texto con dos flechitas en el lado derecho, una hacia arriba y la otra hacia abajo. En el campo de texto se muestra un valor. Con las flechitas arriba y abajo podemos incrementar o decrementar ese valor. Para ello trabaja con Modelos que establecen el tipo de dato que se puede seleccionar. Existen tres objetos ya implementados:

- `javax.swing.SpinnerNumberModel`. Define un modelo numérico (entero, real, etc). Normalmente se crea, bien sin indicar rango (cualquier entero) o bien indicando valor actual, valor mínimo, valor máximo y valor de paso (`new SpinnerNumberModel(3,2,15,1)`).
- `javax.swing.SpinnerListModel`. Model definido por una lista de valores, normalmente cadenas.
- `javax.swing.SpinnerDateModel`. Usado para manejar secuencias de fechas. Permite moverse a través de un valor (`Calendar.ERA`, `Calendar.YEAR`, `Calendar.MONTH`, `Calendar.WEEK_OF_YEAR`, `Calendar.WEEK_OF_MONTH`, `Calendar.DAY_OF_MONTH`, `Calendar.DAY_OF_YEAR`, `Calendar.DAY_OF_WEEK`, `Calendar.DAY_OF_WEEK_IN_MONTH`, `Calendar.AM_PM`, `Calendar.HOUR`, `Calendar.HOUR_OF_DAY`, `Calendar.MINUTE`, `Calendar.SECOND`, `Calendar.MILLISECOND`).

Esta clase define los siguientes métodos:

<i>Métodos de JSpinner</i>	<i>Función que realiza</i>
<code>JSpinner()</code> , <code>JSpinner(SpinnerModel model)</code>	Constructores.
<code>void commitEdit()</code>	Salva el dato actualmenete editado.
<code>SpinnerModel getModel(),</code> <code>void setModel(SpinnerModel model)</code>	Devuelve o establece el modelo con el que se trabaja.
<code>Object getNextValue(),</code> <code>Object getPreviousValue(),</code> <code>Object getValue()</code>	Devuelve valores.
<code>void setValue(Object value)</code>	Establece un nuevo valor.

Ejemplo: `Jspinner = new JSpinner();`
`spinner.setValue(30);`

Por último, un **Slider** es un componente que nos permite seleccionar un valor numérico limitado por un valor máximo y mínimo de una forma gráfica. Para ello se establecen marcas (`MajorTick` y `MinorTick`) y nos aparece un marcador de posición.

Define los siguientes métodos:

<i>Métodos de JSlider</i>	<i>Función que realiza</i>
JSlider(), JSlider(int orientation), JSlider(int min, int max), JSlider(int min, int max, int value), JSlider(int orientation, int min, int max, int value)	Constructores.
void addChangeListener(ChangeListener l)	Añade un controlador para ChangeEvent.
boolean getInverted(), void setInverted(boolean b)	Devuelve o establece si se muestra el rango invertido.
int getMajorTickSpacing(), int getMinorTickSpacing(), void setMajorTickSpacing(int n), void setMinorTickSpacing(int n)	Devuelven o establecen los valores para las marcas.
int getMaximum(), int getMinimum(), void setMaximum(int maximum), void setMinimum(int minimum)	Devuelven o establecen los valores máximo y mínimo.
int getOrientation(), void setOrientation(int orientation)	Devuelve o establece la orientación.
boolean getPaintLabels(), boolean getPaintTicks(), boolean getPaintTrack(), void setPaintLabels(boolean b), void setPaintTicks(boolean b), void setPaintTrack(boolean b)	Devuelve o establece que elementos se visualizan.
int getValue(), void setValue(int n)	Devuelve o establece el valor actual.

Ejemplo: `JSlider sliderDiametro = new JSlider(SwingConstants.HORIZONTAL, 0, 200, 10);
sliderDiametro.setMajorTickSpacing(10);
sliderDiametro.setPaintTicks(true);`

5.10. La clase JProgressBar

El componente **JProgressBar** es una barra de progreso. Se le indica valores inicial, final y actual. Además permite trabajar en un modo indeterminado en el que no se conocen los límites.

Tiene como métodos:

<i>Métodos de JProgressBar</i>	<i>Función que realiza</i>
JProgressBar(int orient), JProgressBar(int min, int max), JProgressBar(int orient, int min, int max)	Constructores.
void addChangeListener(ChangeListener l)	Controlador de evento ChangeEvent.
int getMaximum(), int getMinimum(), void setMaximum(int n), void setMinimum(int n)	Establece o devuelve el máximo o el mínimo.
int getOrientation(), void setOrientation(int newOrientation)	Establece la orientación.
double getPercentComplete(), void setString(String s)	Devuelve el porcentaje actual.
int getValue(), void setValue(int n)	Devuelve o establece el valor.
boolean isBorderPainted(), void setBorderPainted(boolean b)	Devuelve o establece si se pintan los bordes.
boolean isIndeterminate(), void setIndeterminate(boolean newValue)	Devuelve o establece si es indeterminado.
boolean isStringPainted(), void setStringPainted(boolean b)	Indica si se muestra l porcentaje realizado

5.12. JOptionPane

La clase **JOptionPane** ofrece diversos métodos que se pueden usar para crear cuadros de diálogo modales estándar: pequeñas ventanas que hacen una pregunta, avisan a un usuario, o proporcionan mensajes breves e importantes.



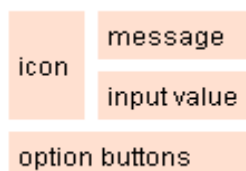
Para crear un diálogo, simple y estándar se utiliza **JOptionPane**. Para crear diálogos personalizados, se utiliza directamente la clase ya vista **JDialog**.

Estas ventanas son una forma efectiva de comunicarse con el usuario sin tener que crear una nueva clase para representar la ventana, añadiéndole componentes y escribiendo métodos gestores de eventos para tomar datos de entrada. Todas estas cosas se manejan de manera automática cuando se usa uno de los cuadros de diálogo estándar ofrecidos por **JOptionPane**.

Existen varios tipos de ventanas que se muestran llamando a los métodos estáticos:

- **showConfirmDialog.** Hace una pregunta para las respuestas si, no, cancelar.
- **showInputDialog.** Pide información al usuario, es decir, entrada de texto.
- **showMessageDialog.** Muestra un mensaje al usuario.
- **showOptionDialog.** Ventana para cualquier función, comprende los otros tres tipos de cuadros de diálogo.

