Interfaz Gráfica de Usuario en

Universidad Nacional de la Patagonia Austral

Unidad Académica Río Gallegos

Analista de Sistemas – Licenciatura en Sistemas

Laboratorio de Programación

Lic. Verónica L. Vanoli

Indice

Objetivos del Curso

Darle al alumno información necesaria que le ayude a conocer y manejar la parte de interfaz con el usuario codificada con el lenguaje de programación orientado a objetos Java. Proporcionándole así un complemento más en el uso del lenguaje.

Al finalizar el curso "Interfaz Gráfica de Usuario en Java", los alumnos deberán ser capaces de:

- Crear ventanas interactivas y poder usarlas.
- Adaptar programas con entrada y salida de datos a este tipo de herramienta.
- Tener la capacidad de continuar con el uso de clases relacionadas.
- Trasladar los contenidos básicos a otras asignaturas u otras actividades.

Introducción: Interfaz Gráfica de Usuario en Java

El objetivo de esta temática es ver todo lo que proporciona el entorno Java para la creación de una interfaz de usuario (UI). *UI* es un término que se refiere a todos los caminos de comunicación entre un programa y sus usuarios. UI no es sólo lo que ve el usuario, sino que también es lo que el usuario oye y siente. Incluso la velocidad con la que un programa interactúa con el usuario es una parte importante de la UI del programa.

El entorno Java proporciona clases para la funcionalidad de la UI, entre ellas se encuentra la UI gráfica (GUI) que es la preferida por la mayoría de los programas Java.

Se pueden crear aplicaciones o applets. Estos últimos son los que se escriben en una página Web y se instalan y se ejecutan desde de un browser. Las aplicaciones y los applets presentan información al usuario y lo invitan a interactuar utilizando una GUI. La parte del entorno Java llamada Herramientas de Ventanas Abstractas (Abstract Windows Toolkit - *AWT*) contiene un completo conjunto de clases para escribir programas GUI. El nombre del paquete es *java.awt*.

También existe un modelo distinto de componentes llamado *SWING,* incluido en Java como paquete adicional. El nombre del paquete es *javax.swing*, que forma parte de las Java Foundation Classes (*JFC*) que permite incorporar en las aplicaciones elementos gráficos de una forma mucho más versátil y con más capacidades que utilizando el AWT básico de Java.
Algunas de las características más interesantes son:

- Cualquier programa que utiliza componentes de *Swing* puede elegir el aspecto que desea para sus ventanas y elementos gráficos.
- Cualquier componente gráfico de *Swing* presenta más propiedades que el correspondiente elemento del *AWT*: Los botones pueden incorporan imágenes, hay nuevos *layouts* y paneles, menúes, …
- Posibilidad de *Drag & Drop*, es decir de seleccionar componentes con el mouse y arrastrar a otro lugar de la pantalla.

Para construir una interfaz gráfica de usuario hace falta principalmente:

1. Un "contenedor" o *Container*, que representa aquellos elementos gráficos susceptibles de contener. Todos los Componentes (excepto Windows y los que derivan de ella) deben ser

añadidos a un *Container* y sólo pueden estar en uno solo, si está en un *Container* y se añade a otro, deja de estar en el primero. También un *Container* puede ser añadido a otro *Container*. De esta clase heredan a su vez las clases *Panel*, *ScrollPane* y *Window*. *Panel* y *ScrollPane*, sirven para representar grupos de objetos dentro de la ventana actual y deben estar siempre dentro de otro *Container*; mientras que *Window* representa las ventanas como tales, y de ella se derivan las subclases *Dialog* y *Frame*, que representan, respectivamente, los cuadros de diálogo y las ventanas normales.

2. Los *componentes* propiamente dichos. Para ello se utiliza la clase *Component* que es una clase abstracta que define todas las características comunes a los componentes visuales. Entre estas características están el manejo de la entrada de datos por teclado (lo que hasta ahora hacen usando la clase Console), la entrada por medio del mouse, las notificaciones de activación/desactivación de ventanas, el posicionamiento en pantalla y el control de repintado.

