Apuntes

de

Java

Tema 7:
AWT

TEMA 7: PROGRAMACIÓN GRÁFICA EN JAVA CON AWT

7.1.- INTRODUCCIÓN

La *interfaz de usuario* es la parte del programa que permite a éste interactuar con el usuario. Las interfaces de usuario pueden adoptar muchas formas, que van desde la simple línea de comandos hasta las interfaces gráficas (IGU-GUI) que proporcionan las aplicaciones más modernas.

La interfaz de usuario es uno de los aspectos 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 de usuario a través de la biblioteca de clases **AWT**, y opciones para mejorarlas mediante una nueva biblioteca denominada **Swing**.

Debido a que el lenguaje de programación Java es **independiente de la plataforma** en que se ejecuten sus aplicaciones, la biblioteca AWT también es independiente de la plataforma en que se ejecute. El AWT proporciona un conjunto de herramientas para la construcción de interfaces gráficas que tienen una apariencia y se comportan de forma semejante en todas las plataformas en que se ejecute.

Los elementos de la interfaz proporcionados por la biblioteca AWT están implementados utilizando ***toolkits*** nativos de las plataformas, preservando una apariencia semejante a todas las aplicaciones que se creen para esa plataforma. Este es un punto fuerte del AWT, pero también tiene la desventaja de que un interfaz gráfico diseñado para una plataforma, puede no visualizarse correctamente en otra diferente. Estas carencias del AWT son subsanadas en parte por **Swing**, y en general por las **JFC (Java Foundation Classes).**

7.2.- AWT (ABSTRACT WINDOW TOOLKIT)

AWT es el acrónimo del Abstract Window Toolkit para Java. Se trata de una biblioteca de clases Java para el desarrollo de las Interfaces de Usuario Gráficas. La versión del AWT que *Sun* proporciona con el JDK 1.0.x se desarrolló en sólo dos meses y 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 novedosos. Quizá la presión de tener que lanzar algo al mercado haya tenido mucho que ver en la pobreza de AWT en la versión 1.0.x.

Javasoft, en vista de la precariedad de que hace gala el AWT, y para asegurarse que los elementos que desarrolla para generar interfaces gráficas sean fácilmente transportables entre plataformas, se ha unido con Netscape, IBM y Lighthouse Design para crear un conjunto de clases que proporcionen una sensación visual agradable y sean más fáciles de utilizar por el programador. Esta colección de clases son las ***Java Foundation Classes*** (**JFC**), que están constituidas por cinco

grupos de clases, al menos en este momento: AWT, Java 2D, Accesibilidad, Arrastrar y Soltar y Swing.

- **AWT** engloba a todos los componentes del AWT que existían en la versión 1.1.2 del JDK y en los que se han incorporado en versiones posteriores.
- **Java 2D** es un conjunto de clases gráficas bajo licencia de IBM/Taligent, que todavía está en construcción.
- **Accesibilidad**, proporciona clases para facilitar el uso de ordenadores y tecnología informática a disminuidos, tiene lupas de pantalla, y cosas así.
- **Arrastrar y Soltar** (*Drag and Drop*), son clases en las que se soporta Glasgow, que es la nueva generación de los JavaBeans.
- **Swing**, es la parte más importante y la que más desarrollada se encuentra. Ha sido creada en conjunción con Netscape y 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, y algunos de ellos participan de la arquitectura MVC (*Modelo-Vista-Controlador*), aunque **Swing** también proporciona otros widgets nuevos como árboles, pestañas, etc.

Se recomienda encarecidamente a la hora de programar el uso de Swing en lugar de AWT ya que ésta última versión soluciona múltiples problemas existentes en la versión predecesora (AWT). Sin embargo, dado que a la hora de programar las diferencias entre Swing y AWT son prácticamente nulas, en este tema seguiremos basándonos en la librería AWT para dar todas las explicaciones (dado que fue este el modelo de programación visual que surgió originalmente). Posteriormente (dentro de la explicaciones de clase) se indicarán las modificaciones que hay que hacer sobre una programa en AWT para convertirlo a un programa en Swing.

7.3.- ESTRUCTURA BÁSICA DE AWT

La estructura básica del AWT se basa en:

- Componentes
- Contenedores

Los *Componentes* son los controles básicos como botones, listas, cuadros de texto, checkbox,...

Los *Contenedores* contienen a otros *Componentes* dentro, los cuales de encuentran posicionados de forma relativa con respecto al contenedor. No se usan posiciones fijas de los Componentes con respecto a los Contenedores, sino que están situados a través de una disposición controlada (*layouts*). Los Contenedores son a su vez también Componentes.

La interacción con el usuario y el sistema se realiza por medio de eventos. Así el manejo de eventos se encarga de tratar la interacción del usuario con el ratón, menús, etc. Los eventos pueden tratarse tanto en Contenedores como en Componentes, corriendo por cuenta del

programador la seguridad de tratamiento de los eventos adecuados. La gestión de eventos se trata en el Tema 8.

Con Swing se va un paso más allá, ya que todos los *JComponentes* son subclases de **Container**, lo que hace posible que widgets Swing puedan contener otros componentes, tanto de AWT como de Swing.

Componentes y Contenedores

La interfaz gráfica de usuario (IGU) está construida en base a elementos gráficos básicos denominados ***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. En la biblioteca AWT, todos los Componentes de la interfaz gráfica de usuario son instancias de la clase ***Component*** o de una clase descendiente de ella.

Los Componentes no se encuentran aislados, sino agrupados dentro de *Contenedores*. Los ***Contenedores*** contienen y organizan la situación de los Componentes. Los Contenedores son en sí mismos Componentes y como tales pueden ser situados dentro de otros Contenedores. En la biblioteca AWT, todos los Contenedores son instancias de la clase ***Container*** o una clase descendiente de ella.

7.4.- PRIMEROS PASOS CON AWT: CREACIÓN DE FRAMES

Un *frame* (marco) es una ventana que no está contenida dentro de otra ventana. Es un contenedor (*container*).

Los contenedores pueden almacenar otros elementos del interfaz de usuario denominados componentes *(Components*), por ejemplo botones o campos de texto.

Ejemplo 1: Primer Frame

Construcción de un Frame en blanco.

Figura 7.1: Frame en blanco

Código:

```
import java.awt.*;
/* Esta clase hereda de la clase Frame de Java, es la
* clase que va a representar un Frame.   */
public class MiFrame extends Frame
{
      public MiFrame()
      {
           setSize(300, 200);
           setTitle("PrimerFrame");
        }

        public static void main (String[] args)
        {
           // Creamos una instancia de esta clase
           MiFrame f =  new  MiFrame();
           // Mostramos el frame
           f.show();
        }
}
```

Jerarquía de la clase *Frame* dentro de la AWT.

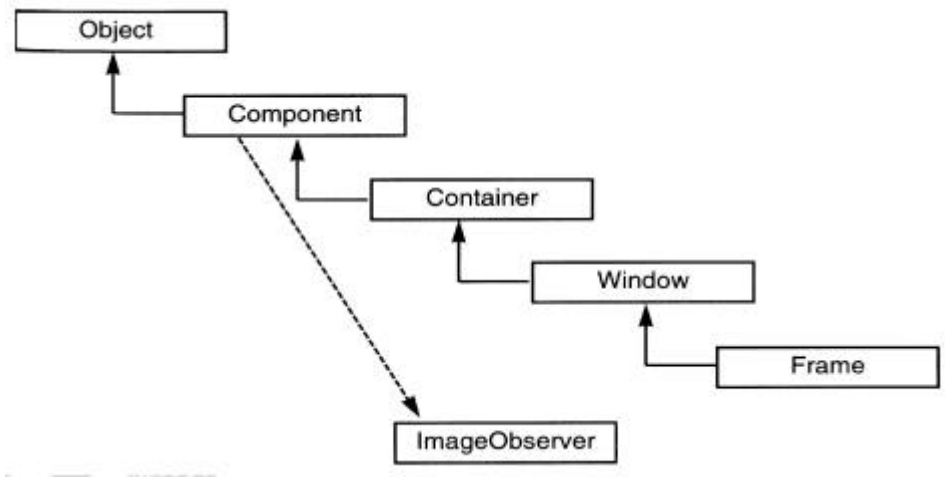


Figura 7.2: Jerarquía de la clase Frame

7.5.- COMPONENTES

Component es una clase abstracta que representa todo lo que tiene una posición, un tamaño, puede ser pintado en pantalla y puede recibir eventos.

Los Objetos derivados de la clase **Component** que se incluyen en el Abstract Window Toolkit son los que aparecen a continuación:

- Button
- Canvas
- Checkbox
- Choice
- Container
    - Panel
    - Window
        - Dialog
        - Frame
- Label
- List
- Scrollbar
- TextComponent
    - TextArea
    - TextField

Vamos a ver un poco más en detalle los Componentes que nos proporciona el AWT para incorporar a la creación de la interface con el usuario.

7.5.1.- Botones

Veremos ejemplos de cómo se añaden botones a un panel para la interacción del usuario con la aplicación, pero antes vamos a ver la creación de botones como objetos.

Se pueden crear objetos Button con el operador **new**:

```
Button boton;
boton = new Button( "Botón");
```

La cadena utilizada en la creación del botón aparecerá en el botón cuando se visualice en pantalla. Esta cadena también se devolverá para utilizarla como identificación del botón cuando ocurra un evento.

Botones de Pulsación

Los botones presentados en la ventana son los botones de pulsación estándar; no obstante, para variar la representación en pantalla y para conseguir una interfaz más limpia, AWT ofrece a los programadores otros tipos de botones.

Botones de Lista

Los botones de selección en una lista (*Choice*) permiten el rápido acceso a una lista de elementos. Por ejemplo, podríamos implementar una selección de colores y mantenerla en un botón Choice:

Figura 7.3: Lista desplegable

Código:

```
import java.awt.*;

public class PanelBotonSeleccion extends Panel{
     Choice Selector;

     public PanelBotonSeleccion(){
          Selector = new Choice();

          Selector.addItem("Rojo");
          Selector.addItem("Verde");
          Selector.addItem("Azul");

          add(Selector);
     }

     public static void main(String[] args){
          // No intentar comprender este código por el momento
          Frame f = new Frame();
          f.add(new PanelBotonSeleccion());
          f.show();
     }
}
```

Botones de Marcación

Los botones de marcación (*Checkbox*) se utilizan frecuentemente como botones de estado. Proporcionan información del tipo *Sí* o *No* (true o false).

Figura 7.4: Botón de Selección Múltiple (CheckBox)

Código:

```
import java.awt.*;

public class PanelBotonComprobacion extends Panel{
     Checkbox Relleno;

     public PanelBotonComprobacion(){
          Relleno = new Checkbox ("Relleno");

          add(Relleno);
     }

     public static void main(String[] args){
          // No intentar comprender este código por el momento
          Frame f = new Frame();
          f.add(new PanelBotonComprobacion());
          f.show();
     }
}
```

Botones de Selección

Los botones de selección se pueden agrupar para formar una interfaz de botón de radio (*CheckboxGroup*), que son agrupaciones de botones Checkbox en las que siempre hay un único botón activo.