Las ventanas tienen como diseño:



Estas funciones tienen como parámetros:

- **Componente padre:** Normalmente la ventana desde la que se ha llamado. Se usa para colocar el dialogo en relación a ella. Si se pasa null, se coloca la ventana centrada.
- **Mensaje:** Mensaje descriptivo que aparece en el dialogo, normalmente un String. Se podrían usar tambien:
 - **Object[]** Se colocan los objetos de forma vertical, cada uno en una linea.
 - **Component.** Se coloca el componente.
 - **Icon.** Crea un JLabel y lo muestra.
 - **Otros.** Se convierte el objeto a String y se muestra el resultado en un JLabel.
- **Tipo mensaje:** Hace que se muestre de diferente forma y, en algunos casos, muestre un icono específico. Posibles valores: **ERROR_MESSAGE**, **INFORMATION_MESSAGE**, **WARNING_MESSAGE**, **QUESTION_MESSAGE**, **PLAIN_MESSAGE**.
- **Tipo opción.** Define el conjunto de botones que se muestran. Pueden ser: **DEFAULT_OPTION**, **YES_NO_OPTION**, **YES_NO_CANCEL_OPTION**, **OK_CANCEL_OPTION**.

El propio usuario puede definir botones a mostrar

- **Opciones.** Definición más detallada de los botones que pueden aparecer. Normalmente se usa un array de Strings para indicar las etiquetas de los botones.
- **Icono.** Imagen decorativa.
- **Titulo.** Titulo para la ventana.
- **Valor inicial.**

Cuando un dialog devuelve un entero, los posibles valores son: **YES_OPTION**, **NO_OPTION**, **CANCEL_OPTION**, **OK_OPTION**, **CLOSED_OPTION**.

Algunos de los métodos de esta clase son, muchos de ellos son métodos de clase:

<i>Métodos de JOptionPane</i>	<i>Función que realiza</i>
JOptionPane() JOptionPane(Object) JOptionPane(Object, int) JOptionPane(Object, int, int) JOptionPane(Object, int, int, Icon) JOptionPane(Object, int, int, Icon, Object[]) JOptionPane(Object, int, int, Icon, Object[], Object)	Constructores
Frame getFrameForComponent(Component) JDesktopPane getDesktopPaneForComponent(Component)	Manejan métodos de clase de JOptionPane que encuentran el frame o desktop pane, respectivamente, en el que se encuentra el componente especificado.
int showMessageDialog(Component, Object) int showMessageDialog(Component, Object, String, int) int showMessageDialog(Component, Object, String, int, Icon)	Muestra un diálogo modal con un botón.
int showOptionDialog(Component, Object, String, int, int, Icon, Object[], Object)	Muestra un diálogo.
int showConfirmDialog(Component, Object) int showConfirmDialog(Component, Object, String, int) int showConfirmDialog(Component, Object, String, int, int) int showConfirmDialog(Component, Object, String, int, int, Icon)	Muestra un diálogo de pregunta.
String showInputDialog(Object) String showInputDialog(Component, Object) String showInputDialog(Component, Object, String, int) String showInputDialog(Component, Object, String, int, Icon, Object[], Object)	Muestra un diálogo de entrada.
int showInternalMessageDialog(...) int showInternalOptionDialog(...) int showInternalConfirmDialog(...) String showInternalInputDialog(...)	Implementa un diálogo estándar como un frame interno.

Por ejemplo podríamos tener:

```
JOptionPane.showMessageDialog(null, "alert", "alert", JOptionPane.ERROR_MESSAGE);

JOptionPane.showConfirmDialog(null, "choose one", "choose one",
JOptionPane.YES_NO_OPTION);
```

```
Object[] options = { "OK", "CANCEL" };
JOptionPane.showOptionDialog(null, "Click OK to continue", "Warning",
    JOptionPane.DEFAULT_OPTION,
    JOptionPane.WARNING_MESSAGE, null, options,
    options[0]);

String inputValue = JOptionPane.showInputDialog("Please input a value");

Object[] possibleValues = { "First", "Second", "Third" };
Object selectedValue = JOptionPane.showInputDialog(null, "Choose one", "Input",
    JOptionPane.INFORMATION_MESSAGE, null, possibleValues,
    possibleValues[0]);
```

5.13. JFileChooser

JFileChooser permite al usuario seleccionar un fichero para abrir o grabar, en otras palabras permite elegir un fichero de una lista. Un selector de ficheros es un componente que podemos situar en cualquier lugar del GUI de nuestro programa. Sin embargo, normalmente los programas los muestran en diálogos modales porque las operaciones con ficheros son sensibles a los cambios dentro del programa. La clase **JFileChooser** hace sencillo traer un diálogo modal que contiene un selector de ficheros.

Los selectores de ficheros se utilizan comunmente para dos propósitos:

- Para presentar una lista de ficheros que pueden ser **abiertos** por la aplicación.
- Para permitir que el usuario seleccione o introduzca el nombre de un fichero a **grabar**.

El selector de ficheros ni abre ni graba ficheros. Presenta un GUI para elegir un fichero de una lista. El programa es responsable de hacer algo con el fichero, como abrirlo o grabarlo.

Por defecto, un selector de ficheros muestra todos los ficheros y directorios que detecta. Un programa puede aplicar uno o más *filtros de ficheros* a un selector de ficheros para que el selector sólo muestre algunos de ellos. El selector de ficheros llama al método **accept** del filtro con cada fichero para determinar si debería ser mostrado. Un filtro de ficheros acepta o rechaza un fichero basándose en algún criterio como el tipo, el tamaño, el propietario, etc.

Algunos de los métodos de esta clase son:

<i>Métodos de JFileChooser</i>	<i>Función que realiza</i>
JFileChooser() JFileChooser(File currentDirectory) JFileChooser(String currentDirectoryPath)	Constructores.
JDialog createDialog(Component parent)	Crea una ventana de dialogo.
File getCurrentDirectory() void setCurrentDirectory(File dir)	Devuelve o establece el directorio en el que se busca.
String getDescription(File f)	Devuelve la descripción del fichero indicado.
String getDialogTitle() void setDialogTitle(String dialogTitle)	Devuelve o establece el título de la ventana.
FileFilter getFileFilter() void setFileFilter(FileFilter filter)	Devuelve o establece el filtro de ficheros aplicado.
File getSelectedFile() File[] getSelectedFiles()	Devuelve los ficheros seleccionados.
int showDialog(Component parent, String approveButtonText), int showOpenDialog(Component parent), int showSaveDialog(Component parent).	Abre el dialogo.

Ejemplo:

```
JFrame f = new JFrame("Ventana con dialogo de guardar un fichero");
FileChooser fc = new JFileChooser();
fc.showSaveDialog(f);
fc.setVisible(true);
```

5.14. Canvas

Una **Canvas** es una zona rectangular de pantalla en la que se puede dibujar y en la que se pueden generar eventos. Las **Canvas** permiten realizar dibujos, mostrar imágenes y crear componentes a medida, de modo que muestren un aspecto similar en todas las plataformas. La siguiente tabla muestra los métodos de la clase **Canvas**.

En realidad, este componente es propio de AWT, en Swing puede ser sustituido por JPanel, JLabel, u otro componente Swing apropiado.

Métodos de Canvas	Función que realiza
Canvas() Canvas (GraphicsConfiguration config)	Constructores
void paint(Graphics g);	Dibuja un rectángulo con el color de background. Lo normal es que las sub-clases de Canvas redefinan este método.