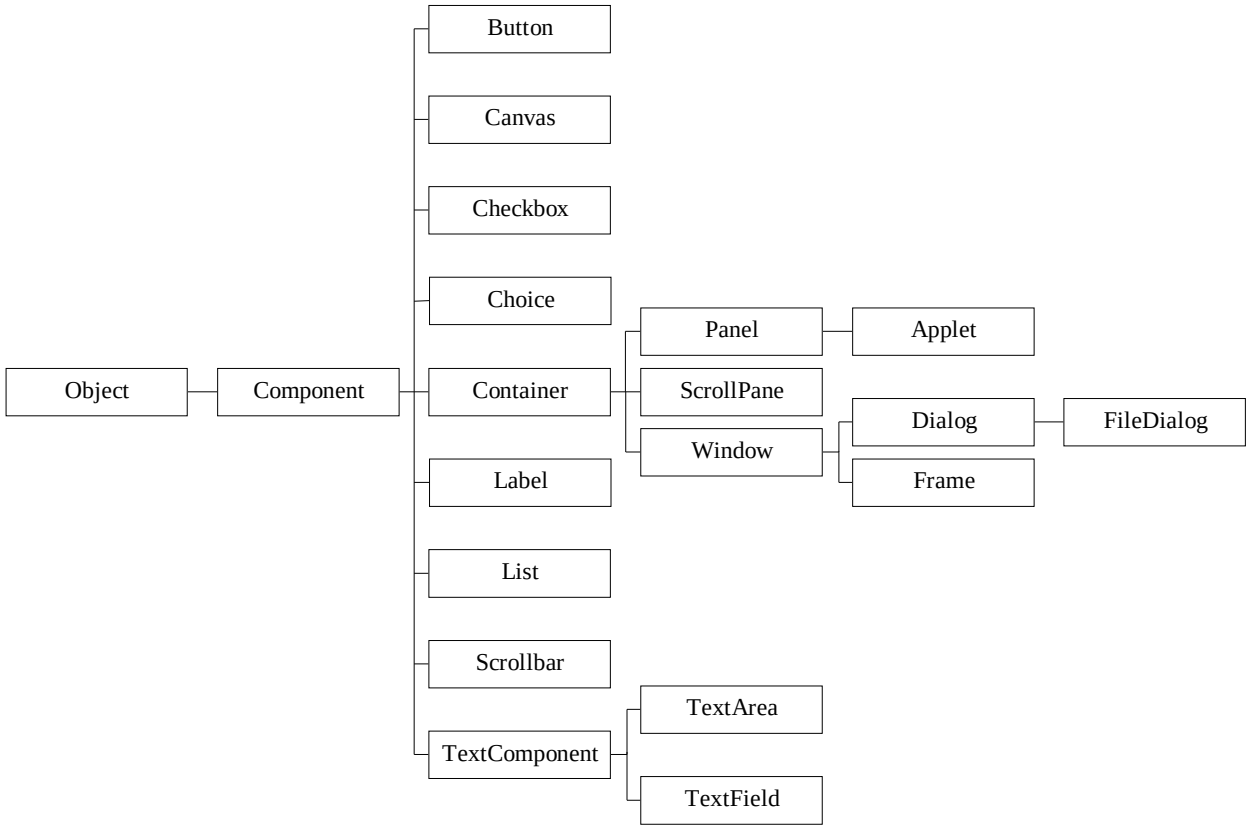
3. El *modelo de eventos*. El usuario controla la aplicación actuando sobre los componentes, ya sea con el mouse o con el teclado. Cada vez que el usuario realiza una determinada acción, se produce el *evento* correspondiente, que el sistema operativo transmite al AWT. El AWT crea un *objeto* de una determinada clase de evento, derivada de *AWTEvent*. Este evento es transmitido a un determinado método para que lo gestione. El componente u objeto que recibe el evento debe "registrar" o indicar previamente qué objeto se va a hacer cargo de gestionar ese evento. Esto también deriva a lo que se conoce como programación orientada a eventos.

También puede ser necesario, para mejorar una interfaz, el uso de otras clases del AWT que trabajan en un contexto gráfico (incluyendo las operaciones de dibujo básico), imágenes, eventos, fuentes y colores. Otro grupo importante de clases del AWT son los controladores de distribución o disposición que controlan el tamaño y la posición de los componentes.

Para empezar...

Jerarquía de Componentes

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



Las características más importantes de los componentes son:

- Todos los componentes excepto los contenedores de más alto nivel (*Window* y sus descendientes) deben de estar dentro de un contenedor.
- Un contenedor se puede poner dentro de otro contenedor.
- Un componente sólo puede estar dentro de un contenedor.