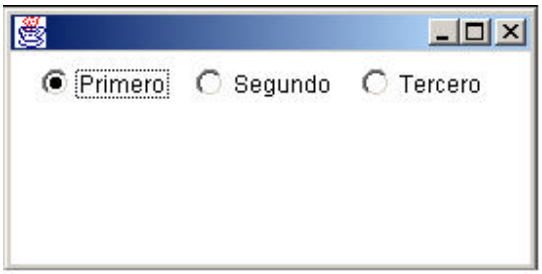


Figura 7.5: Botones de selección exclusiva

Código:

```
import java.awt.*;

public class PanelBotonRadio extends Panel{
     CheckboxGroup Radio;

     public PanelBotonRadio(){
          Radio = new CheckboxGroup();
```

```
            add( new Checkbox( "Primero", Radio, true) );
            add( new Checkbox( "Segundo", Radio, false) );
            add( new Checkbox( "Tercero", Radio, false) );
      }

      public static void main(String[] args){
            // No intentar comprender este código por el momento
            Frame f = new Frame();
            f.add(new PanelBotonRadio());
            f.show();
      }
}
```

En el ejemplo anterior, observamos que siempre hay un botón activo entre los que conforman el interfaz de comprobación que se ha implementado.

7.5.2.- Etiquetas

Las etiquetas (*Label*) proporcionan una forma de colocar texto estático en un panel, para mostrar información que no varía, normalmente, al usuario.

Este ejemplo presenta dos textos en pantalla, tal como aparece en la figura siguiente:

Figura 7.6: Dos etiquetas

Código:

```
import java.awt.*;

public class PanelEtiquetas extends Panel{

      public PanelEtiquetas(){
            Label etiq1 = new Label( "Hola java!" );
            Label etiq2 = new Label( "Otra etiqueta" );

            add(etiq1);
            add(etiq2);
      }

      public static void main(String[] args){
            // No intentar comprender este código por el momento
            Frame f = new Frame();
```

```
            f.add(new PanelEtiquetas());
            f.show();
        }
    }
```

7.5.3.- Listas

Las listas (*List*) aparecen en los interfaces de usuario para facilitar a los operadores la manipulación de muchos elementos. Se crean utilizando métodos similares a los de los botones Choice. La lista es visible todo el tiempo, utilizándose una barra de desplazamiento para visualizar los elementos que no caben en el área que aparece en la pantalla.

El ejemplo siguiente, crea una lista que muestra cuatro líneas a la vez y no permite selección múltiple.

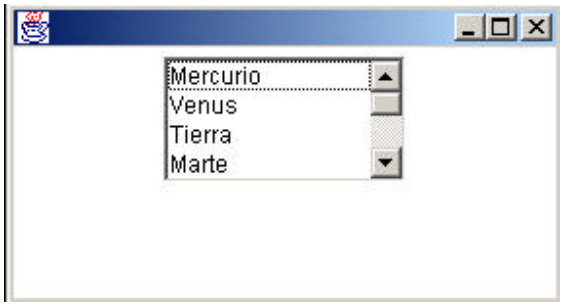


Figura 7.7: Lista de Selección Exclusiva

Código:

```
import java.awt.*;

public class PanelLista extends Panel{

    public PanelLista(){
        List l = new List( 4, false );

        l.addItem( "Mercurio" );
        l.addItem( "Venus" );
        l.addItem( "Tierra" );
        l.addItem( "Marte" );
        l.addItem( "Jupiter" );
        l.addItem( "Saturno" );
        l.addItem( "Neptuno" );
        l.addItem( "Urano" );
        l.addItem( "Plutón" );

        add( l );
    }

    public static void main(String[] args){
```

```
            // No intentar comprender este código por el momento
            Frame f = new Frame();
            f.add(new PanelLista());
            f.show();
        }
    }
```

En la aplicación siguiente, se permite al usuario seleccionar varios elementos de los que constituyen la lista. En la figura se muestra la apariencia de una selección múltiple.

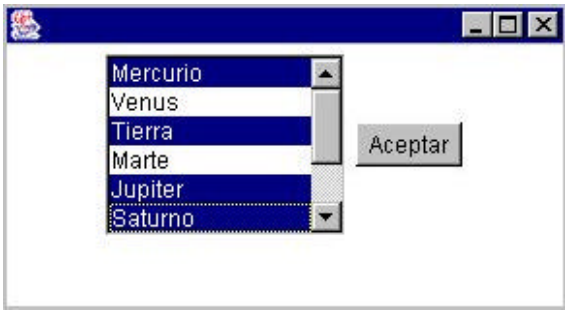


Figura 7.8: Lista de Selección Múltiple

Código:

```
import java.awt.*;

public class PanelListaMult extends Panel{

    List lm = new List(6,true);

    public PanelListaMult(){

        Button boton = new Button("Aceptar");

        lm.addItem("Mercurio");
        lm.addItem("Venus");
        lm.addItem("Tierra");
        lm.addItem("Marte");
        lm.addItem("Jupiter");
        lm.addItem("Saturno");
        lm.addItem("Neptuno");
        lm.addItem("Urano");
        lm.addItem("Pluton");

        add(lm);
        add(boton);
    }

    public static void main(String[] args){
        // No intentar comprender este código por el momento
```

```
            Frame f = new Frame();
            f.add(new PanelListaMult());
            f.show();
        }
}
```

7.5.4.- Campos de Texto

Para la entrada directa de datos se suelen utilizar los campos de texto, que aparecen en pantalla como pequeñas cajas que permiten al usuario la entrada por teclado.

Figura 7.9: Campos de Texto

Los campos de texto (*TextField*) se pueden crear vacíos, vacíos con una longitud determinada, rellenos con texto predefinido y rellenos con texto predefinido y una longitud determinada. Este ejemplo genera cuatro campos de texto con cada una de las características anteriores. La imagen muestra los distintos tipos de campos.

Código:

```
import java.awt.*;

public class PanelCamposTexto extends Panel {

    TextField tf1,tf2,tf3,tf4;

    public PanelCamposTexto(){

        //Campo de texto vacío
        tf1 = new TextField();
        //Campo de texto vacío con 20 columnas
        tf2 = new TextField( 20 );
        //Texto predefinido
        tf3= new TextField( "Hola" );
        //Texto predefinido en 30 columnas
        tf4= new TextField( "Hola", 30);

        add (tf1);
        add (tf2);
```

```
            add (tf3);
            add (tf4);
        }

        public static void main(String[] args){
            // No intentar comprender este código por el momento
            Frame f = new Frame();
            f.add(new PanelCamposTexto());
            f.show();
        }
    }
```

7.5.5.- Áreas de Texto

Java, a través del AWT, permite incorporar texto multilínea dentro de zonas de texto (*TextArea*). Los objetos TextArea se utilizan para elementos de texto que ocupan más de una línea, como puede ser la presentación tanto de texto editable como de sólo lectura.

Para crear un área de texto se pueden utilizar cuatro formas análogas a las que se han descrito en la creación de Campos de Texto. Pero además, para las áreas de texto hay que especificar el número de columnas.

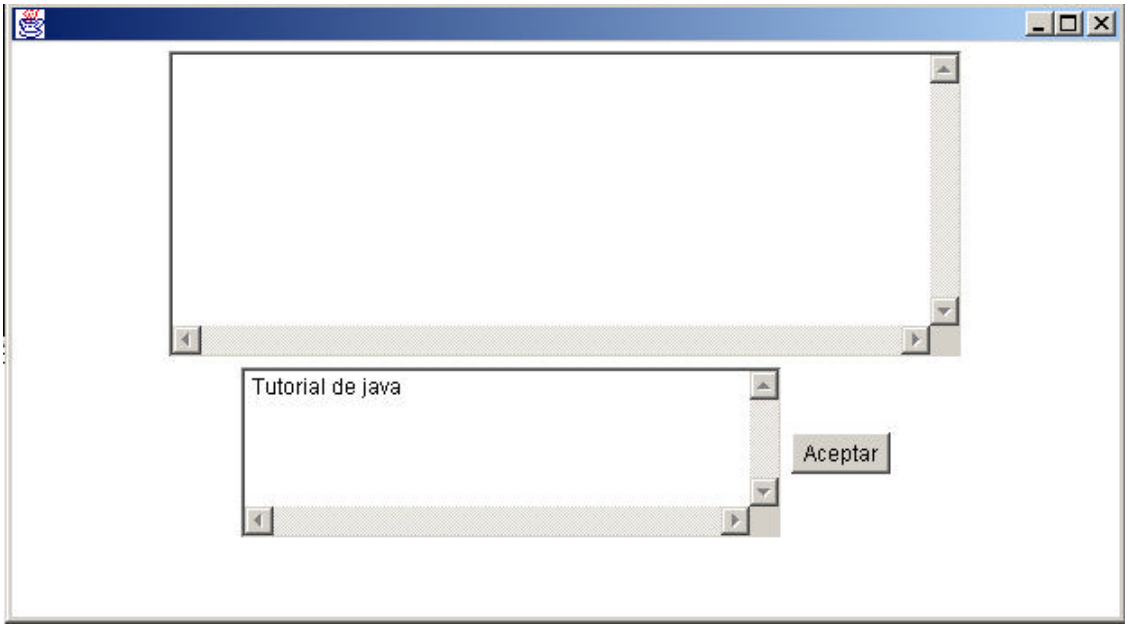


Figura 7.10: Dos Área de Texto

Se puede permitir que el usuario edite el texto con el método *setEditable()* de la misma forma que se hacía en el TextField. En la figura aparece la epresentación de AreaTexto.java, que presenta dos áreas de texto, una vacía y editable y otra con un texto predefinido y no editable.

Código:

```
import java.awt.*;

public class PanelAreaTexto extends Panel{
     TextArea t1,t2;

     public PanelAreaTexto(){
          Button boton = new Button ( "Aceptar" );
          t1 = new TextArea();
          t2 = new TextArea( "Tutorial de java", 5 , 40 );
          t2.setEditable(false);

          add( t1 );
          add( t2 );
          add( boton );
     }

     public static void main(String[] args){
          // No intentar comprender este código por el momento
          Frame f = new Frame();
          f.add(new PanelAreaTexto());
          f.show();
     }
}
```

Para acceder al texto actual de una zona de texto se utiliza el método *getText()*.

7.6.- CREACION DE APLICACIONES CON AWT

Para crear una aplicación utilizando AWT, vamos a ver en principio cómo podemos generar el interface de esa aplicación, mostrando los distintos elementos del AWT, posteriormente miraremos hacia la implementación de la funcionalidad de la aplicación.