Desde los objetos de la clase **Canvas** se puede llamar a los métodos **paint()** y **repaint()** de la super-clase **Component**. Con frecuencia conviene redefinir los siguientes métodos de **Component**: **getPreferredSize()**, **getMinimumSize()** y **getMaximumSize()**, que devuelven un objeto de la clase **Dimension**. El **LayoutManager** se encarga de utilizar estos valores.

La clase **Canvas** no tiene eventos propios, pero puede recibir los eventos **ComponentEvent** de su super-clase **Component**.

Ejemplo:

```
Canvas c = new Canvas();
```

5.15. ScrollBar

Una **Scrollbar** es una barra de desplazamiento con un cursor que permite introducir y modificar valores, entre unos valores mínimo y máximo, con pequeños y grandes incrementos. Las **Scrollbars** de **Java** se utilizan tanto como “sliders” o barras de desplazamiento aisladas (al estilo de Visual Basic), como unidas a una ventana en posición vertical y/u horizontal para mostrar una cantidad de información superior a la que cabe en la ventana.

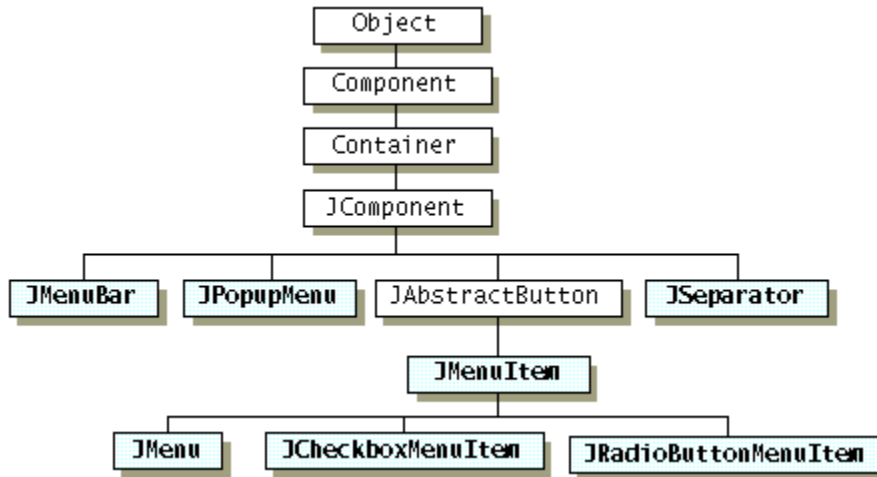
Al igual que ocurría en el apartado anterior, es un componente propio de AWT, en swing se puede utilizar en su lugar JScrollPane, JSlider o JProgressBar.

5.16. Menús

Se pueden añadir a los contenedores superiores. Entre sus características debemos destacar:

- ✓ **JMenuBar** es la barra de menú Swing. En ella se distribuyen los diferentes menús usando BorderLayout.
- ✓ Para añadir una barra de menú al contenedor, se utiliza el método: `void setJMenuBar(JMenuBar)`
- ✓ Los menús manejan tres elementos básicos:
 - Barra de Menú (JMenuBar)
 - Entrada de Menú (JMenu)
 - Item de entrada u opción de menú (JMenuItem, JCheckBoxMenuItem, JRadioButtonMenuItem)

- ✓ El menú de ayuda se añade a un JMenuBar (aún no está implementado): `void setHelpMenu(JMenu)`
- ✓ Un item puede ser a su vez un menú.
- ✓ Para añadir a un JMenuBar una entrada, se utiliza el método `add`.
- ✓ Para añadir entradas a un JMenu, se utiliza también el método `add`, si queremos añadir una línea de separación tenemos el método `addSeparator`.



Veamos a continuación algunos de los métodos de estas clases:

<i>Métodos de JMenuBar</i>	<i>Función que realiza</i>
<code>JMenuBar()</code>	Constructor
<code>JMenu add(JMenu c)</code>	Añade un menu a la barra.
<code>JMenu getMenu(int index)</code>	Devuelve un menu.
<code>int getMenuCount()</code>	Devuelve el número de menus.

<i>Métodos de JMenu</i>	<i>Función que realiza</i>
<code>JMenu()</code> , <code>JMenu(Action a)</code> , <code>JMenu(String s)</code> <code>JMenu(String s, boolean b)</code> .	Constructores.
<code>JMenuItem add(Action a)</code> <code>Component add(Component c)</code> <code>Component add(Component c, int index)</code> <code>JMenuItem add(JMenuItem menuItem)</code> <code>JMenuItem add(String s)</code>	Añade un elemento al menu.
<code>void addSeparator()</code>	Añade un separador.
<code>int getDelay()</code> , <code>void setDelay(int d)</code>	Devuelve o establece el tiempo de retardo de aparición.
<code>JMenuItem getItem(int pos)</code>	Devuelve el elemento pos.
<code>int getItemCount()</code>	Devuelve el número de elementos.
<code>Component getMenuComponent(int n)</code> .	Devuelve el componente n.
<code>int getMenuComponentCount()</code>	Devuelve el número de componentes del menu.
<code>JPopupMenu getPopupMenu()</code> .	Devuelve el popup asignado al menu.
<code>MenuElement[] getSubElements()</code>	Devuelve los subelementos.

JMenuItem insert(Action a, int pos) JMenuItem insert(JMenuItem mi, int pos) void insert(String s, int pos) void insertSeparator(int index)	Inserta un elemento en una posición.
boolean isMenuComponent(Component c)	Devuelve true si el componente es una opción del menú.
boolean isPopupMenuVisible() void setPopupMenuVisible(boolean b)	Devuelve o establece si es visible el popup.
boolean isSelected() void setSelected(boolean b)	Devuelve o establece si está seleccionado o no.
void remove(Component c), void remove(int pos) void remove(JMenuItem item) void removeAll()	Elimina uno o más componentes del menú.
void setMenuLocation(int x, int y).	Establece donde se debe mostrar el popup.

<i>Métodos de JMenuItem</i>	<i>Función que realiza</i>
JMenuItem(), JMenuItem(Action a) JMenuItem(Icon icon) JMenuItem(String text) JMenuItem(String text, Icon icon) JMenuItem(String text, int mnemonic)	Constructores
KeyStroke getAccelerator(), void setAccelerator(KeyStroke keyStroke)	Devuelve o establece el KeyStroke (Teclas aceleradoras, por ejemplo CTRL+A, ALT+B) ⁽¹⁾
MenuElement[] getSubElements()	Devuelve los elementos del submenú.
void setEnabled(boolean b)	Habilita la opción.

(1)

setAccelerator es un método que nos permite presionar una combinación de teclas para acceder rápidamente a un comando del menú. Recibe un objeto de tipo KeyStroke que podemos obtener mediante el método

```
KeyStroke.getKeyStroke(KeyEvent.VK_X, ActionEvent.MASK)
```

Donde:

X es una letra cualquiera, por ejemplo VK_A es la combinación que utilizamos para que el acceso directo utilice la tecla 'A'

MASK es la tecla extra que queremos que se presione, las teclas válidas son CTRL_MASK, ALT_MASK, SHIFT_MASK, META_MASK (para Mac) y 0 (cuando no se presiona ninguna tecla). Esta última es muy poco recomendable.

