TEMA 4

Programación Gráfica. Swing: Contenedores

Interface de Usuario

La interfaz de usuario es la parte del programa que permite a éste interactuar con el usuario. Las interfaz de usuario pueden adoptar muchas formas, que van desde la simple línea de comandos hasta las interfaces gráficas.

La interface de usuario es uno de los aspectos más importante de cualquier aplicación. Una aplicación sin un interfaz fácil y amigable, impide que los usuarios saquen el máximo rendimiento del programa. Java proporciona los elementos básicos para construir decentes interfaces de usuario a través de sus librerías gráficas AWT y SWING.

Al nivel más bajo, el sistema operativo transmite información desde el ratón y el teclado como dispositivos de entrada al programa. El AWT fue diseñado pensado en que el programador no tuviese que preocuparse de detalles como controlar el movimiento del ratón o leer el teclado, ni tampoco atender a detalles como la escritura en pantalla. El AWT constituye la primera librería de clases orientada a objeto para cubrir estos recursos y servicios de bajo nivel.

La librería AWT (Abstract Windows Toolkit)

Constituye una biblioteca de clases Java para el desarrollo de Interfaces de Gráficas de Usuario. Es una de las primeras bibliotecas de clases integrada en el JDK estándar. Se tardó en desarrollar tan sólo dos meses, debido a lo cual, el entorno que ofrece es demasiado simple. En su momento 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.

Estructura del AWT

La estructura de la versión actual del AWT podemos resumirla en los puntos que exponemos a continuación:

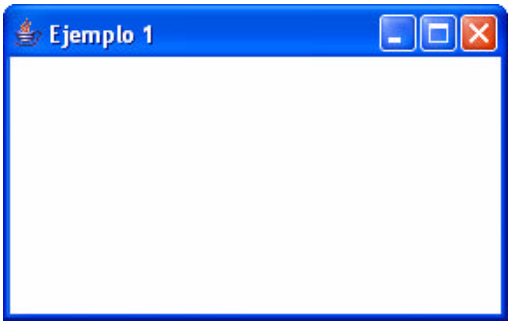
- Los Contenedores contienen Componentes, que son los controles básicos.
- No se usan posiciones fijas de los Componentes, sino que están situados a través de una disposición controlada (**layouts**).
- El común denominador de más bajo nivel se acerca al teclado, ratón y manejo de eventos.
- Alto nivel de abstracción respecto al entorno de ventanas en que se ejecute la aplicación (no hay áreas cliente, ni llamadas a X, ni hWnds, etc.).
- Es bastante dependiente de la máquina en que se ejecuta la aplicación (no puede asumir que un diálogo tendrá el mismo tamaño en cada máquina).

Componentes y Contenedores

Una interface gráfica está construida en base a elementos gráficos básicos, los Componentes. Típicos ejemplos de estos Componentes son los botones, barras de desplazamiento, etiquetas, listas, cajas de selección o campos de texto. Los Componentes permiten al usuario interactuar con la aplicación y proporcionar información desde el programa al usuario sobre el estado del programa. En el AWT, todos los Componentes de la interface de usuario son instancias de la clase **Component** o uno de sus subtipos.

Los Componentes no se encuentran aislados, sino agrupados dentro de Contenedores. Los Contenedores contienen y organizan la situación de los Componentes; además, los Contenedores son en sí mismos Componentes y como tales pueden ser situados dentro de otros Contenedores.

```
import java.awt.*;

class PrimerAWT {

   public static void main (String args[])  {
         Frame f = new Frame("Ejemplo 1");
         f.show();
   }
}
```



Ejemplo 4.1: Primera aplicación con AWT

También contienen el código necesario para el control de eventos, cambiar la forma del cursor o modificar el icono de la aplicación. En el AWT, todos los Contenedores son instancias de la clase Container o uno de sus subtipos.

La librería SWING

Hay que entender que Swing no es una sustitución de AWT, simplemente es una extensión de ésta, aunque a veces de esa sensación como en los distintos componentes como botones, paneles, etc. Swing, aparte de tener mayor cantidad de opciones sobre los componentes (como distintas apariencias, control sobre el focus, mayor número de campos que modifican su aspecto,...) se diferencian de las anteriores radicalmente en su implementación.

Por ejemplo, cuando en AWT se añadía un botón, el compilador generaba código que le pedía al S.O. la creación de un botón en un determinado sitio con unas determinadas propiedades, en Swing, sin embargo, no se le pide al S.O. nada, se dibuja el botón sobre la ventana en la que se desea y punto. Con esto se han eliminado muchos problemas que existían antes con los códigos de las interfaces gráficas, que debido a depender del S.O. para obtener sus componentes gráficos, era necesario testar los programas en distintos S.O., pudiendo tener distintos bugs en cada uno de ellos.

Con Swing se ha mejorado bastante este aspecto, lo único que se pide al S.O. es una ventana, una vez dibujada la ventana, se dibujan los botones, listas, scroll-bars.... y todo lo que se necesita sobre ella. Evidentemente esta aproximación gana mucho en lo que a independencia de la plataforma se refiere. Además, el hecho de que un botón no sea del S.O. sino que sea dibujado por Java ofrece un mayor control sobre su apariencia.

Para ejemplos, los más adecuados y espectaculares son los que proporciona “JavaSoft” con Swing, bajo la carpeta "demo" que se genera en la instalación del JDK. Particularmente interesante es la aplicación que ellos llaman “SwingSet2”, cuya pantalla inicial consta de una serie de pestañas que corresponden, cada una de ellas, a uno de los aspectos de Swing.

Ventanas en Swing

Es el principal contenedor que se emplea para situar en él todos los demás componentes que se necesiten para el desarrollo de la interface. La jerarquía de este tipo de componente parte, como todos desde Object, que digamos es la superclase de la cual heredan todos los objetos en Java.