Interface:

- Crear el Marco de la aplicación (Frame)
- Inicializar Fuentes, Colores, Layouts y demás recursos
- Crear menús y Barras de Menús
- Crear los controles, diálogos, ventanas, etc.

7.7.- CREAR EL MARCO DE LA APLICACION

El Contenedor de los Componentes es el ***Frame***. También es la ventana principal de la aplicación, lo que hace que para cambiar el icono o el cursor de la aplicación no sea necesario crear métodos

nativos; al ser la ventana un Frame, ya contiene el entorno básico para la funcionalidad de la ventana principal.

Vamos a empezar a crear una aplicación básica, a la que iremos incorporando Componentes. Quizás vayamos un poco deprisa en las explicaciones que siguen; no preocuparse, ya que lo que no quede claro ahora, lo estará más tarde. El problema es que para poder profundizar sobre algunos aspectos de Java, necesitamos conocer otros previos, así que proporcionaremos un ligero conocimiento sobre algunas características de Java y del AWT, para que nos permitan entrar a fondo en otras; y ya conociendo estas últimas, volveremos sobre las primeras. Un poco lioso, pero imprescindible.

Comenzaremos el desarrollo de nuestra aplicación básica con AWT a partir del código que mostramos a continuación:

```
import java.awt.*;

public class AplicacionAWT extends Frame {

     static final int HOR_TAMANO = 300;
     static final int VER_TAMANO = 200;

     public AplicacionAWT() {
          super( "Aplicación Java con AWT" );

          pack();
          resize( HOR_TAMANO, VER_TAMANO );
          show();
     }

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

La clase anterior es un **Frame**, ya que extiende esa clase y hereda todas sus características. Tiene un método, el constructor, que no admite parámetros.

Además, se hace una llamada explícita al constructor *super*, es decir, al constructor de la clase padre, para pasarle como parámetro la cadena que figurará como el título de la ventana.

La llamada a *show()* es necesaria, ya que por defecto, los Contenedores del AWT se crean ocultos y hay que mostrarlos explícitamente. La llamada a *pack()* hace que los Componentes se ajusten a sus tamaños correctos en función del Contenedor en que se encuentren situados.

La ejecución de la aplicación mostrará la siguiente ventana en pantalla:

Figura 7.11: Aspecto de una ventana (Frame)

Los atributos fundamentales de la ventana anterior son:

- Marco de 300x200 pixels
- No tiene barra de menú
- No tiene ningún Componente
- Título "Aplicación Java con AWT"
- Color de fondo por defecto
- *Layout* por defecto
- Fondo de la ventana vacío

7.8.- INICIALIZAR FUENTES, COLORES Y RECURSOS

Vamos a ir alterando los recursos de la ventana de la aplicación Java que estamos desarrollando con el AWT, para ir incorporando y visualizando los distintos Componentes que proporciona AWT. Insertemos algunas líneas de código en el constructor para inicializar la aplicación:

- Cambiemos el font de caracteres a Times Roman de 12 puntos

```
setFont( new Font( "TimesRoman",Font.PLAIN,12 ) );
```

- Fijemos los colores de la ventana para que el fondo sea Blanco y el texto resalte en Negro

```
setBackground( Color.white );
setForeground( Color.black );
```

- Seleccionemos como disposición de los Componentes el *BorderLayout* para este Contenedor

```
setLayout( new BorderLayout() );
```

- Incorporemos gráficos. Usamos métodos definidos en la clase Graphics; por ejemplo, reproduzcamos el título de la ventana en medio con una fuente Time Roman de 24 puntos en color Azul. Es necesario utilizar *new* con Font ya que en Java, todo son objetos y no podríamos utilizar un nuevo font de caracteres sin antes haberlo creado

```
public void paint( Graphics g ) {
     g.setFont( new Font( "TimesRoman",Font.BOLD,24 ) );
     g.setColor( Color.blue );
     g.drawString( getTitle(),30,50 );
}
```

- Incorporemos en la parte inferior de la ventana dos botones: *Aceptar* y *Cancelar*

```
Panel p = new Panel();
p.add( new Button( "Aceptar" ) );
p.add( new Button( "Cancelar" ) );
add( "South", p );
```

Los Componentes se incorporan al Contenedor a través de los dos métodos *add()* que hay definidos:

```
add( Component c );
add( String s, Component c );
```

Los Componentes también se podían haber insertado en el Frame, organizándolos en una cierta forma, teniendo en cuenta que su layout por defecto (el de los Frame) es un BorderLayout. Por ejemplo:

```
add( "South",new Button( "Aceptar ) );
add( "South",new Button( "Cancelar" ) );
```

Hemos utilizado un Panel y no el segundo método, porque es más útil el organizar los Componentes en pequeñas secciones. Así, con nuestro código podemos considerar al Panel como una entidad separada del Frame, lo cual permitiría cambiar el fondo, layout, fuente, etc. del Panel sin necesidad de tocar el Frame.

Si ejecutamos de nuevo la aplicación con los cambios que hemos introducido, aparecerá ante nosotros la ventana que se muestra a continuación:

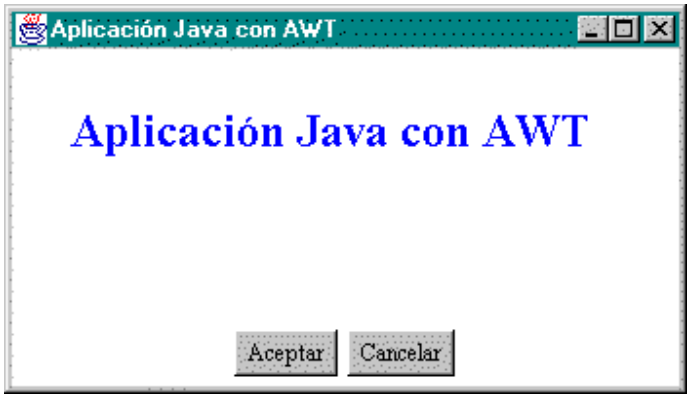


Figura 7.12: Ventana con dos botones y un texto

Si intentásemos en esta aplicación cerrar la ventana, no sucede nada. Cuando se intenta cerrar la ventana, el sistema envía un evento que no se está tratando. El tema de eventos se explicará en el tema 8.

7.9.- DIALOGOS Y VENTANAS

Una Ventana genérica, *Window*, se usa fundamentalmente como clase base de las clases *Frame* y *Dialog* (éstas heredan de ella parte de su funcionalidad). Algunos métodos que pueden resultar interesantes de la clase Window y que pueden ser empleados por las ventanas (Frame) y los dialogos (Dialog) son:

- getToolkit()
- getWarningString()
- pack()
- toBack()
- toFront()

El funcionamiento de una *ventana* (Frame) ya lo hemos visto brevemente en apartados anteriores (podríamos decir que es la ventana "normal" de las aplicaciones Java).

Un *Diálogo* es una subclase de Window, que puede tener un borde y ser modal, es decir, no permite hacer nada al usuario hasta que responda al diálogo. Esto es lo que se usa en las cajas de diálogo "Acerca de...", en la selección en listas, cuando se pide una entrada numérica, etc.

El código Java que se expone a continuación, implementa el diálogo *Acerca de* para la aplicación. Esta clase se crea oculta y necesitaremos llamar al método *show()* de la propia clase para hacerla visible.

```
class AboutDialog extends Dialog {
      static int HOR_TAMANO = 300;
      static int VER_TAMANO = 150;

      public AboutDialog( Frame parent ) {
           super( parent,"Acerca de...",true );
           this.setResizable( false );
           setBackground( Color.gray );
           setLayout( new BorderLayout() );

           Panel p = new Panel();
           p.add( new Button( "Aceptar" ) );
           add( "South", p );
           resize( HOR_TAMANO,VER_TAMANO );
      }

      public void paint( Graphics g ) {
           g.setColor( Color.white );
           g.drawString("Aplicación Java con AWT",HOR_TAMANO/4,VER_TAMANO/3);
           g.drawString( "Versión 1.00", HOR_TAMANO/3+15, VER_TAMANO/3+20 );
      }
```

La ventana que aparece en pantalla generada por la clase anterior es la que muestra la figura:

Figura 7.13: Ventana de tipo Diálogo

No hay razón aparente para que la ventana del ejemplo sea una subclase de la clase **Frame** (con ser una subclase de Dialog es más que suficiente), pero si se quiere proporcionar funcionalidad extra, sí debería serlo. Esto es así porque Frame implementa la interface MenuContainer, con lo cual tiene la posibilidad de proporcionar menús y cambiar el cursor, el icono de la aplicación, etc. (funcionalidad que no posee la clase Dialog).

Un EJEMPLO MÁS COMPLICADO de aplicación gráfica basada en el AWT es el convertidor de decimal a binario/octal/hexadecimal/base36, cuya presentación en pantalla es la que muestra la figura siguiente.

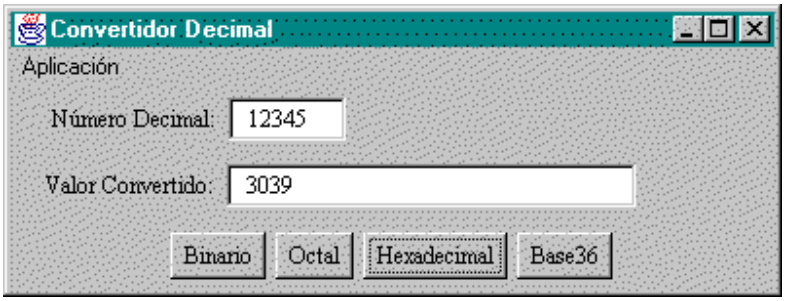


Figura 7.14: Ventana para un convertidos decimal

Código:

```
import java.awt.*;

// Clase que nos permite introducir un número y nos lo presenta en
// diferentes bases
public class Convertidor extends Frame {
     int valorDecimal = 0;
     String valorX = new String( "0" );
     TextField dTexto, xTexto;

     public Convertidor() {
          super( "Convertidor Decimal" );
```

```
// Creamos la barra de menú, los botones de las bases y los
// paneles que vamos a utilizar para posicionar los
// Componentes
MenuBar menub = new MenuBar();
Button Bin = new Button( "Binario" );
Button Octal = new Button( "Octal" );
Button Hexa = new Button( "Hexadecimal" );
Button Base36 = new Button( "Base36" );
Panel p1 = new Panel();
Panel p2 = new Panel();
Panel p3 = new Panel();

// Creamos el menú desplegable
Menu menu = new Menu( "Aplicación" );
menu.add( new CheckboxMenuItem( "Base36 Activa" ) );
menu.add( new MenuItem( "Salir" ) );
menub.add( menu );
setMenuBar( menub );

// Incorporamos los botones a uno de los paneles
p3.setLayout( new FlowLayout() );
p3.add( Bin );
p3.add( Octal );
p3.add( Hexa );
p3.add( Base36 );