Otra forma: `KeyStroke.getKeyStroke('N', ActionEvent.ALT_MASK);`

<i>Métodos de JCheckBoxMenuItem</i>	<i>Función que realiza</i>
JCheckBoxMenuItem() JCheckBoxMenuItem(Action a) JCheckBoxMenuItem(Icon icon) JCheckBoxMenuItem(String text) JCheckBoxMenuItem(String text, boolean b) JCheckBoxMenuItem(String text, Icon icon) JCheckBoxMenuItem(String text, Icon icon, boolean b)	Constructores
boolean getState() void setState(boolean b)	Devuelve o establece el estado.

Los menús **JPopupMenu** son menús que aparecen en cualquier parte de la pantalla normalmente al clicar con el botón derecho del ratón (**pop-up trigger**) sobre un componente determinado (**parent Component**). El menú **pop-up** se muestra en unas coordenadas relativas al **parent Component**, que debe estar visible.

Algunos de sus métodos son:

<i>Métodos de JPopupMenu</i>	<i>Función que realiza</i>
JPopupMenu() JPopupMenu(String label)	Constructores
JMenuItem add(Action a), JMenuItem add(JMenuItem menuItem), JMenuItem add(String s), void addSeparator()	Añade una opción al menú.
int getComponentIndex(Component c)	Devuelve el índice del componente indicado.
Component getInvoker().	Devuelve el componente que invoca al popup.
MenuElement[] getSubElements()	Devuelve los elementos del menú.
void insert(Action a, int index), void insert(Component component, int index)	Inserta un elemento en una posición.
boolean isVisible()	Comprueba si es visible.
void pack()	Empaqueta la ventana.
void remove(int pos)	Elimina un elemento.
void setLocation(int x, int y)	Mueve el popup
void show(Component invoker, int x, int y)	Muestra el popup.
void setSelected(Component sel)	Establece la opción seleccionada.

Ejemplo:

```
// Crear la barra de menus
JMenuBar miBarraMenu = new JMenuBar();

// Crear un menú
JMenu menu = new JMenu("Menu 1");

// Crear las entradas en el menú
JMenuItem menuItem = new JMenuItem("Item 1");
menuItem.setAccelerator(
    KeyStroke.getKeyStroke(KeyEvent.VK_A, ActionEvent.CTRL_MASK));
JCheckBoxMenuItem menuItemCheck = new JCheckBoxMenuItem("Item con check");

// Añadir las entradas (opciones) al menú
menu.add(menuItem);
menu.addSeparator();
menu.add(menuItemCheck);

// Añadir a la barra de menus el menú
miBarraMenu.add(menu);

// Añadir o asociar la barra de menu al JFrame
this.setJMenuBar(miBarraMenu);
```

5.17. Tablas (JTable)

Con la clase **JTable**, se pueden mostrar tablas de datos, y opcionalmente permitir que el usuario los edite. **JTable** no contiene ni almacena datos; simplemente es una vista de nuestros datos. El siguiente ejemplo podemos ver una tabla típica mostrada en un JScrollPane:

Ejemplo:

```
import javax.swing.JTable;
import javax.swing.JScrollPane;
import javax.swing.JFrame;
import java.awt.*;
import java.awt.event.*;

public class SimpleTableDemo
extends JFrame {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private boolean DEBUG = true;

    public SimpleTableDemo() {
        super("SimpleTableDemo");

        Object[][] data = {
            {"Mary", "Campione",
             "Snowboarding", new Integer(5), new Boolean(false)},
            {"Alison", "Huml",
             "Rowing", new Integer(3), new Boolean(true)},
            {"Kathy", "Walrath",
             "Chasing toddlers", new Integer(2), new Boolean(false)},
            {"Mark", "Andrews",
             "Speed reading", new Integer(20), new Boolean(true)},
            {"Angela", "Lih",
             "Teaching high school", new Integer(4), new Boolean(false)}
        };

        String[] columnNames={"First Name","Last Name","Sport","# of Years","Vegetarian"};

        final JTable table = new JTable(data, columnNames);
        table.setPreferredScrollableViewportSize(new Dimension(500, 70));

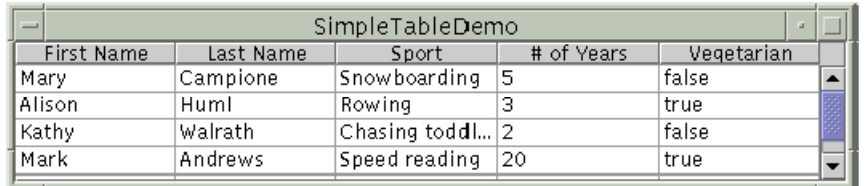
        if (DEBUG) {
            table.addMouseListener(new MouseAdapter() {
                public void mouseClicked(MouseEvent e) {
                    printDebugData(table);
                }
            });
        }

        //Create the scroll pane and add the table to it.
        JScrollPane scrollPane = new JScrollPane(table);

        //Add the scroll pane to this window.
        getContentPane().add(scrollPane, BorderLayout.CENTER);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    private void printDebugData(JTable table) {
        int numRows = table.getRowCount();
        int numCols = table.getColumnCount();
        javax.swing.table.TableModel model = table.getModel();
```



First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	false
Alison	Huml	Rowing	3	true
Kathy	Walrath	Chasing toddl...	2	false
Mark	Andrews	Speed reading	20	true

```

        System.out.println("Value of data: ");
        for (int i=0; i < numRows; i++) {
            System.out.print("    row " + i + " :");
            for (int j=0; j < numCols; j++) {
                System.out.print(" " + model.getValueAt(i, j));
            }
            System.out.println();
        }
        System.out.println("-----");
    }

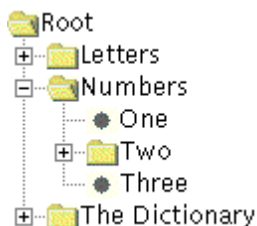
    public static void main(String[] args) {
        SimpleTableDemo frame = new SimpleTableDemo();
        frame.pack();
        frame.setVisible(true);
    }
}

```

En Tema_11_12_Anexo_swing, se puede ampliar la información sobre el manejo de las tablas (pág. 206)

5.18. Árboles (JTree)

Con la clase **JTree**, se puede mostrar un árbol de datos. **JTree** realmente no contiene datos, simplemente es un vista de ellos.



Como muestra la figura anterior, **JTree** muestra los datos verticalmente. Cada fila contiene exactamente un ítem de datos (llamado un *nodo*). Cada árbol tiene un nodo raíz (llamado Root en la figura anterior, del que descienden todos los nodos. Los nodos que no pueden tener hijos se llaman nodos *leaf* (hoja). En la figura anterior, el aspecto-y-comportamiento marca los nodos hojas con un círculo.