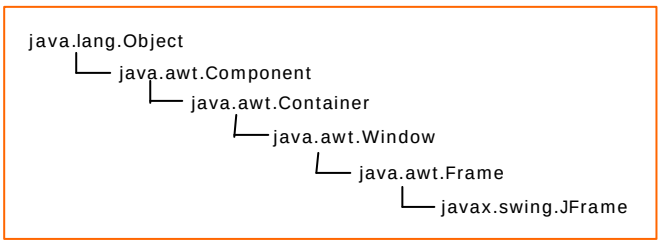
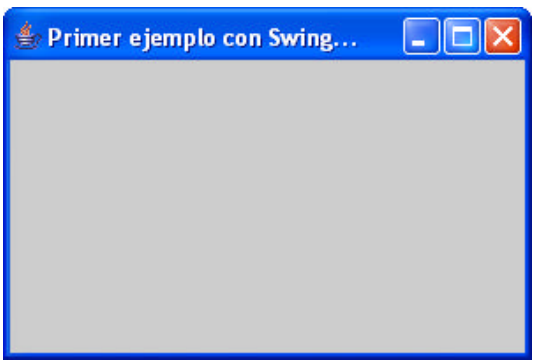


Fig. 4.2: Jerarquía de la clase JFrame

Por lo visto en el tema de herencias es correcto decir que los métodos de este componente están repartidos por las clases de las cuales hereda, por lo tanto resulta intuitivo que debiera haber un método para cambiar el color de fondo del frame, pero él no tiene ningún método para ello, lo tiene Component.

```
//Lo primero es importar la librería necesaria. En este caso importaremos
//todas las clases del paquete

import javax.swing.*;

class frame extends JFrame {

        //El constructor
        public frame() {
           //Este método es uno heredado de la clase JFrame.
           //Pone un título a la ventana.
           setTitle("Primer ejemplo con Swing...");
          //Utilizando otro método heredado señalamos el
          //tamaño de la ventana
          setSize(300,200);
        }
}

//Mediante la siguiente clase invocaremos a la anterior para la construcción del JFrame

public class ejemploFrame{

        public static void main(String[] args) {
                //Creamos el objeto frame
                JFrame f = new frame();
                //Por último invocamos al método heredado de JFrame,
                //show. Los frame por defecto son invisibles. Este es el modo
                // de hacerlos visibles
                f.show();
        }
}
```



Ejemplo 4.3: Primera aplicación con Swing

Qué bonito ha quedado, verdad?, pero si nos fijamos tiene un ligero problema, y es que no podemos cerrar la ventana. La única manera es hacer un break en el programa, Control + C, si lo estamos ejecutando desde una consola. La explicación es que se necesita cierto código que escuche los eventos que se puedan producir en la ventana, como por ejemplo, que se haga clic en el aspa de cierre de ventana. Necesitamos los eventos.

Paneles en Swing.

JPanel

Una vez construido la ventana (frame) se podría pensar en ir añadiendo los componentes que se fuesen necesitando sin más, pero no es considerado la mejor técnica de programación añadir componentes directamente sobre los contenedores pesados como son las ventanas o los applets, que veremos posteriormente.

Para ello se utiliza unos **contenedores menos pesados** que son los paneles. Las ventajas que tiene el empleo de paneles son varias, entre otras la **modularidad** en el diseño del interfaz ya que cada panel puede tener su propia disposición de elementos así como su propio diseño gracias a métodos como paintComponent, como ya se verá.

Para añadir un JPanel lo primero que se debe de hacer es obtener uno de los objetos que forman el frame o ventana, **el contenedor de paneles**, "content pane". Para ello se invocará el método getContentPane de la clase JFrame. El objeto que obtenemos es de tipo Container.

```
Container[nombre_del contentpane] = frame.getContentPane() ;
```

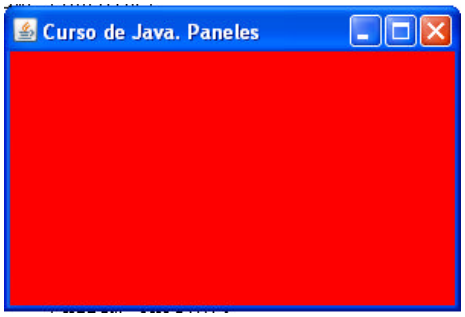
A continuación se invoca el método add del Container obtenido para añadir el panel, pasándole el propio panel al método:

```
[nombre_del_contentpane].add(nombre_del_panel);
```

Añadiendo un Panel al ejemplo, el código resultante sería el siguiente:

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.Color;
import java.awt.Container;

class frame extends JFrame {
        public frame()  {
         setTitle("Curso de Java. Paneles");
         setSize(300,200);
         //Le indicamos que tiene un  manejador de eventos asociado.
         addMouseListener(new manejadorMouse());
          addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e)
                        {
                                System.out.println("...Cerrando Ventana");
                                System.exit(0);
                        }
          });

          //Obtengo el objeto contenedor del frame
          Container contentpane = getContentPane();
          //Se crea un objeto de tipo JPanel
           JPanel panel = new JPanel();
          //Se añade el panel al objeto contenedor del frame
           contentpane.add(panel);
          //Para que se pueda apreciar el panel pongo su color de fondo verde
          panel.setBackground(Color.green);

        }
  }
//Definimos las clases que van a escuchar eventos
class manejadorMouse extends MouseAdapter{
        public void mousePressed(MouseEvent e)
          {
                System.out.println("...Mouse presionado");
           }
  }