// Creamos los dos campos de texto que vamos a utilizar , uno
// para la introducción del número decimal que queremos
// convertir y el otro para presentar el número convertido
// Y asociamos cada uno de los campos, junto con su etiqueta
// explicativa a uno de los paneles
Label dEtiq = new Label( " Número Decimal:" );
Label xEtiq = new Label( "Valor Convertido:" );
dTexto = new TextField( Integer.toString( valorDecimal ),7 );
xTexto = new TextField( valorX,32 );
p1.setLayout( new FlowLayout( FlowLayout.LEFT ) );
p2.setLayout( new FlowLayout( FlowLayout.LEFT ) );
p1.add( dEtiq );
p1.add( dTexto );
p2.add( xEtiq );
p2.add( xTexto );

// Incorporamos los paneles a la ventana
add( "North",p1 );
add( "Center",p2 );
add( "South",p3 );

resize( 400,150 );
show();
```

```
        }

        public static void main( String args[] ) {
            Convertidor c = new Convertidor();
        }
    }
```

En el tema 8 se explicará el tema de eventos, con lo que seremos capaces de dotar de comportamiento a la aplicación.

7.10.- PANELES

La clase **Panel** es el más simple de los Contenedores de Componentes gráficos.

El uso de Paneles permite que las aplicaciones puedan utilizar múltiples *layouts*, es decir, que la disposición de los componentes sobre la ventana de visualización pueda modificarse con mucha flexibilidad. Permite que cada Contenedor pueda tener su propio esquema de fuentes de caracteres, color de fondo, zona de diálogo, etc.

Podemos, por ejemplo, crear una barra de herramientas para la zona superior de la ventana de la aplicación o incorporarle una zona de estado en la zona inferior de la ventana para mostrar información útil al usuario. Para ello vamos a implementar dos Paneles:

```
class BarraHerram extends Panel {
    public BarraHerram() {
        setLayout( new FlowLayout() );
        add( new Button( "Abrir" ) );
        add( new Button( "Guardar" ) );
        add( new Button( "Cerrar" ) );

        Choice c = new Choice();
        c.addItem( "Times Roman" );
        c.addItem( "Helvetica" );
        c.addItem( "System" );
        add( c );
        add( new Button( "Ayuda" ) );
    }
}

class BarraEstado extends Panel {
    Label texto;
    Label mas_texto;

    public BarraEstado() {
        setLayout( new FlowLayout() );
        add( texto = new Label( "Creada la barra de estado" ) );
        add( mas_texto = new Label( "Información adicional" ) );
    }
```

```
        public void verEstado( String informacion ) {
             texto.setText( informacion );
        }
    }
```

Ahora deberemos crear los objetos correspondientes a la barra de herramientas y a la barra de estado y añadirlos a la ventana principal (Frame) de nuestra aplicación. Así dentro del código de la ventana principal deberían incluirse las siguientes sentencias:

```
add( "North", tb = new BarraHerram() );
add( "South", sb = new BarraEstado() );
```

Al final, la apariencia de la aplicación en pantalla es la que presenta la figura anterior.

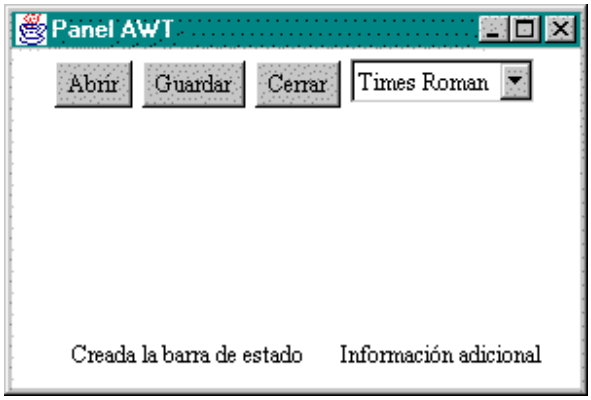


Figura 7.15: Ventana con varios paneles

7.11.- LAYOUTS

Los *layout managers* o *manejadores de composición*, en traducción literal, permiten controlar la disposición de los diversos componentes dentro de un contenedor. Es decir, especifican la distribución que tendrán los Componentes a la hora de colocarlos sobre un Contenedor. Java dispone de varios, en la actual versión, tal como se muestra en la imagen:

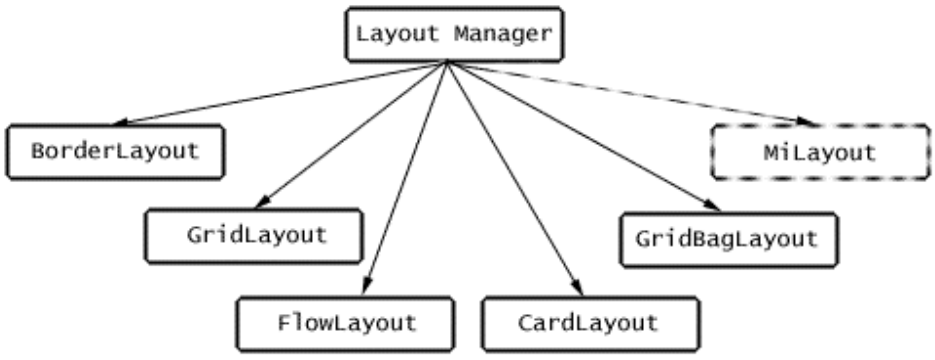


Figura 7.16: Clasificación de Layouts

¿Por qué Java proporciona estos esquemas predefinidos de disposición de componentes? La razón es simple: imaginemos que deseamos agrupar objetos de distinto tamaño en celdas de una rejilla virtual: si confiados en nuestro conocimiento de un sistema gráfico determinado, y codificamos a mano tal disposición, deberemos prever el redimensionamiento de la ventana, su repintado cuando sea cubierto por otra ventana, etc., además de todas las cuestiones relacionadas con un posible cambio de plataforma (uno nunca sabe a donde van a ir a parar nuestras ventanas).

Sigamos imaginando, ahora, que un hábil equipo de desarrollo ha previsto las disposiciones gráficas más usadas y ha creado un gestor para cada una de tales configuraciones, que se ocupará, de forma transparente para nosotros, de todas esas cuitas de formatos. Bien, pues estos gestores son instancias de las distintas clases derivadas de Layout Manager y que se utilizan en los contenedores de nuestras aplicaciones.

Los Layouts liberan al programador de tener que preocuparse de dónde ubicar cada uno de los componentes cuando una ventana es redimensionada o cuando una ventana es refrescada o cuando una ventana es llevada a una plataforma que maneja un sistema de coordenadas diferente.

FlowLayout

Es el más simple y el que se utiliza por defecto en todos los Paneles si no se fuerza el uso de alguno de los otros. Los Componentes añadidos a un Panel con FlowLayout se encadenan en forma de lista. La cadena es horizontal, de izquierda a derecha, y se puede seleccionar el espaciado entre cada Componente.

Figura 7.17: Ejemplo de FlowLayout

Por ejemplo, podemos poner un grupo de botones con la composición por defecto que proporciona FlowLayout:

Código:

```
import java.awt.*;

public class PanelAwtFlow extends Panel{

     Button boton1,boton2,boton3;

     public PanelAwtFlow(){
          boton1 = new Button ( "Aceptar" );
```

```
            boton2 = new Button ("Salir");
            boton3 = new Button ("Cerrar");

            add(boton);
            add(boton2);
            add(boton3);
      }

      public static void main(String[] args){
            // No intentar comprender este código por el momento
            Frame f = new Frame();
            f.add(new PanelAwtFlow());
            f.show();
      }
}
```

Este código, construye tres botones con un pequeño espacio de separación entre ellos. No se especifica ningún layout porque se usa el layout por defecto de los paneles que es el FlowLayout.

BorderLayout

La composición BorderLayout (de borde) proporciona un esquema más complejo de colocación de los Componentes en un panel. La composición utiliza cinco zonas para colocar los Componentes sobre ellas: Norte, Sur, Este, Oeste y Centro. Es el layout o composición que se utilizan por defecto Frame y Dialog.

El Norte ocupa la parte superior del panel, el Este ocupa el lado derecho, Sur la zona inferior y Oeste el lado izquierdo. Centro representa el resto que queda, una vez que se hayan rellenado las otras cuatro partes.

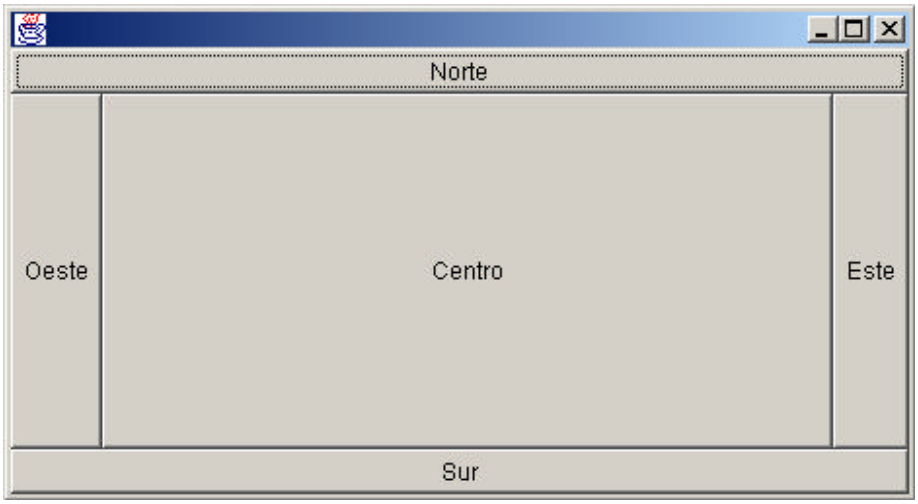


Figura 7.18: Ejemplo BorderLayout

Con BorderLayout se podrían representar botones de dirección:

Código:

```
import java.awt.*;

public class PanelAwtBord extends Panel{
     Button botonN,botonS,botonE,botonO,botonC;

     public PanelAwtBord(){

          setLayout (new BorderLayout() );

          botonN = new Button ( "Norte" );
          botonS = new Button ("Sur");
          botonE = new Button ("Este");
          botonO = new Button ( "Oeste" );
          botonC = new Button ( "Centro" );

          add( "North", botonN );
          add( "South", botonS );
          add( "East", botonE);
          add( "West", botonO);
          add( "Center", botonC);
     }

     public static void main(String[] args){
          // No intentar comprender este código por el momento
          Frame f = new Frame();
          f.add(new PanelAwtBord());
          f.show();
     }
}
```

Como se puede observar en el código, la manera de asignar un determinado layout a un contenedor es mediante el uso del método *setLayout ()* invocado sobre el propio contenedor:

```
<Objeto_Contenedor>.setLayout(<Objeto_Layout>)
```

Una vez asignado el layout al contenedor, la manera de añadir elementos a ese contenedor será mediante el método *add()* y en este caso indicando dónde queremos que se sitúe el elemento (North, South, East, West, Center).

GridLayout

La composición GridLayout proporciona gran flexibilidad para situar Componentes. El layout se crea con un número de filas y columnas y los Componentes van dentro de las celdas de la tabla así definida.

En la figura siguiente se muestra un panel que usa este tipo de composición para posicionar seis botones en su interior, con tres filas y dos columnas que crearán las seis celdas necesarias para albergar los botones:

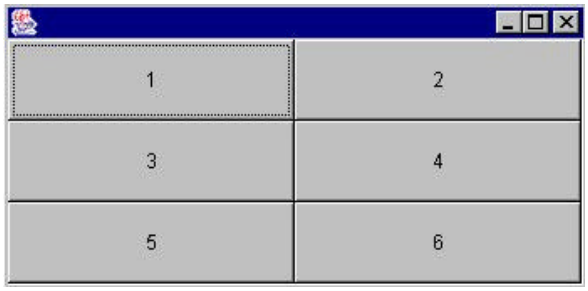


Figura 7.19: Ejemplo GridLayout

Código:

```
import java.awt.*;

public class PanelAwtGrid extends Panel{

     Button boton1,boton2,boton3,boton4,boton5,boton6;

     public PanelAwtGrid(){

          setLayout (new GridLayout( 3,2 ) );

          boton1 = new Button ( "1" );
          boton2 = new Button ("2");
          boton3 = new Button ("3");
          boton4 = new Button ( "4" );
          boton5 = new Button ( "5" );
          boton6 = new Button ("6");

          add( boton1 );
          add( boton2 );
          add( boton3 );
          add( boton4 );
          add( boton5 );
          add( boton6 );
     }

     public static void main(String[] args){
          // No intentar comprender este código por el momento
          Frame f = new Frame();
          f.add(new PanelAwtGrid());
          f.show();

     }
}
```

CardLayout

Este es el tipo de composición que se utiliza cuando se necesita una zona de la ventana que permita colocar distintos Componentes en cada momento. Este layout suele ir asociado con botones de lista (Choice), de tal modo que cada selección determina el panel (grupo de componentes) que se presentarán.

En la figura siguiente mostramos el efecto de la selección sobre la apariencia de la ventana que contiene el panel con la composición CardLayout:

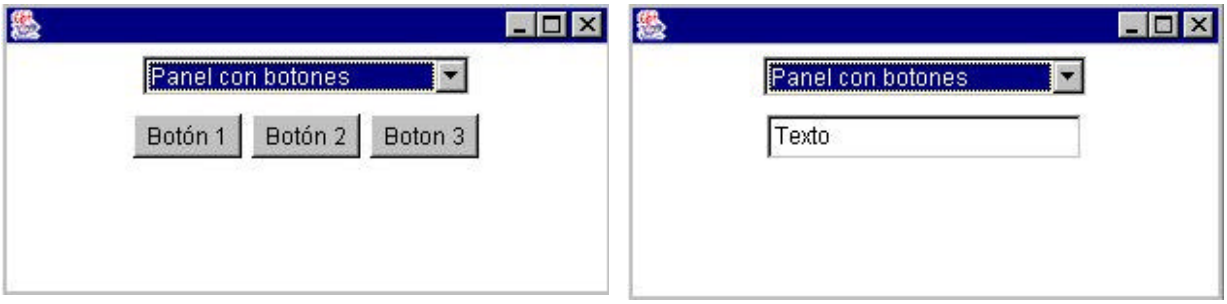


Figura 7.20: Ejemplo CardLayout (CardBotones y CardTexto)

Código:

```
import java.awt.*;

public class PanelAwtCard extends Panel{
  Panel card;
  final static String PanelBoton = "Panel con botones";
  final static String PanelTexto = "Panel con campo de texto";


    public PanelAwtCard(){

        setLayout (new BorderLayout() );

        Panel ac = new Panel();
        Choice c = new Choice();

        c.addItem(PanelBoton);
        c.addItem(PanelTexto);
        ac.add(c);
        add("North", ac);

        card = new Panel();
        card.setLayout( new CardLayout() );

        Panel p1 = new Panel();
        p1.add( new Button ("Botón 1") );
        p1.add( new Button ("Botón 2") );
```

```
            p1.add( new Button ("Boton 3") );
            Panel p2 = new Panel();
            p2.add( new TextField( "Texto", 20) );

            card.add( PanelBoton, p1 );
            card.add( PanelTexto, p2);
            add( "Center", card );

            // Para conmutar entre un Card y el otro
            new CardLayout().show(card, PanelAwtCard.PanelTe xto);
            new CardLayout().show(card, PanelAwtCard.PanelBoton);
      }

      public static void main(String[] args){
            // No intentar comprender este código por el momento
            Frame f = new Frame();
            f.add(new PanelAwtCard());
            f.show();
      }
}
```

Crear un Layout Personalizado

Se puede crear un Layout personalizado en base a la interface *LayoutManager*. Hay que redefinir los cinco métodos que utiliza este interface, lo cual puede no resultar sencillo, así que en lo posible se deben utilizar los métodos de colocación de componentes que proporciona AWT.

No obstante, vamos a implementar un layout propio para poder colocar los Componentes en posiciones absolutas del panel que contenga a este layout. Hacemos que nuestro nuevo layout implemente el interface LayoutManager e implementamos los cinco métodos de este interface para que podamos posicionar los Componentes.

```
import java.awt.*;

public class MiLayout implements LayoutManager {

      public MiLayout() {
      }

      public void addLayoutComponent( String name,Component comp  ) {
      }

      public void removeLayoutComponent( Component comp ) {
      }

      public Dimension preferredLayoutSize( Container parent ) {
            Insets insets = parent.insets();
            int numero = parent.countComponents();
            int ancho = 0;
            int alto = 0;
```

```
            for( int i=0; i < numero; i++ )
            {
                Component comp = parent.getComponent( i );
                Dimension d = comp.preferredSize();
                Point p = comp.location();
                if( ( p.x + d.width ) > ancho )
                    ancho = p.x + d.width;
                if( ( p.y + d.height ) > alto )
                    alto = p.y + d.height;
            }
            return new Dimension(insets.left + insets.right + ancho,
                                 insets.top + insets.bottom + alto );
        }

        public Dimension minimumLayoutSize( Container parent ) {
            Insets insets = parent.insets();
            int numero = parent.countComponents();
            int ancho = 0;
            int alto = 0;

            for( int i=0; i < numero; i++ )
            {
                Component comp = parent.getComponent( i );
                Dimension d = comp.preferredSize();
                Point p = comp.location();
                if( ( p.x + d.width ) > ancho )
                    ancho = p.x + d.width;
                if( ( p.y + d.height ) > alto )
                    alto = p.y + d.height;
            }
            return new Dimension(insets.left + insets.right + ancho,
                                 insets.top + insets.bottom + alto );
        }

        public void layoutContainer( Container parent ) {
            int numero = parent.countComponents();

            for( int i=0; i < numero; i++ )
            {
                Component comp = parent.getComponent( i );
                Dimension d = comp.preferredSize();
                comp.resize( d.width,d.height );
            }
        }
    }
```

Y ahora vamos a ver un ejemplo en que utilicemos nuestro Layout. Posicionaremos tres botones en el panel y un campo de texto con una etiqueta precediéndolo. La apariencia que tendrá en pantalla será la que se muestra en la figura:

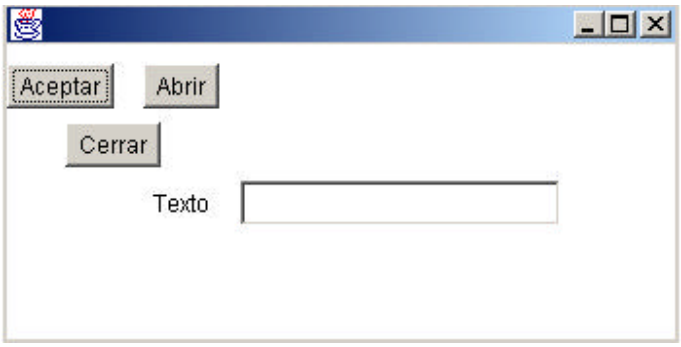


Figura 7.21: Ejemplo de un Layout personalizado

El código es el siguiente:

```
import java.awt.*;

public class PanelAwtLibre extends Panel{

     Button boton1,boton2,boton3;
     Label etiqueta;
     TextField texto;

     public PanelAwtLibre(){

          setLayout (new MiLayout() );

          boton1 = new Button ("Aceptar");
          boton2 = new Button ("Abrir");
          boton3 = new Button ("Cerrar");
          etiqueta = new Label ("Texto");
          texto = new TextField ("",20 );

          add (boton1);
          add (boton2);
          add (boton3);
          add (etiqueta);
          add (texto);

          boton1.move(0,10);
          boton2.move(70,10);
          boton3.move(30,40);
          etiqueta.move(75,70);
          texto.move(120, 70);
     }

     public static void main(String[] args){
          // No intentar comprender este código por el momento
          Frame f = new Frame();
```

```
            f.add(new PanelAwtLibre());
            f.show();
        }
    }
```

7.11.1.- Un Ejemplo Completo

En la ventana de la figura siguiente, se utilizan los diferentes tipos de layouts que proporciona el paquete AWT.

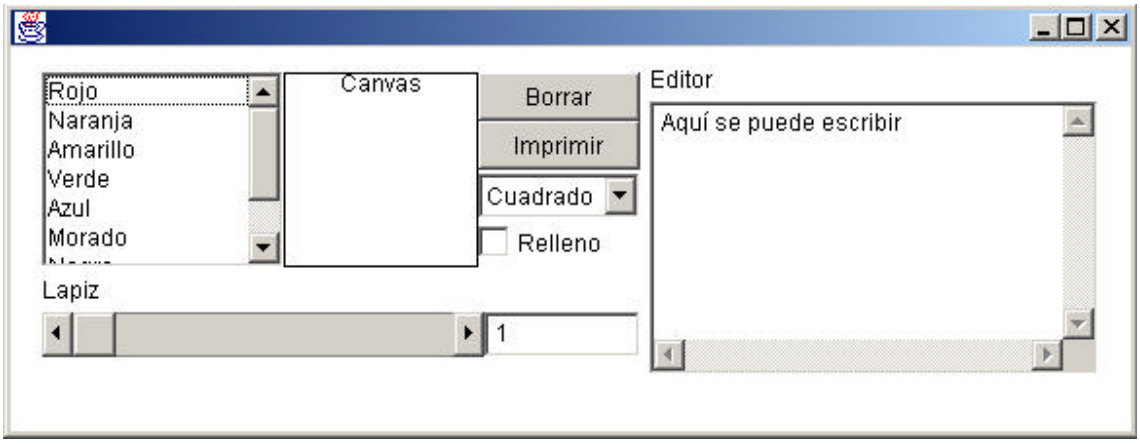


Figura 7.22: Ventana para la ejecución de un ejemplo completo

Código:

```
import java.awt.*;
import java.io.*;

//clase para poder pintar casi todos los componentes que ofrece el AWT
//de Java y poder visualizar su apariencia en la pantalla
public class PanelAwtGui extends Panel{

     //Creamos una clase para poder pintar una zona de dibujo y que
     //se muestre el canvas. Pintamos un rectangulo alrededor suyo
     class miCanvas extends Canvas {
          public void paint ( Graphics g ) {
               int w = size().width;
               int h = size().height;
               g.drawRect (0,0,w-1,h-1);
               g.drawString(   "Canvas", (w-g.getFontMetrics().stringWidth(
               "Canvas"))/2,10);
          }
     }

     TextArea edicion;
     miCanvas dibujo;
     Label edicionLab, lapizLab;
```

```
List colores;
Button imprimir, borrar;
Choice figuras;
Checkbox relleno;
Scrollbar lapizBar;
TextField lapizTex;
Panel panelIzq, panelDch, panelBot, panelDib;

public PanelAwtGui(){

     //Creamos los paneles con sus layout managers
     panelIzq = new Panel();
     panelIzq.setLayout( new BorderLayout() );

     panelDch = new Panel();
     panelDch.setLayout( new BorderLayout() );

     panelBot = new Panel();
     panelBot.setLayout( new GridLayout(4,0) );

     panelDib = new Panel();
     panelDib.setLayout( new BorderLayout() );
     panelDib.reshape(1,1,200,20);

     //construimos el lado izquierdo de la ventana
     //creamos la lista de colores
     colores = new List (6, false);
     colores.addItem("Rojo");
     colores.addItem("Naranja");
     colores.addItem("Amarillo");
     colores.addItem("Verde");
     colores.addItem("Azul");
     colores.addItem("Morado");
     colores.addItem("Negro");
     colores.addItem("Blanco");

     //Añadimos la lista de colores al panel izquierdo
     panelIzq.add("West", colores);

     //creamos un nuevo Canvas
     dibujo = new miCanvas();
     dibujo.reshape(0,0,100,100);
     //Añadimos el canvas al panel izquierdo
     panelIzq.add("Center", dibujo);

     //creamos los botones
     borrar = new Button ("Borrar");
     imprimir = new Button ("Imprimir");
     figuras = new Choice();
     figuras.addItem("Cuadrado");
```

```
            figuras.addItem("Circulo");
            figuras.addItem("Triángulo");
            relleno = new Checkbox ("Relleno");
            //Añadimos los botones a su propio panel
            panelBot.add(borrar);
            panelBot.add(imprimir);
            panelBot.add(figuras);
            panelBot.add(relleno);
            //añadimos el panel de botones al lado derecho
            panelIzq.add("East", panelBot);

            //Creamos el area del lapiz
            lapizLab = new Label ("Lapiz");
            lapizBar = new Scrollbar (Scrollbar.HORIZONTAL,1,1,1,10);
            lapizBar.reshape(1,1,100,5);
            lapizTex = new TextField ("1",8);
            //añadimos las partes anteriores a su propio panel
            panelDib.add("North",lapizLab);
            panelDib.add("Center", lapizBar);
            panelDib.add("East", lapizTex);
            //añadimos el panel a la parte inferior
            panelIzq.add("South", panelDib);

            // construimos el lado derecho de la ventana
            edicionLab = new Label ("Editor");
            edicion = new TextArea ("Aquí se puede escribir", 8, 30);
            //añadimos la etiqueta y el área de texto al lado derecho
            panelDch.add("North", edicionLab);
            panelDch.add("South", edicion);
            //incorporamos los dos paneles al panel principal
            add (panelIzq);
            add (panelDch);
      }

      public static void main(String[] args){
            // No intentar comprender este código por el momento
            Frame f = new Frame();
            f.add(new PanelAwtGui());
            f.show();
      }
}
```

El ejemplo ilustra el uso de paneles, listas, barras de desplazamiento, botones, selectores, campos de texto, áreas de texto y varios tipos de layouts.

En el tratamiento de los Layouts se utiliza un *método de validación*, de forma que los Componentes son marcados como *no válidos* cuando un cambio de estado afecta a la disposición de éstos (por ejemplo, debido a un redimensionado de la ventana). Esta validación se realiza mediante los métodos *validate()* e *invalidate()*. A su vez, los métodos *pack()* y *show()* se encargan

de la actualización y distribución correcta de todos los elementos que así lo requieran (los que en la validación hayan sido marcados como *no validos*).

7.12.- APÉNDICE I: OTROS EJEMPLOS SENCILLOS

Frame que se cierra

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

// Esta clase hereda de la clase Frame de Java, es la
// clase que va a representar un Frame que termina l a
// aplicación cuando el usuario lo cierra.
public class FrameQueSeCierra  extends  Frame
{
     public FrameQueSeCierra()
     {
          addWindowListener(
               new  WindowAdapter()
               {
                    public void  windowClosing(WindowEvent e)
                    {
                         System.exit(0);
                    }
               }
          );
          setSize(300, 200);
          setTitle(getClass().getName());
       }
}
```

Segundo Frame

El Frame anterior aparecía siempre en la esquina superior izquierda. Vamos a mejorar ese frame para hacer que aparezca en cualquier posición y a cualquier tamaño.

```
/* Esta clase va a mostrar un Frame centrado en la pantalla.   */
import java.awt.*;

// Hereda la funcionalidad de la clase FrameQueSeCierra
public class FrameCentrado extends FrameQueSeCierra
{
     public FrameCentrado()
     {
          Toolkit tk = Toolkit.getDefaultToolkit();
          Dimension d = tk.getScreenSize();
          int resolucionPantallaAlto = d.height;
          int resolucionPantallaAncho = d.width;
          setSize(resolucionPantallaAncho / 2, resolucionPantallaAlto/ 2);
          setLocation( resolucionPantallaAncho/ 4,resolucionPantallaAlto / 4);
       }
```

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

Explicación del código:

```
Toolkit tk = Toolkit.getDefaultToolkit();
Dimension d = tk.getScreenSize();
```

La clase *Toolkit* de AWT:

- permite interactuar directamente con el sistema operativo obteniendo información dependiente del sistema.
- Es la superclase de todas las clases de AWT
- Tiene un método *getScreenSize* que devuelve la resolución de la pantalla como un objeto de la clase *Dimension*. La clase *Dimension* de AWT encapsula dos variables alto y ancho (*height*, *width*).

```
setSize(resolucionPantallaAncho / 2, resolucionPantallaAlto/ 2);
setLocation( resolucionPantallaAncho/ 4, resolucio nPantallaAlto / 4);
```

La clase *Component* de AWT tiene los métodos que cambian el tamaño del componente al valor especificado en los parámetros

```
void setSize(int width, int height)
void setSize(Dimension d)
```

Los siguientes métodos mueve el componente a la nueva posición

```
void setLocation (int x, int y)
void setLocation (Point p)
```

Escribiendo algo en una ventana

Vamos a construir un Frame que muestre un mensaje en una ventana, por ejemplo: Hola a todos.

Código:

```
import java.awt.*;
public class HolaATodos extends FrameQueSeCierra
{
     public void paint(Graphics g)
     {
          g.drawString("Hola a todos", 75, 100);
     }
     public static void main(String[] args)
     {
          Frame f = new HolaATodos();
          f.show();
     }
}
```

Explicación del código:

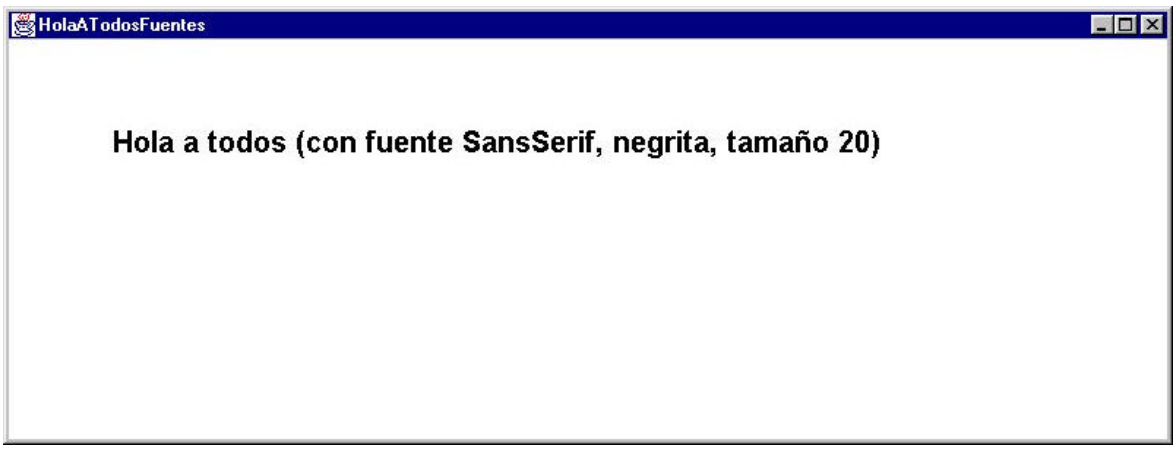
Para dibujar sobre un frame se hereda y se redefine el método *paint* de la clase *Container*. Este método tiene un parámetro que es un objeto de la clase *Graphics* de AWT. El objeto de la clase *Graphics* es similar al contexto de dispositivo programando en Windows o al contexto gráfico en X11 (Unix).

Todos los dibujos y textos en Java se realizan sobre objetos *Graphics*. Las operaciones sobre objetos *Graphics* se realizan en pixels. Para escribir texto la clase *Graphics* utiliza el método:

```
drawString( String s, int xCoord, int yCoord );
```

Utilizando fuentes

Construir un frame que muestre el mensaje anterior, pero ahora definiendo la fuente, el tipo y el tamaño de la letra.