Los nodos que no sean hojas pueden tener cualquier número de hijos, o incluso no tenerlos. En la figura anterior, el aspecto-y-comportamiento marca los nodos que no son hojas con una carpeta. Normalmente el usuario puede expandir y contraer los nodos que no son hojas -- haciendo que sus hijos se vean visibles o invisibles -- pulsando sobre él. Por defecto, los nodos que no son hojas empiezan contraídos.

Cuando se inicializa un árbol, se crea un ejemplar de **TreeNode** para cada nodo del árbol, incluyendo el raíz. Cada nodo que no tenga hijos es una hoja. Para hacer que un nodo sin hijos no sea una hoja, se llama al método **setAllowsChildren(true)** sobre él.

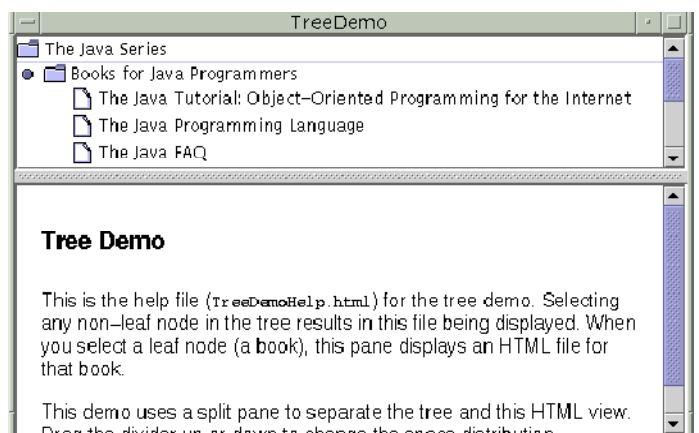
Ejemplo:

```

import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.event.TreeSelectionListener;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.tree.TreeSelectionModel;
import java.net.URL;
import java.io.IOException;
import javax.swing.JEditorPane;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JFrame;
import java.awt.*;
import java.awt.event.*;

public class TreeDemo extends JFrame {
    private JEditorPane htmlPane;
    private static boolean DEBUG = false;
    private URL helpURL;
}

```



```

public TreeDemo() {
    super("TreeDemo");

    //Create the nodes.
    DefaultMutableTreeNode top = new DefaultMutableTreeNode("The Java Series");
    createNodes(top);

    //Create a tree that allows one selection at a time.
    JTree tree = new JTree(top);
    tree.getSelectionModel().setSelectionMode
        (TreeSelectionMode.SINGLE_TREE_SELECTION);

    //Listen for when the selection changes.
    tree.addTreeSelectionListener(new TreeSelectionListener() {
        public void valueChanged(TreeSelectionEvent e) {
            DefaultMutableTreeNode node = (DefaultMutableTreeNode)
                (e.getPath().getLastPathComponent());
            Object nodeInfo = node.getUserObject();
            if (node.isLeaf()) {
                BookInfo book = (BookInfo)nodeInfo;
                displayURL(book.bookURL);
                if (DEBUG) {
                    System.out.print(book.bookURL + ": \n");
                }
            } else {
                displayURL(helpURL);
            }
            if (DEBUG) {
                System.out.println(nodeInfo.toString());
            }
        }
    });

    //Create the scroll pane and add the tree to it.
    JScrollPane treeView = new JScrollPane(tree);

    //Create the HTML viewing pane.
    htmlPane = new JEditorPane();
    htmlPane.setEditable(false);
    initHelp();
    JScrollPane htmlView = new JScrollPane(htmlPane);

    //Add the scroll panes to a split pane.
    JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
    splitPane.setTopComponent(treeView);
    splitPane.setBottomComponent(htmlView);

    Dimension minimumSize = new Dimension(100, 50);
    htmlView.setMinimumSize(minimumSize);
    treeView.setMinimumSize(minimumSize);
    splitPane.setDividerLocation(100); //XXX: ignored in some releases
                                         //of Swing. bug 4101306

    //workaround for bug 4101306:
    //treeView.setPreferredSize(new Dimension(100, 100));

    splitPane.setPreferredSize(new Dimension(500, 300));

    //Add the split pane to this frame
    getContentPane().add(splitPane);
}

private class BookInfo {
    public String bookName;
    public URL bookURL;
    public String prefix = "file:"
        + System.getProperty("user.dir")
        + System.getProperty("file.separator");

    public BookInfo(String book, String filename) {
        bookName = book;
        try {

```

```

        bookURL = new URL(prefix + filename);
    } catch (java.net.MalformedURLException exc) {
        System.err.println("Attempted to create a BookInfo "
            + "with a bad URL: " + bookURL);
        bookURL = null;
    }
}

public String toString() {
    return bookName;
}

private void initHelp() {
    String s = null;
    try {
        s = "file:"
            + System.getProperty("user.dir")
            + System.getProperty("file.separator")
            + "TreeDemoHelp.html";
        if (DEBUG) {
            System.out.println("Help URL is " + s);
        }
        helpURL = new URL(s);
        displayURL(helpURL);
    } catch (Exception e) {
        System.err.println("Couldn't create help URL: " + s);
    }
}

private void displayURL(URL url) {
    try {
        htmlPane.setPage(url);
    } catch (IOException e) {
        System.err.println("Attempted to read a bad URL: " + url);
    }
}

private void createNodes(DefaultMutableTreeNode top) {
    DefaultMutableTreeNode category = null;
    DefaultMutableTreeNode book = null;

    category = new DefaultMutableTreeNode("Books for Java Programmers");
    top.add(category);

    //Tutorial
    book = new DefaultMutableTreeNode(new BookInfo(
        "The Java Tutorial: Object-Oriented Programming for the Internet",
        "tutorial.html"));
    category.add(book);

    //Arnold/Gosling
    book = new DefaultMutableTreeNode(new BookInfo(
        "The Java Programming Language", "arnold.html"));
    category.add(book);

    //FAQ
    book = new DefaultMutableTreeNode(new BookInfo(
        "The Java FAQ", "faq.html"));
    category.add(book);

    //Chan/Lee
    book = new DefaultMutableTreeNode(new BookInfo(
        "The Java Class Libraries: An Annotated Reference",
        "chanlee.html"));
    category.add(book);

    //Threads
    book = new DefaultMutableTreeNode(new BookInfo(
        "Concurrent Programming in Java: Design Principles and Patterns",

```

```

        "thread.html"));
category.add(book);

category = new DefaultMutableTreeNode("Books for Java Implementers");
top.add(category);

//VM
book = new DefaultMutableTreeNode(new BookInfo
    ("The Java Virtual Machine Specification",
    "vm.html"));
category.add(book);

//Language Spec
book = new DefaultMutableTreeNode(new BookInfo
    ("The Java Language Specification",
    "jls.html"));
category.add(book);
}

public static void main(String[] args) {
    JFrame frame = new TreeDemo();

    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {System.exit(0);}
    });

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