//Por último escribimos la clase principal
public class ejemploEventos
  {
        public static void main(String[] args) {
          JFrame frame = new frame();
         frame.setVisible(true);
       }
  }
```

Ejemplo 4.4: Añadiendo un Panel a la ventana



Vista de la ventana creada por el programa anterior

- **Métodos Constructores**

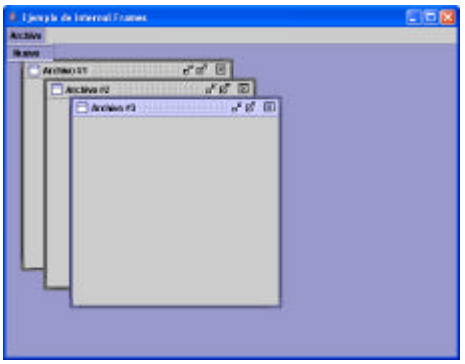
Métodos	Descripción
JPanel()	Crea un panel swing
JPanel(boolean)	Crea un panel swing. Un valor boolean true significa que el panel tiene doble buffer.
JPanel(LayoutManager)	Crea un panel swing conel gestor de presentación indicado. Por defecto el panel tendrá un gestor FlowLayout.
JPanel(LayoutManager, boolean)	Crea un panel swing con los valores indicados

- **Principales Métodos**

Métodos	Descripción
void add(Component) **void add(Component, int)** **void add(Component, Object)** **void add(Comonent,Object,int)** **void add(String,Component)**	Añade el componente especificado al panel. El parámetro String establece un nombre para el componente. El parámetro int indica la posición del componente dentro del panel y el parámetro Object proporciona información sobre el posicionamiento.
void remove(Component) **void remove(int)** **void removeAll()**	Elimina del panel los componentes especificados.
int getComponentCount()	Obtiene el número de componentes del panel
void setLayout(LayoutManager) **LayoutManager getLayout()**	Establece y obtiene, respectivamente, el gestor de presentación del panel.

JDesktopPane

Un LayeredPane o *panel por capas*, es un panel que proporciona una tercera dimensión para el posicionamiento de componentes: el **eje Z**, o dicho de otra manera, la **profundidad**. Cuando añadimos un componente a un panel por capas debemos especificar la profundidad que va a tener. Para ello, en un LayeredPane se definen varias capas o profundidades.



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

class internalFrame extends JFrame implements ActionListener{
      JDesktopPane desktop;
      static int numDeInternos=0;
      static final int xOffset=20;
      static final int yOffset=20;

      public internalFrame() {
        super("Ejemplo de Internal Frames");
        Dimension tamPantalla=Toolkit.getDefaultToolkit().getScreenSize();
        setBounds(100,100, tamPantalla.width-200, tamPantalla.height-200);
        desktop=new JDesktopPane();
        crearFrame();
        setContentPane(desktop);

        JMenuBar menuBar=new JMenuBar();
        JMenu menu=new JMenu("Archivo");
        JMenuItem menuItem=new JMenuItem("Nuevo");

        // añadimos item al menu y el menu a la barra
        menu.add(menuItem);
        menuBar.add(menu);
        setJMenuBar(menuBar);

        // controlamos la eleccion por el usuario de la opcion "Nuevo"
       menuItem.addActionListener(this);

      }
...
```

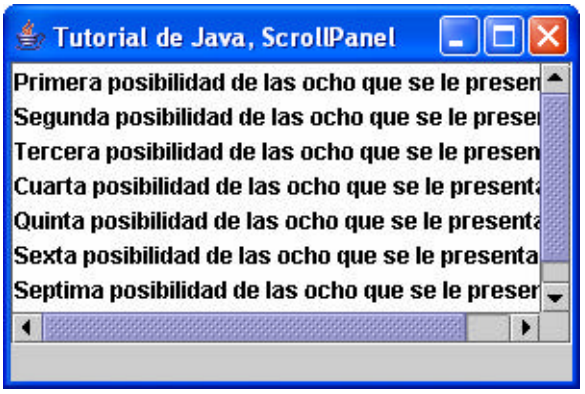
```
...
protected void crearFrame() {

     JInternalFrame frame= new JInternalFrame("Archivo #"+(++numDeInternos),
        true,  // resizable
        true,  // cerrable
        true,  // maximizable
        true);  // iconificable

     // hay que establecer  siempre el tamano
     frame.setSize(290,290);

     // hay que establecer siempre la localizacion

     frame.setLocation( xOffset*numDeInternos,yOffset*numDeInternos);

     try {
       frame.setSelected(true);  // activamos el actual frame
     } catch(java.beans.PropertyVetoException e) {System.out.println("Error cometido"); }


     desktop.add(frame);
     frame.setVisible(true); //Anteriormente a la versión 1.2 ->frame.show();

  }
  public void actionPerformed(ActionEvent e) {

    crearFrame();
  }
}

public class ejemploInternalFrame extends WindowAdapter {

    public static void main(String args[]) {

        internalFrame frame=new internalFrame();
        frame.setVisible(true);
    }

    public void WindowClosing(WindowEvent e) {
       System.exit(0);
    }
}
```

al no especificar un segundo argumento al metodo add, se esta indicando que el frame se incluya en la capa por defecto, esto es, DEFAULT_LAYER

Ejemplo 4.5: Construcción de un InternalFrame dentro de un JDesktopPane

JScrollPane

Mediante esta clase se crear una herramienta que permite una visión desplazable de otro componente. Un ScrollPane suele utilizarse para mostrar un componente que es muy grande y no puede observarse en su totalidad, o para mostrar un componente cuyo tamaño puede variar dinámicamente.