Código:

```
import java.awt.*;
public class HolaATodosFuentes extends FrameQueSeCierra
{
       public HolaATodosFuentes()
       {
            Toolkit tk = Toolkit.getDefaultToolkit();
            Dimension d = tk.getScreenSize();
            int resolucionPantallaAlto = d.height;
            int resolucionPantallaAncho = d.width;
            setSize(resolucionPantallaAncho, resolucionPantallaAlto/ 2);
            setLocation( 0, resolucionPantallaAlto / 4);
        }
        public void paint(Graphics g)
        {
             Font f = new Font("SansSerif", Font.BOLD, 20);
             g.setFont(f);
             g.drawString("Hola a todos (con fuente SansSerif,  negrita,
                tama&ntilde;o 20)", 75, 100);
        }
        public static void main(String[] args)
        {
            Frame f = new HolaATodosFuentes();
            f.show();
        }
}
```

Explicación del código:

La clase *Font* de AWT produce las fuentes. Su constructor también puede combinar estilos de la siguiente forma:

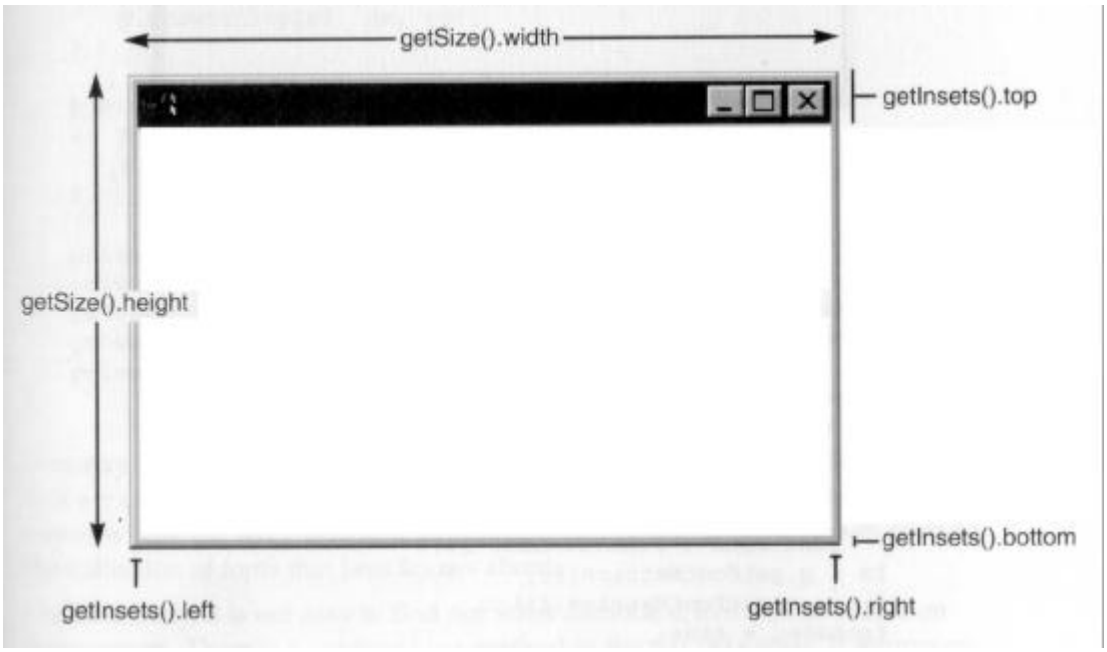
```
Font f = new Font("SansSerif", Font.BOLD+ Font.ITALIC, 20);
```

Para realizar operaciones de medida de lo que ocupan las fuentes se utiliza la clase *FontMetrics* de AWT. Así, por ejemplo, el método *stringWidth* de la clase *FontMetrics* toma una cadena y devuelve su ancho en pixels:

```
int stringWidth( String str );
```

El tamaño exacto de un Frame

La clase *Component* que es padre de la clase *Frame* tiene el método *getSize()* que devuelve el tamaño del Frame. Sin embargo *getSize()* devuelve el tamaño de la ventana incluyendo los bordes y la barra de título.



El método *getInsets()* devuelve las medidas de los bordes y la barra de título de la ventana.

La altura útil de un frame es:

```
getSize().height - getInsets().top -getInsets().bottom
// Es decir, a la altura completa de un frame , le quitamos
// la anchura del borde superior y del inferior.
```

La anchura útil es:

```
getSize().width - getInsets().left -getInsets().right
// Al ancho total del frame le quitamos la anchura de los
// bordes izquierdo y derecho.
```

La clase *Component*, padre de Frame, tiene los métodos getSize y getInsets.

Utilizando varias fuentes

Vamos a modificar el frame anterior para que muestre un mensaje utilizando varias fuentes. Además, vamos a manejar tamaños exactos de fuentes y ventanas, a parte de situar el mensaje en el centro de la ventana, haciendo que aparezca siempre en el centro, aunque se modifique el tamaño del frame.

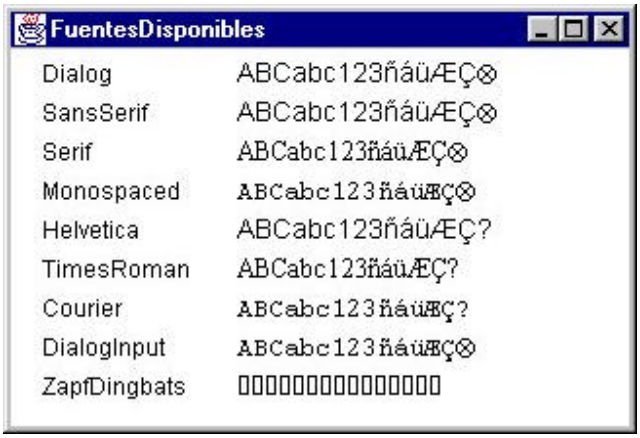
Código:

```
import java.awt.*;
public class HolaATodosVariasFuentes extends FrameQueSeCierra
{
       private Font f;
       private Font fi;
       private FontMetrics fm;
       private FontMetrics fim;
       private boolean fuentesPuestas = false;

       public HolaATodosVariasFuentes()
      {
            Toolkit tk = Toolkit.getDefaultToolkit();
            Dimension d = tk.getScreenSize();
            int resolucionPantallaAlto = d.height;
            int resolucionPantallaAncho = d.width;
            setSize(resolucionPantallaAncho/2, resolucionPantallaAlto/ 2);
            setLocation( resolucionPantallaAncho/4, resolucionPantallaAlto / 4);
        }
        public void ponFuentes(Graphics g)
        {
           if (fuentesPuestas)
                return;
```

```
            f = new Font("SansSerif", Font.BOLD, 14);
            fi = new Font("SansSerif", Font.BOLD + Font.ITALIC, 14);
            fm = g.getFontMetrics(f);
            fim = g.getFontMetrics(fi);
            fuentesPuestas = true;
         }
         public void paint(Graphics g)
         {
            ponFuentes(g);
            String s1 = "Hola a todos ";
            String s2 = "con varias fuentes";
            String s3 = " y centrado";
            int w1 = fm.stringWidth(s1);
            int w2 = fim.stringWidth(s2);
            int w3 = fm.stringWidth(s3);
            Dimension d = getSize();
            Insets in = getInsets();
            int clientWidth = d.width - in.right - in.left;
            int clientHeight = d.height - in.bottom - in.top;
            int cx = (clientWidth - w1 - w2 - w3) /  2 + in.left;
            int cy = clientHeight / 2 + in.top;
            g.drawRect(in.left, in.top, clientWidth - 1, clientHeight - 1);
            g.setFont(f);
            g.drawString(s1, cx, cy);
            cx += w1;
            g.setFont(fi);
            g.drawString(s2, cx, cy);
            cx += w2;
            g.setFont(f);
            g.drawString(s3, cx, cy);
         }
         public static void main(String[] args)
         {
            Frame f = new HolaATodosVariasFuentes();
            f.show();
         }
}
```

Fuentes disponibles

Programa que muestra las fuentes disponibles en el sistema para trabajar con Java.



Código:

```
import java.awt.*;
public class FuentesDisponibles extends FrameQueSeCierra
{
       public void paint(Graphics g)
        {
           String [] listaFuentes =  getToolkit().getFontList();
           Font unaFuente = g.getFont();
           for (int i = 0; i <listaFuentes.length; i++)
           {
                g.setFont(unaFuente);
                g.drawString(listaFuentes[i], 20,  i * 20 + 40);
                Font f = new Font(listaFuentes[i], Font.PLAIN, 14);
                g.setFont(f);
                g.drawString( "ABCabc123&ntilde;&aacute;&uuml;\u00C6\u00C7\u2297",
                120, i  *20 + 40);
           }
        }
        public static void main(String[] args)
        {
           Frame f = new FuentesDisponibles();
           f.show();
        }
}
```

Uso de Colores

Programa que muestra mensajes en distintos colores.

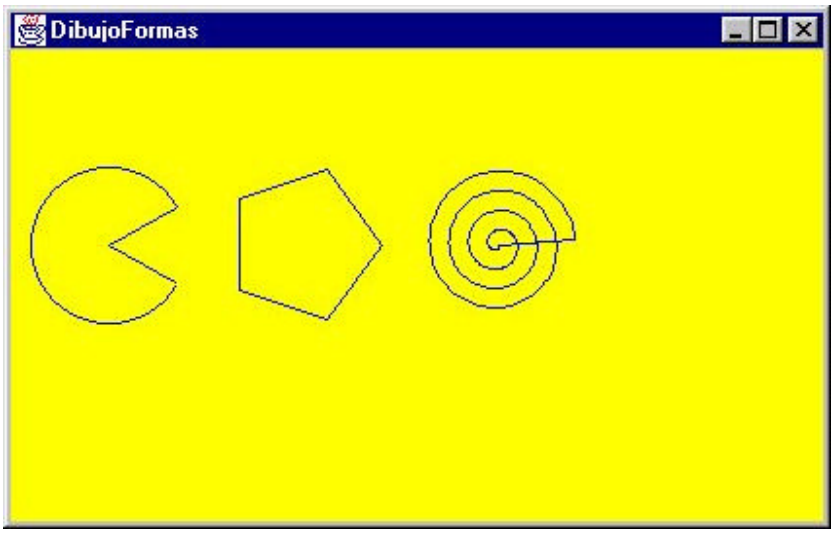
Código:

```
import java.awt.*;
public class HolaATodosColores extend s FrameQueSeCierra
{
     public HolaATodosColores()
     {
          Toolkit tk = Toolkit.getDefaultToolkit();
          Dimension d = tk.getScreenSize();
          int resolucionPantallaAlto = d.height;
          int resolucionPantallaAncho = d.width;
          setSize(resolucionPantallaAncho/2,  resolucionPantallaAlto/ 2);
          setLocation( resolucionPantallaAncho/4,  resolucionPantallaAlto / 4);
       }
       public void paint(Graphics g)
       {
          Font f =new Font("SansSerif", Font.BOLD, 20);
          g.setFont(f);

          g.setColor(Color.blue);
          g.drawString("Hola a todos en azul", 75, 100);

          g.setColor(new Color(0,128,128));
          g.drawString("Hola a todas en azul verdoso", 75, 125);

          Font f2 =new Font("TimesRoman", Font.BOLD, 20);
          g.setFont(f2);
```

```
            g.setColor(SystemColor.window);
            g.drawString("Colores del sistema", 75, 150);
        }
        public static void main(String[] args)
        {
            Frame f = new HolaATodosColores();
            f. setBackground(Color.yellow);
            f.show();
        }
}
```

Dibujando formas

Programa que dibuja distintas formas: un comecocos, un polígono (pentágono) y una espiral construida como polígono.