```

En Tema_11_12_Anexo_swing, se puede ampliar la información sobre el manejo de árboles (pág. 268)

5.19. Tool Tips

Crear un tool tip para cualquier **JComponent** es fácil. Sólo debemos usar el método **setToolTipText** para configurar un tool tip para el componente. Por ejemplo, para añadir tool tips a tres botones, sólo tenemos que añadir tres líneas de código:

```

b1.setToolTipText("Click this button to disable the middle button.");
b2.setToolTipText("This middle button does nothing when you click it.");
b3.setToolTipText("Click this button to enable the middle button.");

```

Cuando el usuario del programa para el cursor sobre cualquiera de los botones, aparece el tool tip del botón. La siguiente imagen muestra el el tool tip que aparece cuando el cursor se para sobre el botón de la izquierda:



6. GESTORES DE ESQUEMAS (LAYOUT MANAGER).

La portabilidad de *Java* a distintas plataformas y distintos sistemas operativos necesita flexibilidad a la hora de situar los *Components* (*Buttons*, *Canvas*, *TextAreas*, etc.) en un *Container* (*Window*, *Panel*, ...). Un *Layout Manager* (gestor de esquemas) es un objeto que controla cómo los *Components* se sitúan en un *Container*.

El AWT define cinco *Layout Managers*: dos muy sencillos (*FlowLayout* y *GridLayout*), dos más especializados (*BorderLayout* y *CardLayout*) y uno muy general (*GridBagLayout*). Además, los usuarios pueden escribir su propio *Layout Manager*, implementando la interface *LayoutManager*, que especifica 5 métodos. Estos gestores también se utilizan en los contenedores Swing

Java permite también posicionar los *Components* de *modo absoluto*, sin *Layout Manager*, pero de ordinario puede perderse la portabilidad y algunas otras características.

Todos los *Containers* tienen un *Layout Manager* por defecto, que se utiliza si no se indica otra cosa: Para *Panel*, el defecto es un objeto de la clase *FlowLayout*. Para *Window* (*Frame* y *Dialog*), el defecto es un objeto de la clase *BorderLayout*.

Se debe elegir el *Layout Manager* que mejor se adecue a las necesidades de la aplicación que se desea desarrollar. Recuérdese que cada *Container* tiene un *Layout Manager* por defecto. Si se desea utilizar el *Layout Manager* por defecto basta crear el *Container* (su constructor crea un objeto del *Layout Manager* por defecto e inicializa el *Container* para hacer uso de él).

Para utilizar un *Layout Manager* diferente hay que crear un objeto de dicho *Layout Manager* y pasárselo al constructor del container o decirle a dicho container que lo utilice por medio del método *setLayout()*, en la forma:

```
unContainer.setLayout(new GridLayout());
```

La clase *Container* dispone de métodos para manejar el *Layout Manager* (ver Tabla siguiente).

Si se cambia de modo indirecto el tamaño de un *Component* (por ejemplo cambiando el tamaño del *Font*), hay que llamar al método *invalidate()* del *Component* y luego al método *validate()* del *Container*, lo que hace que se ejecute el método *doLayout()* para reajustar el espacio disponible.

Métodos de Container para manejar Layout Managers	Función que realizan
<code>add()</code>	Permite añadir Components a un Container
<code>remove()</code> y <code>removeAll()</code>	Permiten eliminar Components de un Container
<code>doLayout()</code> , <code>validate()</code> <code>doLayout()</code>	Se llama automáticamente cada vez que hay que redibujar el Container y sus Components. Se llama también cuando el usuario llama al método <code>validate()</code>

6.1. FlowLayout.

FlowLayout es el *Layout Manager* por defecto para *Panel* (*JPanel* en el caso de Swing). *FlowLayout* coloca los componentes en una fila, de izquierda a derecha y de arriba a abajo, en la misma forma en que procede un procesador de texto. Los componentes se añaden en el mismo orden en que se ejecutan los métodos *add()*. Si se cambia el tamaño de la ventana los componentes se redistribuyen de modo acorde, ocupando más filas si es necesario.

La clase *FlowLayout* tiene tres constructores:

```
FlowLayout();
FlowLayout(int alignment);
FlowLayout(int alignment, int horizontalGap, int verticalGap);
```


Se puede establecer la alineación de los componentes (centrados, por defecto), por medio de las constantes `FlowLayout.LEFT`, `FlowLayout.CENTER` y `FlowLayout.RIGHT`.

Es posible también establecer una distancia horizontal y vertical entre componentes (el **gap**, en pixels). El valor por defecto son 5 pixels.

6.2. BorderLayout.

BorderLayout es el **Layout Manager** por defecto para **Windows** y **Frames**. **BorderLayout** define cinco áreas: **North**, **South**, **East**, **West** y **Center**. Si se aumenta el tamaño de la ventana todas las zonas se mantienen en su mínimo tamaño posible excepto **Center**, que absorbe casi todo el crecimiento. Los componentes añadidos en cada zona tratan de ocupar todo el espacio disponible. Por ejemplo, si se añade un botón, el botón se hará tan grande como la celda, lo cual puede producir efectos muy extraños. Para evitar esto se puede introducir en la celda un panel con **FlowLayout** y añadir el botón al panel y el panel a la celda.

Los constructores de **BorderLayout** son los siguientes:

```
BorderLayout();
BorderLayout(int horizontalGap, int verticalGap);
```

Por defecto **BorderLayout** no deja espacio entre componentes. Al añadir un componente a un **Container** con **BorderLayout** se puede especificar la zona como primer argumento:

```
Container contentPane = this.getContentPane();
contentPane.add("North", new JButton("Norte"));
```

o también, como segundo argumento:

```
contentPane.add(new JButton("Norte"), BorderLayout.NORTH);
```

6.3. GridLayout.

Con **GridLayout** las componentes se colocan en una matriz de celdas. Todas las celdas tienen el mismo tamaño. Cada componente utiliza todo el espacio disponible en su celda, al igual que en **BorderLayout**.

GridLayout tiene dos constructores:

```
GridLayout(int nfil, int ncol);
GridLayout(int nfil, int ncol, int horizontalGap, int verticalGap);
```

Al menos uno de los parámetros **nfil** y **ncol** debe ser distinto de cero. El valor por defecto para el espacio entre filas y columnas es cero pixels.

6.4. CardLayout.

CardLayout permite disponer distintos componentes (de ordinario **Panels**) que comparten la misma ventana para ser mostrados sucesivamente. Son como transparencias, diapositivas o cartas de baraja que van apareciendo una detrás de otra.

El **orden** de las "cartas" se puede establecer de los siguientes modos:

1. Yendo a la primera o a la última, de acuerdo con el orden en que fueron añadidas al container.
2. Recorriendo las cartas hacia delante o hacia atrás, de una en una.
3. Mostrando una carta con un nombre determinado.