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

public class ejemploScrollPanel {
   public static void main( String args[] ) {
      // Aqui se instancia un objeto de tipo Interfaz Hombre-Maquina
      IHM ihm = new IHM();
      }
   }

// Esta clase se utiliza para instaciar un objeto de tipo interfaz de usuario
class IHM {

   public IHM() {
      // Crea un objeto visual Frame
      JFrame miFrame = new JFrame();
      miFrame.setSize( 300,200 );
      miFrame.setTitle( "Tutorial de Java, ScrollPanel" );
      miFrame.setName( "Frame" );

         JPanel panelprincipal=new JPanel(); //panel que contendrá al scrollpane
         JScrollPane miscroll;  // definimos el scrollpane
          //Definimos los elementos de la lista
         Object[] miarray={"Primera posibilidad de las ocho que se le presentan.",
                "Segunda posibilidad de las ocho que se le presentan.",
                "Tercera posibilidad de las ocho que se le presentan.",
                "Cuarta posibilidad de las ocho que se le presentan.",
                "Quinta posibilidad de las ocho que se le presentan.",
                "Sexta posibilidad de las ocho que se le presentan.",
                "Septima posibilidad de las ocho que se le presentan.",
                "Octava posibilidad de las ocho que se le presentan."};
...
```

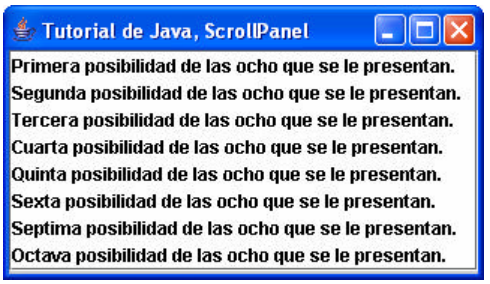
```
...
            JList milista=new JList(miarray);  // definimos la lista
            panelprincipal.setLayout(new BorderLayout());
            miscroll=new JScrollPane(milista);  // incluimos la lista en el scrollpane
            // incluimos el scrollpane en el panel
           panelprincipal.add(miscroll,BorderLayout.CENTER);

           // añadimos el panel que contiene el scrollpane a la ventana
          miFrame.getContentPane().add("North",panelprincipal);

       miFrame.setVisible(true);