Contenedores

La clase *Container* es una clase abstracta derivada de *Component*, con métodos adicionales para permitir (como se mencionó antes) que, dentro de un objeto de tipo *Container* (*contenedor*), se puedan incluir otros objetos de tipo *Component*. El hecho de que la clase *Container* sea abstracta impide instanciarla directamente, por lo que se deberá derivar de ella subclases concretas antes de poder crear objetos.

Panel

Es la clase más simple para propósitos generales. Se puede utilizar tal y como es para contener componentes, o se puede definir una subclase para realizar alguna función específica, como el manejo de eventos para los objetos contenidos en el Panel. Para el caso de contener componentes, sirve para hacer una buena distribución manual de los mismos en otro contenedor, por ejemplo, un *Frame*. Existe también una manera de ahorrarse todo este posicionamiento manual, mediante el uso de los *Layout Managers*, que veremos más adelante.

Applet

Es una subclase de Panel con broches especiales para ejecutarse en un navegador o un visualizador de applets. Siempre que se vea un programa que se puede ejecutar tanto como un applet como una aplicación, el truco está en que define una subclase de applet, pero no utiliza ninguna de las capacidades especiales del Applet, utilizando sólo los métodos heredados de la clase Panel.

ScrollPane

Es como una ventana de tamaño limitado en la que se puede mostrar un componente de mayor tamaño con dos *Scrollbars* (barras de desplazamiento), una horizontal y otra vertical. El componente puede ser una imagen, por ejemplo. Las *Scrollbars* son visibles sólo si son necesarias (por defecto). Las constantes de la clase *ScrollPane* son (su significado es evidente): SCROLLBARS_AS_NEEDED, SCROLLBARS_ALWAYS, SCROLLBARS_NEVER. Los *ScrollPane* no generan eventos.
En el caso en que no aparezcan scrollbars (SCROLLBARS_NEVER) será necesario desplazar el componente (hacer *scrolling*) desde el programa, con el método *setScrollPosition()*.

Window

Los objetos de la clase *Window* son ventanas de máximo nivel, pero *sin bordes* y *sin barra de menú*es. En realidad son más interesantes las clases que derivan de ella: *Frame* y *Dialo*g.

Frame

Crea ventanas normales y completamente maduras. Es por ello que proporciona ventanas para los applets y las aplicaciones. Cada aplicación necesita al menos un Frame (Marco). Si una aplicación tiene una ventana que debería depender de otra ventana - desapareciendo cuando la otra ventana se minimiza, por ejemplo - debería utilizar un Dialog (Cuadro de Diálogo) en vez de un Frame para la ventana dependiente. Desafortunadamente, los applets no puede utilizar cajas de diálogo por ahora, por eso utilizan Frames en su lugar.

Dialog

Lo único que distingue a los cuadros de diálogo de las ventanas normales (que son implementadas con objetos Frame) es que el cuadro de diálogo depende de alguna otra ventana (un Frame). Cuando esta otra ventana es destruida, también lo son sus cuadros de diálogo dependientes. Cuando esta otra ventana es minimizada sus cuadros de diálogo desaparecen de la pantalla. Cuando esta otra ventana vuelve a su estado normal, sus cuadros de diálogo vuelven a aparecer en la pantalla. El AWT proporciona automáticamente este comportamiento.
Como no existe un API actualmente que permita a los Applets encontrar la ventana en la que se están ejecutando, estos generalmente no pueden utilizar cuadros de diálogo. La excepción son los applets que traen sus propias ventanas (Frames) que pueden tener cuadros de diálogo dependientes de esas ventanas.
Los cuadros de diálogo pueden ser *modales*. Los cuadros de diálogo modales requieren la atención del usuario, para evitar que el usuario no haga nada en la aplicación del cuadro de diálogo hasta que se haya finalizado con él. Por defecto, los cuadros de diálogo no son modales - el usuario puede mantenerlos y seguir trabajando con otras ventanas de la aplicación.

La clase Dialog proporciona una subclase muy útil llamada FileDialog que proporciona cuadros de diálogos que puede servir para Abrir o Guardar archivos.