Los **constructores** de esta clase son:

```
CardLayout()
CardLayout(int horizGap, int vertGap)
```

Para añadir componentes a un container con **CardLayout** se utiliza el método:

```
Container.add(Component comp, int index)
```

donde **index** indica la posición en que hay que insertar la carta. Los siguientes métodos de **CardLayout** permiten controlar el orden en que aparecen las cartas:

```
void first(Container cont);
void last(Container cont);
void previous(Container cont);
void next(Container cont);
void show(Container cont, String nameCard);
```

6.5. GridBagLayout.

El **GridBagLayout** es el **Layout Manager** más completo y flexible, aunque también el más complicado de entender y de manejar. Al igual que el **GridLayout**, el **GridBagLayout** parte de una matriz de celdas en la que se sitúan los componentes. La diferencia está en que las filas pueden tener distinta altura, las columnas pueden tener distinta anchura, y además en el **GridBagLayout** un componente puede ocupar varias celdas contiguas.

La posición y el tamaño de cada componente se especifican por medio de unas “restricciones” o **constraints**. Las restricciones se establecen creando un objeto de la clase **GridBagConstraints**, dando valor a sus propiedades (variables miembro) y asociando ese objeto con el **GridBagLayout** por medio del método **setConstraints()**.

Las **variables miembro** de **GridBagConstraints** son las siguientes:

- **gridx** y **gridy**. Especifican la fila y la columna en la que situar la esquina superior izquierda del componente (se empieza a contar de cero). Con la constante **GridBagConstraints.RELATIVE** se indica que el componente se sitúa relativamente al anterior componente situado (es la condición por defecto).
- **gridwidth** y **gridheight**. Determinan el número de columnas y de filas que va a ocupar el componente. El valor por defecto es una columna y una fila. La constante **GridBagConstraints.REMAINDER** indica que el componente es el último de la columna o de la fila, mientras que **GridBagConstraints.RELATIVE** indica que el componente es el penúltimo de la fila o columna.
- **fill**. En el caso en que el componente sea más pequeño que el espacio reservado, esta variable indica si debe ocupar o no todo el espacio disponible. Los posibles valores son: **GridBagConstraints.NONE** (no lo ocupa; defecto), **GridBagConstraints.HORIZONTAL** (lo ocupa en dirección horizontal), **GridBagConstraints.VERTICAL** (lo ocupa en vertical) y **GridBagConstraints.BOTH** (lo ocupa en ambas direcciones).
- **ipadx** y **ipady**. Especifican el espacio a añadir en cada dirección al tamaño interno del componente. Los valores por defecto son cero. El tamaño del componente será el tamaño mínimo más dos veces el **ipadx** o el **ipady**.
- **insets**. Indican el espacio mínimo entre el componente y el espacio disponible. Se establece con un objeto de la clase **java.awt.Insets**. Por defecto es cero.
- **anchor**. Se utiliza para determinar dónde se coloca el componente, cuando éste es menor que el espacio disponible. Sus posibles valores vienen dados por las constantes de la clase **GridBagConstraints**: **CENTER** (el valor por defecto), **NORTH**, **NORTHEAST**, **EAST**, **SOUTHEAST**, **SOUTH**, **SOUTHWEST**, **WEST** y **NORTHWEST**.
- **weightx** y **weighty**. Son unos coeficientes entre 0.0 y 1.0 que sirven para dar más o menos “peso” a las distintas filas y columnas. A más “peso” más probabilidades tienen de que se les dé más anchura o más altura.

A continuación se muestra una forma típica de crear un container con **GridBagLayout** y de añadirle componentes:

```
GridBagLayout unGBL = new GridBagLayout();
GridBagConstraints unasConstr = new GridBagConstraints();
unContainer.setLayout(unGBL);
// Ahora ya se pueden añadir los componentes
//...Se crea un componente unComp
//...Se da valor a las variables del objeto unasConstr
// Se asocian las restricciones con el componente
unGBL.setConstraints(unComp, unasConstr);
// Se añade el componente al container
unContainer.add(unComp);
```

6.6. Nuevos layout Swing

Swing incorpora nuevos gestores, ampliando los cinco que *AWT* incorporaba. Entre ellos conviene destacar los siguientes:

- **BoxLayout**: Es similar al **FlowLayout** de *AWT*, con la diferencia de que con él se pueden especificar los ejes (x o y). Viene incorporada en el componente **Box**, pero está disponible como una opción en otros componentes. Se explica a continuación, al ser el más utilizado.
- **OverlayLayout**: Todos los componentes se añaden encima de cada componente previo.
- **SpringLayout**: El espacio se asigna en función de una serie de restricciones asociadas con cada componente.
- **ScrollPaneLayout**. Lo usa **JScrollPane**.
- **ViewportLayout**. Lo usa **JViewport**.

6.6.1. BoxLayout

Un **BoxLayout** está pensado para distribuir componentes como en un **GridLayout** o **GridBagLayout**, pero reduciendo la complejidad. Para ello se tienen en cuenta las propiedades de alineación y el tamaño máximo de los componentes.

Un **BoxLayout** permite distribuir los componentes formando una línea (**X_AXIS**) o una columna (**Y_AXIS**), intentando que todos los componentes ocupen la misma posición (en una línea, misma posición vertical) y tengan el mismo tamaño. Para modificar este comportamiento se trabaja con las propiedades **alignX(columna)** o **alignY(fila)** indicando valores que están en el rango 0 a 1

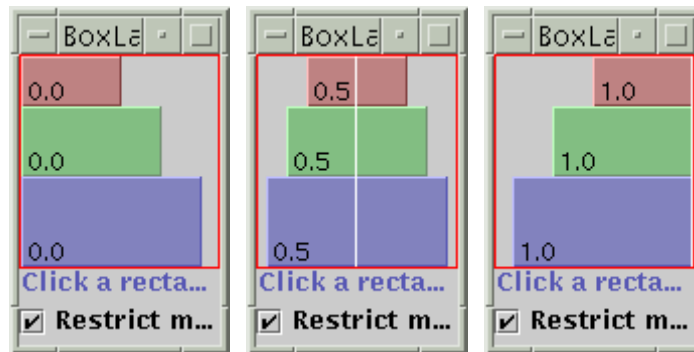
```
(0 = Component.LEFT_ALIGNMENT,
0.5 = Component.CENTER_ALIGNMENT,
1 = Component.RIGHT_ALIGNMENT)
```

El código para crear un panel que contenga una barra de botones vertical en el lado izquierdo quedaría así:

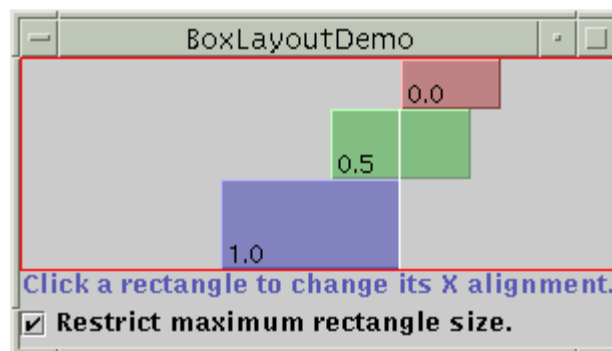
```
panelIzquierdo = new JPanel();
panelIzquierdo.setLayout(new BoxLayout(panelIzquierdo, BoxLayout.Y_AXIS));
panelIzquierdo.setBackground(Color.red);
panelIzquierdo.add(new JButton("1"));
panelIzquierdo.add(new JButton("2"));
panelIzquierdo.add(new JButton("3"));
```

El constructor del **BoxLayout** es más complejo que el del **FlowLayout**. Debemos pasarle el contenedor al que lo estamos añadiendo, es decir, el parámetro **panelIzquierdo**. También debemos pasarle si queremos orientación vertical **BoxLayout.Y_AXIS** u orientación horizontal **BoxLayout.X_AXIS**.