       }
  }
```

Ejemplo 4.6: Utilización de un ScrollPanel

Vigilantes de los ScrollBars

Un ScrollPane solo muestra sus barras de desplazamiento o ScrollBars cuando son necesarias. Es decir, imaginemos que en la aplicación anterior agrandamos la ventana hasta que la *lista swing* cabe perfectamente en ella:

En este caso, ambas barras de desplazamiento ya no son necesarias. Este comportamiento está controlado por el **vigilante de ScrollBar** (realmente existen 2 vigilantes, uno por cada barra de desplazamiento).



Los vigilantes de ScrollBar pueden establecerse o fijarse a través de los métodos **setHorizontalScrollBarPolicy** y **setVerticalScrollBarPolicy**. Con ambos métodos se utilizan los enteros definidos en la interface **ScrollPaneConstants** que es implementada por JScrollPane. Los posibles valores de estos enteros son:

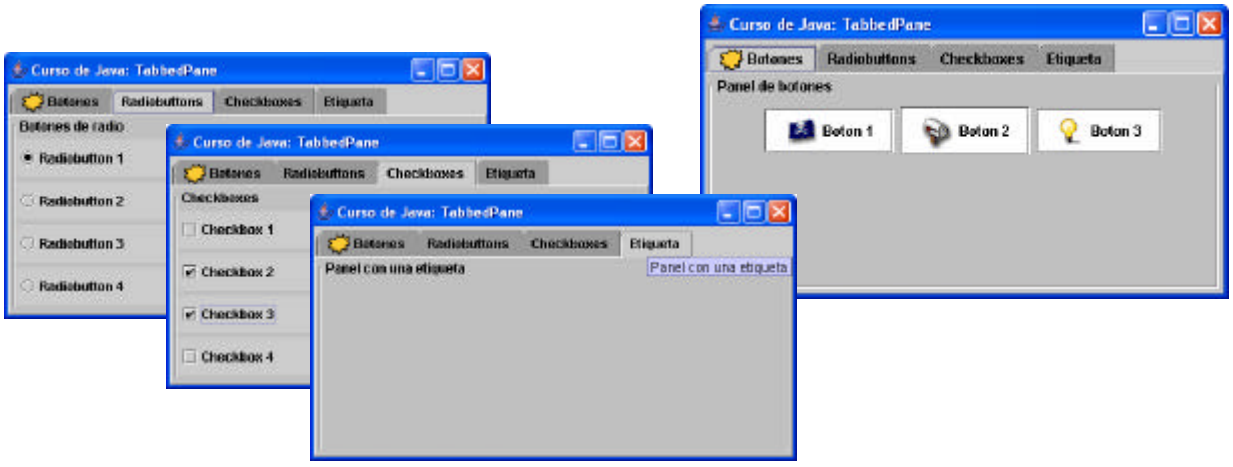
VERTICAL_SCROLLBAR_AS_NEEDED **HORIZONTAL_SCROLLBAR_AS_NEEDED**	Valor por defecto. Las barras de desplazamiento aparecerán únicamente cuando sean necesarias.
VERTICAL_SCROLLBAR_NEVER **HORIZONTAL_SCROLLBAR_NEVER**	Las barras de desplazamiento no aparecerán nunca.
VERTICAL_SCROLLBAR_ALWAYS **HORIZONTAL_SCROLLBAR_ALWAYS**	Las barras de desplazamiento aparecerán siempre.

- **Principales Métodos**

Métodos	Descripción
JScrollPane()	Crea un ScrollPane.
JScrollPane(Component)	Crea un ScrollPane con el cliente especificado. El cliente de un ScrollPane es lo que se muestra en dicho ScrollPane.
JScrollPane(int,int)	Crea un ScrollPane estableciendo los vigilantes de las barras de desplazamiento vertical y horizontal, respectivamente.
JScrollPane(Component,int,int)	Crea un ScrollPane con el cliente y los vigilantes de las barras de desplazamiento especificados.
void **setViewporView**(Component)	Selecciona el cliente del ScrollPane.
void **setVerticalScrollBarPolicy**(int) int **getVerticalScrollBarPolicy**()	Establece y obtiene, respectivamente, el vigilante de barra de desplazamiento vertical del ScrollPane.
void **setHorizontalScrollBarPolicy**(int) int **getHorizontalScrollBarPolicy**()	Establece y obtiene, respectivamente, el vigilante de barra de desplazamiento horizontal del ScrollPane.

JTabbedPane

La clase JTabbedPane, permite al usuario tener varios componentes compartiendo el mismo espacio, de manera que puede elegir el componente deseado mediante un clic en una pestaña.



Una de las ventajas de esta estructura, es que el programador no tiene que manejar eventos a la hora de cambiar el *panel activo* del JTabbedPane, ya que es el propio objeto quién se encarga de ello.

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

public class ejemploTabbedPane extends JFrame {

  public static int ANCHURA=450;
  public static int ALTURA=250;
  public static String TITULO="Curso de Java: TabbedPane";

  JTabbedPane mitabbedpane=new JTabbedPane();
  JPanel panelbotones=new JPanel();
  JPanel panelradio=new JPanel();
  JPanel panelcheckbox=new JPanel();
  JPanel panellabel=new JPanel();

  ImageIcon imagen1=new ImageIcon("images/boton1.gif");
  ImageIcon imagen2=new ImageIcon("images/boton3.gif");
  ImageIcon imagen3=new ImageIcon("images/boton2.gif");
  JButton boton1=new JButton("Boton 1",imagen1);
  JButton boton2=new JButton("Boton 2",imagen2);
  JButton boton3=new JButton("Boton 3",imagen3);

  ButtonGroup migrupo=new ButtonGroup();
  JRadioButton button1=new JRadioButton("Radiobutton 1");
  JRadioButton button2=new JRadioButton("Radiobutton 2");
  JRadioButton button3=new JRadioButton("Radiobutton 3");
  JRadioButton button4=new JRadioButton("Radiobutton 4");

  JCheckBox box1=new JCheckBox("Checkbox 1");
  JCheckBox box2=new JCheckBox("Checkbox 2");
  JCheckBox box3=new JCheckBox("Checkbox 3");
  JCheckBox box4=new JCheckBox("Checkbox 4");

  ImageIcon imagen=new ImageIcon("images/etiqueta.gif");
  JLabel label=new JLabel(imagen);

  public ejemploTabbedPane() {
    super(TITULO);
    setSize(ANCHURA,ALTURA);
    addWindowListener(new WindowHandler());
    ColocarComponentes();
    setBackground(Color.darkGray);  // color de fondo
    show();
  }

 void ColocarComponentes() {

// colocamos las componentes del panel de botones

  boton1.setBackground(Color.white);
  boton2.setBackground(Color.white);
  boton3.setBackground(Color.white);
...
```

```
...
    panelbotones.add(boton1);
    panelbotones.add(boton2);
    panelbotones.add(boton3);
    panelbotones.setBackground(Color.lightGray);

panelbotones.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(),"Pan
el de botones"));
    ImageIcon imagen4=new ImageIcon("images/middle.gif");
    mitabbedpane.addTab("Botones",imagen4,panelbotones,"Panel de botones");
    // colocamos las componentes del panel de radiobuttons

    panelradio.setLayout(new GridLayout(0,1,0,5));
    button1.setSelected(true);
    migrupo.add(button1);
    migrupo.add(button2);
    migrupo.add(button3);
    migrupo.add(button4);
    panelradio.add(button1);
    panelradio.add(button2);
    panelradio.add(button3);
    panelradio.add(button4);

    panelradio.setBackground(Color.lightGray);

panelradio.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(),"Botone
s de radio"));
    mitabbedpane.addTab("Radiobuttons",null,panelradio,"Panel de botones de radio");

    // colocamos las componentes del panel de checkboxes
    panelcheckbox.setLayout(new GridLayout(0,1,0,5));
    panelcheckbox.add(box1);
    panelcheckbox.add(box2);
    panelcheckbox.add(box3);
    panelcheckbox.add(box4);
    panelcheckbox.setBackground(Color.lightGray);

panelcheckbox.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(),"Ch
eckboxes"));
       mitabbedpane.addTab("Checkboxes",null,panelcheckbox,"Panel con checkboxes");

       // colocamos las componentes del panel que contiene la etiqueta

       panellabel.add(label);
       panellabel.setBackground(Color.lightGray);

   panellabel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(),"Panel
con una etiqueta"));
       mitabbedpane.addTab("Etiqueta",null,panellabel,"Panel con una etiqueta");

       this.getContentPane().add(mitabbedpane);
    }
    public static void main(String[] args) {
      ejemploTabbedPane ventana=new ejemploTabbedPane();
    }

    class WindowHandler extends WindowAdapter {
       public void windowClosing(WindowEvent e) {
          System.exit(0);
       }
    }
}
```

Ejemplo 4.7: Construcción de un JTabbedPane

- **Métodos constructores**

Métodos	Descripción
JTabbedPane ()	Crea un tabbed pane.
JTabbedPane (int)	Crea un tabbed pane indicando la posición de las pestañass. Estas posiciones (definidas en la interfaz *SwingConstants*, que JTabbedPane implementa) pueden ser: **TOP** (por defecto), **BOTTOM**, **LEFT** y **RIGHT**.