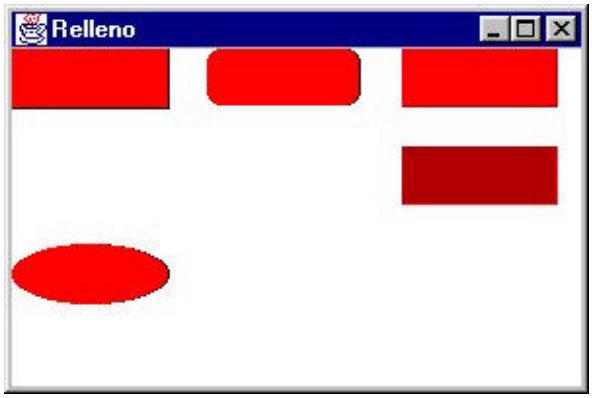
Código:

```
import java.awt.*;
public class DibujoFormas extends FrameQueSeCierra
{
        public DibujoFormas()
        {
            Toolkit tk = Toolkit.getDefaultToolkit();
            Dimension d = tk.getScreenSize();
            int resolucionPantallaAlto = d.height;
            int resolucionPantallaAncho = d.width;
            setSize(resolucionPantallaAncho/2,  resolucionPantallaAlto/ 2);
            setLocation( resolucionPant allaAncho/4,  resolucionPantallaAlto / 4);
        }
        public void paint(Graphics g)
        {
```

```
        g.setColor(Color.blue);
        // Situa el origen de coordenadas en la zona util del frame
        g.translate(getInsets().left,getInsets().top);

        // Dibuja un comecocos
        int r = 40;
        // radio del círculo del comecocos
        int cx = 50;
        // centro del círculo
        int cy = 100;
        int angulo = 30;
        //ángulo de la boca
        int dx = (int)(r * Math.cos(angulo * Math.PI /  180));
        int dy = (int)(r * Math.sin(angulo * Math.PI /  180));
        g.drawLine(cx, cy, cx + dx, cy + dy);
        // mandíbula inferior
        g.drawLine(cx, cy, cx + dx, cy  - dy);
        // mandíbula superior
        g.drawArc(cx - r, cy - r, 2 * r, 2 * r, angulo,  360  - 2 * angulo);

        // Dibuja un polígono: Pentágono
        Polygon p = new Polygon();
        //centro polígono
        cx = 150;
        cy = 100;
        int i;
        for (i = 0; i < 5; i++)
             p.addPoint( (int)(cx + r * Math.cos(i * 2 * Math.PI /  5)),
             (int)(cy + r * Math.sin(i * 2 * Math.PI / 5)));
        g.drawPolygon(p);

        //Espiral
        Polygon s = new Polygon();
        //centro espiral
        cx = 250;
        cy = 100;
        for (i = 0; i <360; i++)
        {
             double t = i / 360.0;
             s.addPoint( (int)(cx + r * t * Math.cos(8 *  t * Math.PI)),(int)(cy
             + r * t * Math.sin(8 * t * Math.PI)));
        }
        g.drawPolygon(s);
    }
    public static void main(String[] args)
    {
        Frame f = new DibujoFormas();
        f. setBackground(Color.yellow);
        f.show();
    }
}
```

Dibujando formas rellenas

Para dibujar formas rellenas, hay que utilizar los métodos de la clase *java.awt.Graphics* Estos métodos comienzan por *fill*, por ejemplo, *fillRect*, *fillRoundRect*, *fillOval*, *fillArc*, *fillPolygon*, etc.



Dado que este ejemplo es sencillo, se queda propuesto como ejercicio.

Manejando imágenes

Vamos a ver un ejemplo que muestra una imagen en formato jpg ó gif.

Código:

```
import java.awt.*;
import java.awt.image.*;
public class Foto extends FrameQueSeCierra
{
        private Image imagen;
        public Foto()
        {
           imagen = Toolkit.getDefaultToolkit().getImage ("../image18.jpg");
        }
        public void paint(Graphics g)
        {
```

```
            g.translate(getInsets().left, getInsets().top);
            g.drawImage(imagen, 0, 0, this);
        }
        public static void main(String args[])
        {
            Frame f = new Foto();
            f.show();
        }
}
```

7.13.- APÉNDICE II: EL PAQUETE AWT

En el árbol siguiente mostramos la relación que existe entre todas las clases que proporciona AWT para la creación de interfaces de usuario.

JERARQUÍA DE CLASE E INTERFACES DEL PAQUETE AWT

A continuación se muestra la jerarquía de clases e interfaces del paquete AWT:

Class Hierarchy

- class java.lang.**Object**
  - class java.awt.**AlphaComposite** (implements java.awt.Composite)
  - class java.awt.**AWTEventMulticaster** (implements java.awt.event.ActionListener, java.awt.event.AdjustmentListener, java.awt.event.ComponentListener, java.awt.event.ContainerListener, java.awt.event.FocusListener, java.awt.event.InputMethodListener, java.awt.event.ItemListener, java.awt.event.KeyListener, java.awt.event.MouseListener, java.awt.event.MouseMotionListener, java.awt.event.TextListener, java.awt.event.WindowListener)
  - class java.awt.**BasicStroke** (implements java.awt.Stroke)
  - class java.awt.**BorderLayout** (implements java.awt.LayoutManager2, java.io.Serializable)
  - class java.awt.**CardLayout** (implements java.awt.LayoutManager2, java.io.Serializable)
  - class java.awt.**CheckboxGroup** (implements java.io.Serializable)
  - class java.awt.**Color** (implements java.awt.Paint, java.io.Serializable)
    - class java.awt.**SystemColor** (implements java.io.Serializable)
  - class java.awt.**Component** (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.**Button**
    - class java.awt.**Canvas**
    - class java.awt.**Checkbox** (implements java.awt.ItemSelectable)
    - class java.awt.**Choice** (implements java.awt.ItemSelectable)
    - class java.awt.**Container**
      - class java.awt.**Panel**
      - class java.awt.**ScrollPane**
      - class java.awt.**Window**
        - class java.awt.**Dialog**
          - class java.awt.**FileDialog**
        - class java.awt.**Frame** (implements java.awt.MenuContainer)
    - class java.awt.**Label**
    - class java.awt.**List** (implements java.awt.ItemSelectable)
    - class java.awt.**Scrollbar** (implements java.awt.Adjustable)

- class java.awt.**TextComponent**
  - class java.awt.**TextArea**
  - class java.awt.**TextField**
- class java.awt.**ComponentOrientation** (implements java.io.Serializable)
- class java.awt.**Cursor** (implements java.io.Serializable)
- class java.awt.geom.**Dimension2D** (implements java.lang.Cloneable)
  - class java.awt.**Dimension** (implements java.io.Serializable)
- class java.awt.**Event** (implements java.io.Serializable)
- class java.util.**EventObject** (implements java.io.Serializable)
  - class java.awt.**AWTEvent**
- class java.awt.**EventQueue**
- class java.awt.**FlowLayout** (implements java.awt.LayoutManager, java.io.Serializable)
- class java.awt.**Font** (implements java.io.Serializable)
- class java.awt.**FontMetrics** (implements java.io.Serializable)
- class java.awt.**GradientPaint** (implements java.awt.Paint)
- class java.awt.**Graphics**
  - class java.awt.**Graphics2D**
- class java.awt.**GraphicsConfigTemplate** (implements java.io.Serializable)
- class java.awt.**GraphicsConfiguration**
- class java.awt.**GraphicsDevice**
- class java.awt.**GraphicsEnvironment**
- class java.awt.**GridBagConstraints** (implements java.lang.Cloneable, java.io.Serializable)
- class java.awt.**GridBagLayout** (implements java.awt.LayoutManager2, java.io.Serializable)
- class java.awt.**GridLayout** (implements java.awt.LayoutManager, java.io.Serializable)
- class java.awt.**Image**
- class java.awt.**Insets** (implements java.lang.Cloneable, java.io.Serializable)
- class java.awt.**MediaTracker** (implements java.io.Serializable)
- class java.awt.**MenuComponent** (implements java.io.Serializable)
  - class java.awt.**MenuBar** (implements java.awt.MenuContainer)
  - class java.awt.**MenuItem**
    - class java.awt.**CheckboxMenuItem** (implements java.awt.ItemSelectable)
    - class java.awt.**Menu** (implements java.awt.MenuContainer)
      - class java.awt.**PopupMenu**
- class java.awt.**MenuShortcut** (implements java.io.Serializable)
- class java.security.**Permission** (implements java.security.Guard, java.io.Serializable)
  - class java.security.**BasicPermission** (implements java.io.Serializable)
    - class java.awt.**AWTPermission**
- class java.awt.geom.**Point2D** (implements java.lang.Cloneable)
  - class java.awt.**Point** (implements java.io.Serializable)

- class java.awt.**Polygon** (implements java.io.Serializable, java.awt.Shape)
- class java.awt.**PrintJob**
- class java.awt.geom.**RectangularShape** (implements java.lang.Cloneable, java.awt.Shape)
  - class java.awt.geom.**Rectangle2D**
    - class java.awt.**Rectangle** (implements java.io.Serializable, java.awt.Shape)
- class java.awt.**RenderingHints** (implements java.lang.Cloneable, java.util.Map)
- class java.awt.**RenderingHints.Key**
- class java.awt.**TexturePaint** (implements java.awt.Paint)
- class java.lang.**Throwable** (implements java.io.Serializable)
  - class java.lang.**Error**
    - class java.awt.**AWTError**
  - class java.lang.**Exception**
    - class java.awt.**AWTException**
    - class java.lang.**RuntimeException**
      - class java.lang.**IllegalStateException**
        - class java.awt.**IllegalComponentStateException**
- class java.awt.**Toolkit**

Interface Hierarchy

- interface java.awt.**ActiveEvent**
- interface java.awt.**Adjustable**
- interface java.awt.**Composite**
- interface java.awt.**CompositeContext**
- interface java.awt.**ItemSelectable**
- interface java.awt.**LayoutManager**
- interface java.awt.**LayoutManager2**
- interface java.awt.**MenuContainer**
- interface java.awt.**PaintContext**
- interface java.awt.**PrintGraphics**
- interface java.awt.**Shape**
- interface java.awt.**Stroke**
- interface java.awt.**Transparency**
- interface java.awt.Paint

A continuación se muestra la parte de la ayuda del API referente al paquete AWT, incluyendo una breve descripción de las clases que contiene:

Package java.awt

Contains all of the classes for creating user interfaces and for painting graphics and images.

See:

Description

	The `Stroke` interface allows a `Graphics2D` object to obtain a `Shape` that is the decorated outline, or stylistic representation of the outline, of the

	A flow layout arranges components in a left-to-right flow, much like

	The `GridBagLayout` class is a flexible layout manager that aligns components vertically and horizontally, without requiring that the