Ejemplos:

1. Veamos un ejemplo sencillo respecto a las clases Frame, Panel y Dialog.

```
//importación del paquete necesario para trabajar
import java.awt.*;

//clase ejemplo para creación de un marco (hereda todo de Frame)
class EjemploMarco extends Frame
     {
     //atributos de la clase
     Panel panel;
     Dialog dialogo;

     EjemploMarco()
          {
          //título del marco que se ubica en la barra de título
          this.setTitle("Primer ejemplo de Marco...");
          //tamaño del marco (x,y,ancho,alto)
          this.setBounds(100,100,600,350);
          //color de fondo del marco
          this.setBackground(Color.gray);

          //creación del objeto panel
          panel=new Panel();
          //color de fondo del panel (usando la clase Color)
          panel.setBackground(Color.red);
          //agregado del objeto panel al marco
          this.add(panel);
```

```
          //creación del objeto dialogo
          dialogo=new Dialog(this,"Ejemplo de diálogo...",false);
          //tamaño del dialogo (x,y,ancho,alto)
          dialogo.setBounds(250,200,300,100);
          //coloca al dialogo como visible en la pantalla
          dialogo.setVisible(true);
          }
     }

//clase principal que hace uso del marco ejemplo
class EjemploContenedores
     {
     public static void main(String[] args)
          {
          //creación del objeto marco
          EjemploMarco marco=new EjemploMarco();
          //coloca al marco como visible en la pantalla
          marco.setVisible(true);
          }
     }
```

2. Veamos otro ejemplo sencillo respecto a las clases Frame, ScrollPane y Window.

```
//importación del paquete necesario para trabajar
import java.awt.*;

//clase ejemplo para creación de un marco (hereda todo de Frame)
class EjemploMarco extends Frame
     {
     //atributos de la clase
     ScrollPane panel;
     Window ventana;

     EjemploMarco()
          {
          //título del marco que se ubica en la barra de título
          this.setTitle("Segundo ejemplo de Marco...");
          //tamaño del marco (x,y,ancho,alto)
          this.setBounds(100,100,600,350);
          //color de fondo del marco
          this.setBackground(Color.gray);

          //creación del objeto panel(con barras de desplazamiento)
          panel=new ScrollPane(ScrollPane.SCROLLBARS_ALWAYS);
          //color de fondo del panel
          panel.setBackground(Color.blue);
          //agregado del objeto panel al marco
          this.add(panel);

          //creación del objeto ventana
          ventana=new Window(this);
```

```
          //tamaño de la ventana
          ventana.setBounds(250,200,300,100);
          //color de fondo de la ventana
          ventana.setBackground(Color.yellow);
          //coloca a la ventana como visible en la pantalla
          ventana.setVisible(true);
          }
     }

//clase principal que hace uso de los marcos ejemplos
class EjemploContenedores
     {
     public static void main(String[] args)
          {
          //creación del objeto marco
          EjemploMarco marco=new EjemploMarco();
          //coloca al marco como visible en la pantalla
          marco.setVisible(true);
          }
     }
```

Componentes

Canvas

La clase Canvas existe para tener descendientes. No hace nada por sí misma, sólo proporciona una forma para implementar un componente personalizado. Es un área rectangular de la pantalla y sirve como Lienzo (Canvas), que son útiles para áreas de dibujo para imágenes y gráficos del usuario, tanto si se desea o no manejar eventos que ocurran dentro del área de pantalla.
Los Canvas también son útiles cuando se quiera un control - un botón, por ejemplo - que se parezca a la implementación por defecto de ese control.

Label

La clase *Label* es una clase derivada de *Canvas* que dibuja un texto dentro de las fronteras delimitadas por su representación gráfica. Se puede especificar tanto el texto como el alineamiento dentro del rectángulo por medio de los parámetros del constructor. Para esto último, existen tres constantes, *Label.LEFT*, *Label.RIGHT* y *Label.CENTER*, que sitúan el texto, respectivamente, ajustado a la izquierda, ajustado a la derecha y centrado.
Por ejemplo:

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

//crea el texto alineado a la izquierda
Label etiizq=new Label("Izquierda", Label.LEFT);

//crea el texto alineado a la derecha
Label etider=new Label("Derecha", Label.RIGHT);