- **Principales métodos**

Métodos	Descripción
addTab(String,Component)	Añade una pestaña al tabbed pane. El primer argumento indica el texto de la pestaña. El segundo argumento indica la componente que el tabbed pane debe mostrar cuando la pestaña sea seleccionada.
addTab(String,Icon,Component)	Añade una pestaña al tabbed pane. Los argumentos *String* y *Component* son los mismos que en el caso anterior. El argumento *Icon* es el icono que se mostrará en la pestaña.
addTab(String,Icon,Component,String)	Añade una pestaña al tabbed pane. Los 3 primeros argumentos son los mismos que en el caso anterior. El último argumento especifica el tool tip de la pestaña.
insertTab(String,Icon,Component,String,int)	Inserta una pestaña en el índice (argumento *int*) especificado. La primera pestaña tiene índice 0.
void **setSelectedIndex**(int)	Selecciona la pestaña cuyo índice se indica. El efecto de seleccionar una pestaña es mostrar su componente asociada.
void **setSelectedComponent**(Component)	Selecciona la pestaña cuya componente se indica.
int **getSelectedIndex**()	Devuelve el índice de la pestaña seleccionada.
Component **getSelectedComponent**()	Devuelve la componente de la pestaña seleccionada.
int **indexOfTab**(String)	Devuelve el índice de la pestaña que tiene el título indicado.
int **indexOfTab**(Icon)	Devuelve el índice de la pestaña que tiene el icono indicado.
int **indexOfComponent**(Component)	Devuelve el índice de la pestaña que tiene la componente indicada.
removeTabAt(int)	Elimina la pestaña que tiene el índice especificado.
remove(Component)	Elimina la pestaña que tiene la componente especificada.
removeAll()	Elimina todas las pestañas.

JSplitPane

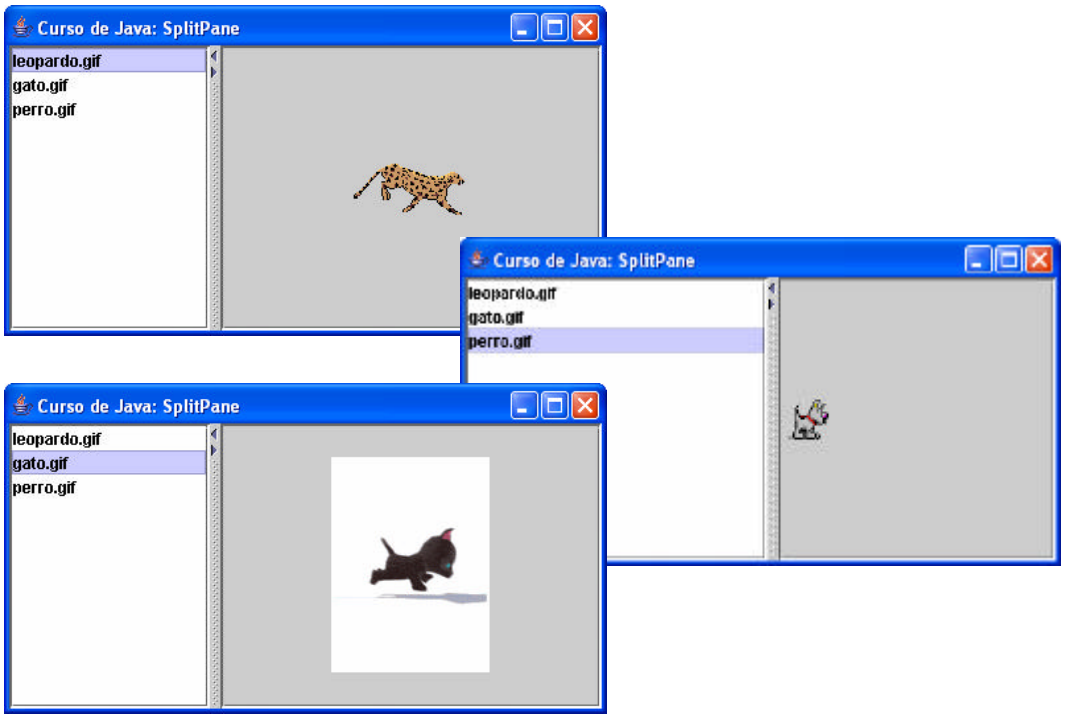
Un JSplitPane es un contenedor de swing que permite situar dos componentes de peso ligero (aquellos componentes que no son contenedores) separados por una línea divisoria. Esta línea divisoria permite especificar al usuario la cantidad de área que pertenece a cada componente; para ello, el usuario tan sólo tendrá que arrastrar dicha línea divisoria.

Este objeto es útil cuando se quiere mostrar dos componentes que contienen información relacionada, de manera que el usuario pueda cambiar el tamaño relativo de dichos componentes. Cuanto más tamaño se le dé a un componente, menor será el área de visualización del otro componente.

La división puede ser horizontal o vertical, según se indique en el constructor o en el método **setOrientation()**, las constantes **VERTICAL_SPLIT** u **HORIZONTAL_SPLIT**.

La llamada al método **setOneTouchExpandable(true)** hace que el JSplitPane muestre controles que permitan al usuario ocultar uno o bs dos componentes y asignar todo el espacio al otro componente.

Hay que señalar que un Split Pane no permite que el usuario haga que un componente sea más pequeño que su tamaño mínimo. Un ejemplo de Split Pane se muestra a continuación:



- **Métodos constructores**

Método	Descripción
JSplitPane() **JSplitPane(int)** **JSplitPane(int, boolean)** **JSplitPane(int, Component, Component)** **JSplitPane(int, boolean, Component, Component)**	Crea un SplitPane. Cuando existe, el parámetro **int** indica la orientación del SplitPane, **HORIZONTAL_SPLIT** o **VERTICAL_SPLIT**. El parámetro booleano selecciona si los componentes se redibujan contínuamente cuando el usuario arrastra el divisor. Los parámetros **Component** seleccionan los componentes izquierdo y derecho o superior e inferior, respectivamente.