Por ejemplo si trabajamos con `Y_AXIS` y tres componentes podríamos encontrarlos:



Si además tienen definido un tamaño máximo menor al de la columna nos podemos encontrar con:



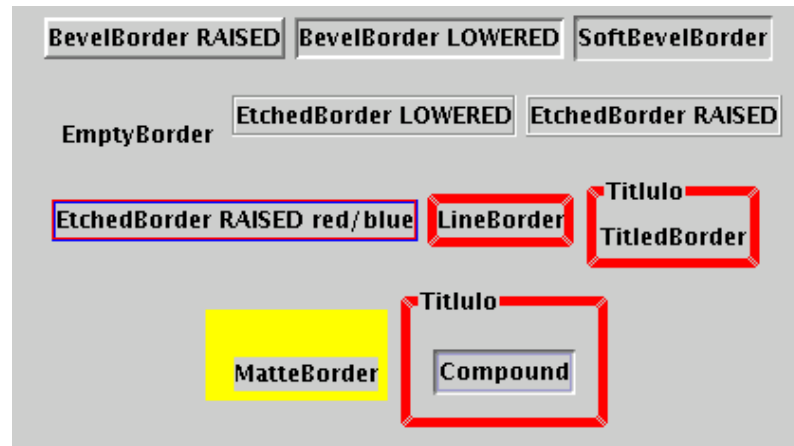
en donde se ajusta automáticamente el tamaño y la posición de los componentes.

Para completar el trabajo con este layout se utiliza la clase `Box` para el relleno de zonas y como elemento de separación entre componentes.

7.- BORDES

Los bordes se encuentran en el paquete `javax.swing.Border`. `JComponent` tiene la propiedad `border`. Se puede especificar el borde a un componente y a algunos contenedores con el método `setBorder()`. `javax.swing.Border` es un interface que define como es un borde. Algunas implementaciones de esta interface son:

Clase	Descripción
<code>BevelBorder</code>	Presenta al componente alzado o hundido.
<code>CompoundBorder</code>	Compone dos bordes en uno solo.
<code>EmptyBorder</code>	Borde para añadir espacio vacío.
<code>EtcherBorder</code>	Crea una línea resaltada o hundida.
<code>LineBorder</code>	Dibuja una línea de diferentes grosores..
<code>MatteBorder</code>	Crea un borde usando un color o una imagen.
<code>SoftBevelBorder</code>	Más suave que <code>BevelBorder</code> .
<code>TitledBorder</code>	Crea un borde añadiendo un título a otro borde.



Ejemplos:

```
// PRUEBA DE BORDES SOBRE UN PANEL
// EtcherBorder --> Crea una linea resaltada o hundida.
panell.setBorder(new EtchedBorder(EtchedBorder.RAISED));

// TitledBorder --> Crea un borde añadiendo un titulo a otro borde.
Panell.setBorder(new TitledBorder("Marca tu edad"));
panell.setBorder(new TitledBorder(new LineBorder(Color.blue,3,true),"Titulo"));

// LineBorder --> Dibuja una linea de diferentes grosores..
panell.setBorder(new LineBorder(Colo.red,5));

//MatteBorder --> Crea un borde usando un color o una imagen.
panell.setBorder(new MatteBorder(5,15,30,30,Color.green));

// BevelBorder --> Presenta al componente alzado o hundido.
panell.setBorder(new BevelBorder(BevelBorder.RAISED));

// SoftBevelBorder --> Mas suave que BevelBorder.
panell.setBorder(new SoftBevelBorder(BevelBorder.LOWERED));

// EmptyBorder --> Borde para añadir espacio vacio.
panell.setBorder(new EmptyBorder(5,15,30,30));

// CompoundBorder --> Compone dos bordes en uno solo
panell.setBorder(new CompoundBorder(new EtchedBorder(),
                                   new LineBorder(Color.red,3));
panell.setBorder(new CompoundBorder(new TitledBorder("Borde compuesto"),
                                   new BevelBorder(BevelBorder.LOWERED));
```

También es posible crear bordes propios y colocarlos dentro de botones, etiquetas, etc.; virtualmente en cualquier cosa que derive de **JComponent**.

8. Interfaz Icon y Clase ImageIcon

En algunos constructores y métodos aparece un argumento `Icon` que representa un icono. `Icon` es una interface. Para cargar un icono desde un fichero debemos crear un objeto de la clase `ImageIcon`.

```
Icon i = new ImageIcon("c:\\misIconos\\bruja.gif")
```

O bien

```
ImageIcon i = new ImageIcon("c:\\misIconos\\bruja.gif")
```

Representa una imagen que puede incorporarse a un JLabel y un JButton. Se pueden cargar imágenes desde ficheros jpg y gif.

Tiene como métodos principales:

<i>Métodos de ImageIcon</i>	<i>Función que realiza</i>
ImageIcon(), ImageIcon(Image image), ImageIcon(String filename), ImageIcon(URL location)	Constructores
String getDescription(), void setDescription(String description)	Devuelve o establece la descripción.
Image getImage(), void setImage(Image image)	Pone o devuelve la imagen.

9. LOOK AND FEEL

Look and Feel es la capacidad que tienen todos los componentes de Swing de modificar su entorno de visualización para adaptarlo al sistema operativo o al modelo definido por el usuario. Para gestionar todo el entorno de visualización se trabaja con la clase UIManager.

Se pueden asignar diversos entornos de visualización ya existentes mediante una llamada a **UIManager.setLookAndFeel(LookAndFeel newLookAndFeel)**, cuyo argumento puede ser:

- UIManager.getCrossPlatformLookAndFeelClassName(). Entorno de visualización Java.
- UIManager.getSystemLookAndFeelClassName(). Entorno de visualización según el sistema operativo.
- “javax.swing.plaf.metal.MetalLookAndFeel”. Entorno de visualización Metal.
- “com.sun.java.swing.plaf.windows.WindowsLookAndFeel”. Entorno de visualización Windows (sólo en plataformas windows).

NOTA: Puedes ampliar la información de este tema y del siguiente en el [Tema_11_12_Anexo_Swing.pdf](#)