//crea el texto centrado en el rectángulo del componente
Label eticen=new Label("Centro", Label.CENTER);
```

Button

Este es un componente gráfico que permite invocar una cierta acción cuando el usuario presiona el botón del mouse manteniendo el cursor del mismo encima del botón. La representación gráfica varía según el sistema de ventanas sobre el que se trabaje; en Windows 95 es un botón con efecto tridimensional que se “hunde” en la superficie de la ventana al presionar con el mouse.
Los botones llevan asociada una etiqueta, que se pasa como objeto de tipo *String*, y cuyo texto va necesariamente centrado dentro de la representación gráfica del botón. La etiqueta se le asigna al botón en el momento de su creación como un parámetro del constructor.
Por ejemplo:

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

//crea un botón con el texto "Aceptar"
Button botaceptar=new Button("Aceptar");
```

```
//crea un botón con el texto "Cancelar"
Button botcancelar=new Button("Cancelar");
```

Checkbox y CheckboxGroup

Un componente de tipo *Checkbox* está compuesto por un pequeño recuadro, que puede contener o no una marca y una etiqueta que lo identifique. Habitualmente se emplea para selecciones de tipo si/no o activo/desactivo. El usuario puede cambiar de un estado a otro haciendo click con el mouse encima del recuadro de marca.
En el siguiente ejemplo se crean tres checkboxes que permiten seleccionar diversas características para un trozo de texto:

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

//se crea con estado inicial marcado
Checkbox chbnegrita=new Checkbox("Negrita", null, true);
Checkbox chbcursiva=new Checkbox("Cursiva");
Checkbox chbsubrayado=new Checkbox("Subrayado");
```

En el primer caso, se especifica en el constructor que se desea que el estado inicial sea marcado. El argumento intermedio, *null*, indica que se trata de un checkbox independiente, es decir, que no forma parte de un *CheckboxGroup*.
El estado de la marca se puede examinar o decidir por medio de los métodos *getState()* y *setState()*, respectivamente. El primero de estos métodos entrega *true* en caso de que el recuadro esté marcado y *false* en caso contrario. El método *setState()* admite un argumento booleano con este mismo significado.
En ocasiones, es interesante disponer, no de controles con marcado independiente, sino de grupos de ellos que permitan seleccionar una de entre varias opciones mutuamente excluyentes, al modo de los Radiobuttons de Windows. En este caso, lo que se hace es agrupar objetos de tipo *Checkbox* dentro de otro de tipo *CheckboxGroup*. Esto se hace pasando el objeto de *CheckboxGroup* como parámetro al constructor del *Checkbox*. Cuando se agrupan varios *Checkbox* en un grupo, lo normal es que el sistema de ventanas les asigne una representación gráfica diferente a la de los *Checkbox* aislados.
Veamos un ejemplo de *CheckboxGroup* en el que se crean cuatro botones de radio para cuatro opciones mutuamente excluyentes:

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

//crea el grupo en el que se meterán las opciones
CheckboxGroup chbgcolpelo=new CheckboxGroup();

//se crean las opciones
//se pasa el grupo como parámetro del constructor de cada una
//inicialmente marcada
Checkbox chbmoreno=new Checkbox("Moreno", chbgcolpelo, true);

//inicialmente sin marca
Checkbox chbcastanio=new Checkbox("Castaño", chbgcolpelo, false);
```

```
//inicialmente sin marca
Checkbox chbrubio=new Checkbox("Rubio", chbgcolpelo, false);

//inicialmente sin marca
Checkbox chbpelirojo=new Checkbox("Pelirojo", chbgcolpelo, false);
```

Choice

La clase *Choice* define un tipo de componente gráfico que permite presentar una lista de selección desplegable. Por defecto, se presenta en un recuadro una de las opciones. Un botón a la derecha de este recuadro permite desplegar la lista completa. Una vez escogida una de las opciones con el mouse o el teclado, la lista desaparece y la nueva opción se presenta en el recuadro de texto.
Los textos de las opciones se añaden al componente mediante el método *addItem()*, y deben ser de tipo *String*. En el siguiente ejemplo, creamos una lista desplegable con tres opciones:

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

//crea el objeto
Choice clistadesp=new Choice();

//añade primera opción
clistadeps.addItem("Primera");

//añade segunda opción
clistadeps.addItem("Segunda");