- **Principales métodos**

Método	Descripción
void setOrientation(int) **int getOrientation( )**	El primero selecciona y el segundo obtiene la orientación del SplitPane. Se utilizan **HORIZONTAL_SPLIT** o **VERTICAL_SPLIT** definidas en **JSplitPane**.
void setDividerSize(int) **int getDividerSize()**	Selecciona u obtiene el tamaño de la línea divisorar en pixeles.
void setContinuousLayout(boolean) **boolean getContinuousLayout()**	Selecciona u obtiene si los componetes del SplitPane se redistribuyen y redibujan mientras el usuario arrastra la línea divisora.
void setOneTouchExpandable(boolean) **boolean getOneTouchExpandable()**	Selecciona u obtiene si el SplitPane muestra los botones de control expandible, para ocultar uno de los dos componentes.
void setTopComponent(Component) **void setBottomComponent(Component)** **void setLeftComponent(Component)** **void setRightComponent(Component)** **Component getTopComponent()** **Component getBottomComponent()** **Component getLeftComponent()** **Component getRightComponent()**	Selecciona u obtiene el componente indicado: el de arriba, el del fondo, el de la izquierda o el de la derecha respectivamente.
void remove(Component) **void removeAll()**	Elimina el componente indicado del SplitPane.
void add(Component)	Añade el componete al SplitPane. sólo podemos añadir dos componentes a un SplitPane.
void setDividerLocation(double)	Selecciona u obtiene la posición actual de la

void setDividerLocation(int) **int getDividerLocation()**	línea divisora. Cuando se selecciona la posición, podemos especificar un porcentaje (double) o una posición de pixel (int).
void setLastDividerLocation(int) **int getLastDividerLocation()**	Selecciona u obtiene la posición anterior de la línea divisora.
int getMaximumDividerLocation() **int getMinimumDividerLocation()**	Obtienen las posiciones mínima y máxima de la linea divisora.

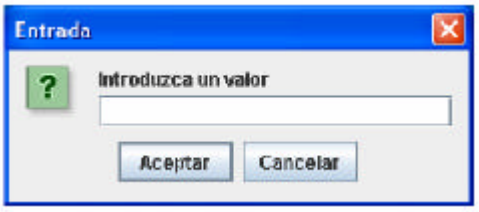
JOptionPane: E/S de datos con GUIS

La biblioteca de clases estándar de Java incluye una amplia gama de componentes para la construcción de interfaces gráficas de usuario. El componente JOptionPane se puede emplear para obtener datos de entrada y mostrar mensajes de salida.

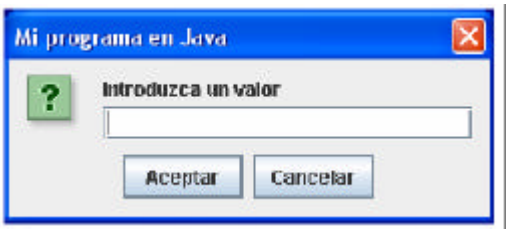
Entrada de Datos.

La entrada de datos mediante este componente se realiza mediante el comando showInputDiaglog con diferentes parámetros se consiguen distintos tipos de paneles:

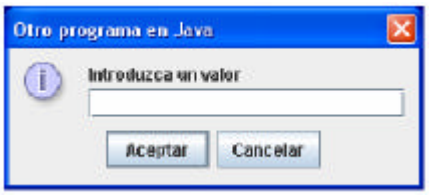
- JOptionPan.showInputDialog("Introduce un valor");



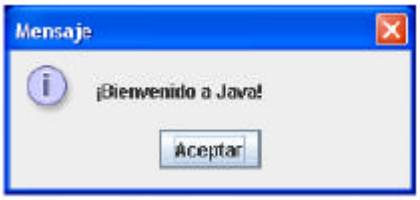
- JOptionPane.showInputDialog(null,"Introduzca un valor","Mi programa en Java",JOptionPane.**QUESTION_MESSAGE**);



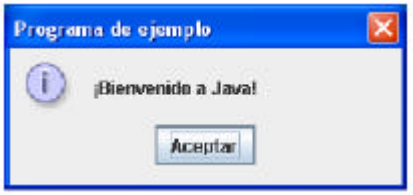
- JOptionPane.showInputDialog(null,"Introduzca un valor","Otro programa en Java", JOptionPane.**INFORMATION_MESSAGE**);



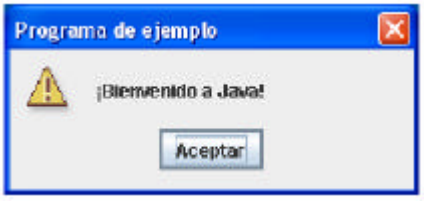
- JOptionPane.showMessageDialog(null,"¡Bienvenido a Java!");



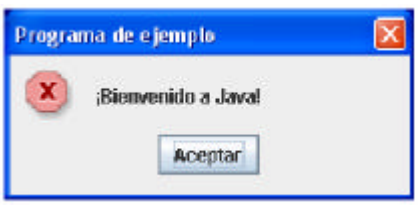
- JOptionPane.showMessageDialog(null,"¡Bienvenido a Java!","Programa de ejemplo", JOptionPane.**INFORMATION_MESSAGE**);



- JOptionPane.showMessageDialog(null,"¡Bienvenido a Java!","Programa de ejemplo", JOptionPane.**WARNING_MESSSAGE**);



- JOptionPane.showMessageDialog(null,"¡Bienvenido a Java!","Programa de ejemplo", JOptionPane.**ERROR_MESSAGE**);



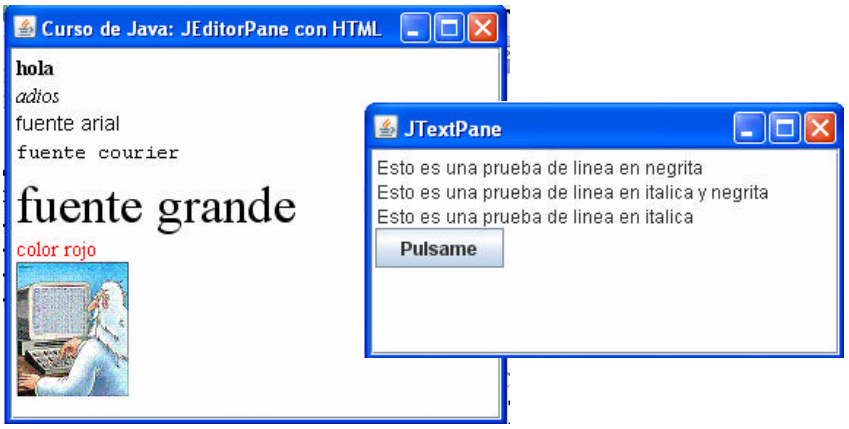
```
import javax.swing.JOptionPane;

public class Hipoteca  {

    public static void main (String args[])  {
        double cantidad; // en euros
        double interes; // en porcentaje (anual)
        int tiempo; // en años
        double cuota; // en euros (mensual)
        double interesMensual; // en tanto por uno
        String entrada; // variable auxiliar

        // Entrada de datos
        entrada = JOptionPane.showInputDialog("Importe de la hipoteca (€)");
        cantidad = Double.parseDouble(entrada);
        entrada = JOptionPane.showInputDialog("Tipo de interés (%)");
        interes = Double.parseDouble(entrada);
        entrada = JOptionPane.showInputDialog("Período de amortización (años)");
        tiempo = Integer.parseInt(entrada);

        // Cálculo de la cuota mensual
        interesMensual = interes/(12*100);
        cuota = (cantidad*interesMensual)
        / (1.0–1.0/Math.pow(1+interesMensual,tiempo*12));
        cuota = Math.round(cuota*100)/100.0;

        // Resultado
        JOptionPane.showMessageDialog(null, "Cuota mensual = "+cuota+"€" );
        System.exit(0);
    }
}
```

Ejemplo 4.8: Entrada y Salida de datos mediante JOptionPane

JEditorPane y JTextPane

Java proporciona varios componentes visuales que permiten escribir y tratar un texto. El componente base para este grupo de objetos java es el JEditorPane y su componente principial JTextPane que permite más parametrización del texto que contiene como el color, la fuente o los tamaños de las letras e incluso permite hasta la inserción de iconos basándose en la clase Icon de Java u otro tipo de componentes Java como botones, etiquetas, etc…

Sin embargo, estos componentes no tienen métodos para ir añadiendo poco a poco el texto. Simplemente tienen un método **setText()** al que se pasa todo el texto de golpe. Para poder trabajar más cómodamente con estos objetos es preferible obtener el objeto **Document** asociado al componente, mediante el método **getDocument()** y trabajar con él ya que este objeto sí posee métodos para ir añadiendo texto poco a poco.



Ejemplos de ejecución de los componentes JEditorPane y JTextPane

```
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.WindowConstants;

public class ejemplo4_9 {

   public ejemplo4_9()  {
        // Preparamos la ventana de ejemplo
         JFrame v = new JFrame("Curso de Java: JEditorPane con HTML");
         JEditorPane editor = new JEditorPane();
         JScrollPane scroll = new JScrollPane(editor);
         v.getContentPane().add(scroll);
         v.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

         // Marcamos el editor para que use HTML
         editor.setContentType("text/html");

         // Insertamos un texto
         editor.setText(
           "<head><base href=\"file:c:/\"></head>"+
             "<b>hola</b><br>" + "<i>adios</i><br>" +
             "<font face=\"arial\">fuente arial</font><br>" +
             "<font face=\"courier\">fuente courier</font><br>" +
             "<font size='24'>fuente grande</font><br>" +
             "<font color='red'>color rojo</font><br>" +
             "<img src='viejo.gif'/>");

         System.out.println(editor.getText());
         // Se visualiza la ventana
         v.pack();
         v.setVisible(true);
   }

   public static void main(String[] args){
      new ejemplo4_9();
   }
}
```

Ejemplo 4.9: Generación de un JEditorPane

Laboratorio:

El siguiente código muestra un ejemplo de construcción de una interfaz. Esta vez se han utilizado componentes no vistos todavía pero que se verán en los próximos temas como es el componente JButton que dibuja un botón o bien el GridLayout que ayuda a la distribución de componentes en un panel. Sirva este laboratorio para ir familiarizándonos con ellos.

```
// Uso de un objeto JPanel para ayudar a distribuir los componentes.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Lab4_1 extends JFrame
{
  private JPanel panelBotones;
  private JButton botones[];

  // configurar GUI
  public Lab4_1()
  {
    super( "Curso de Java: Demostración JPanel" );

    // obtener panel de contenido
    Container contenedor = getContentPane();

    // crear arreglo botones
    botones = new JButton[ 5 ];

    // configurar panel y establecer su esquema
    panelBotones = new JPanel();
    panelBotones.setLayout( new GridLayout( 1, botones.length ) );

    // crear y agregar botones
    for ( int cuenta = 0; cuenta < botones.length; cuenta++ ) {
      botones[ cuenta ] = new JButton( "Botón " + ( cuenta + 1 ) );
      panelBotones.add( botones[ cuenta ] );
    }

    contenedor.add( panelBotones, BorderLayout.SOUTH );

    setSize( 425, 150 );
    setVisible( true );

  } // fin del constructor de DemoPanel

  public static void main( String args[] )
  {
    Lab4_1 aplicacion = new Lab4_1();
    aplicacion.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }

} // fin de la clase DemoPanel
```

Ejercicios:

- Utilizando los componentes vistos en este tema desarrollar un programa para comprobar si un año es bisiesto o no: Un año es bisiesto si es divisible por 4 pero no por 100, o bien es divisible por 400.

- Desarrollar una aplicación que utilizando los paneles de interacción con el usuario vistos en este tema construya una estructura de paneles siguiendo las peticiones del usuario. La aplicación preguntará al usuario si desea añadir un nuevo panel o borrarlo y en función de la respuesta que se obtenga se creará un nuevo panel o se borrará uno existente. La aplicación deberá también mostrar un mensaje de error cuando se intente borrar un panel no habiendo paneles creados.