//añade tercera opción
clistadeps.addItem("Tercera");
```

El método *select()* permite decidir cuál de los elementos de la lista está seleccionado en un momento dado, pasándole como parámetro ya sea el índice del elemento (el primer elemento tiene índice 0) o el texto del mismo.
El método *getSelectedItem()* entrega el texto del elemento seleccionado en el momento en que se llama, y *getSelectedIndex()* devuelve el índice correspondiente.

List

Esta clase define un tipo de componente gráfico que permite presentar a un tiempo varias opciones, con la posibilidad de seleccionar una o varias de ellas, según el tipo de lista. Se corresponde casi exactamente con las List boxes de Windows. El que la lista sea de selección simple o múltiple viene dado por el segundo argumento del constructor que toma, respectivamente, los valores false o true. El primer argumento determina cuántos elementos van a ser visibles al mismo tiempo.
En el siguiente ejemplo creamos una lista con selección simple, esto es, sólo un elemento puede estar seleccionado en un momento dado:

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

```
//se van a ver tres elementos a la vez
List lislenguaje=new List(3, false);

//se añaden los ítems
lislenguaje.addItem("C++");
lislenguaje.addItem("Pascal");
lislenguaje.addItem("Java");
lislenguaje.addItem("Delphi");
lislenguaje.addItem("Prolog");
lislenguaje.addItem("Lisp");
```

Y en este otro, creamos una lista con posibilidad de selección múltiple:

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

//se van a ver los cuatro elementos
List lisversion=new List(4, true);

//se añaden los ítems
lisversion.addItem("1.0");
lisversion.addItem("1.3");
lisversion.addItem("2.0");
lisversion.addItem("3.0.1");
```

Scrollbar

Este componente no es otra cosa que la familiar barra de scroll que nos permite desplazarnos de una parte a otra de un documento que es demasiado grande para caber en la parte de cliente de la ventana a la que va asociado. También se puede emplear de manera aislada para asignar un valor a una variable cualquiera dentro de un rango. En la mayoría de los sistemas de ventanas, las barras de scroll tienen una pieza móvil (thumb) que puede moverse manualmente mediante el mouse. Además, existen dos botones en los extremos que permiten llevar a cabo un desplazamiento de la pieza móvil hacia el lado del botón que se presione. Si se hace click en la parte de la barra que está entre la pieza móvil y el botón del extremo, se produce un desplazamiento mayor que el obtenido al presionar el botón. Si se trata de un procesador de textos, por ejemplo, esta última acción efectuada en la barra de desplazamiento vertical se interpreta como un avance o retroceso de página, equivalente a presionar las teclas RePág o AvPág.
Los parámetros del constructor permiten determinar si la barra va a ser horizontal o vertical, por medio de las constantes predefinidas *Scrollbar.HORIZONTAL* y *Scrollbar.VERTICAL*, así como parámetros tales como el rango total de valores, el tamaño de la pieza móvil, etc.
Veamos un ejemplo:

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

Scrollbar sbhorizontal=new Scrollbar(Scrollbar.HORIZONTAL,
          50, //valor inicial
          3, //tamaño de la pieza móvil
          0, //valor mínimo
          100); //valor máximo
```

TextComponent

La clase TextComponent tiene dos subclases llamadas TextField y TextArea. Las mismas pueden heredar métodos que les permiten cambiar y obtener la selección actual, permitir o desactivar la edición, obtener el texto seleccionado actualmente (o todo el texto), y modificar el texto. Veamos bien estas dos subclases.

TextField

Esta clase define un componente gráfico que permite editar una única línea de texto. El constructor de la clase permite decidir cuántos caracteres van a poder introducirse en el campo y cuál va a ser el texto por defecto, es decir, el texto que se carga inicialmente en el componente. El contenido del componente se puede editar (siempre que esta acción esté permitida) mediante las teclas habituales y mediante el mouse. Este contenido se puede recuperar con el método *toString()*, heredado de la clase *Component*.
El método *setEditable()*, cuando se le pasa false como parámetro, impide que se pueda editar el texto contenido en el campo. El método *isEditable()* devuelve true si la edición está permitida, y false en caso contrario.
Es frecuente que exista la necesidad de ocultar los caracteres tecleados, presentando en su lugar un único caracter, tal y como sucede al introducir una contraseña. En este caso, se puede llamar al método *setEchoCharacter()*, pasándole el caracter único que se va a presentar en el lugar de los que se introduzcan realmente. Para averiguar si un *TextField* está en modo contraseña, se invoca el método *echoCharlsSet()*, que devuelve true si está en este modo. Para saber además qué caracter en concreto se está usando, basta con hacer una llamada al método *getEchoChar()*.
Veamos un ejemplo en el que se crean dos campos, nombre de usuario y contraseña, que se utilizan habitualmente para controlar el acceso a un sistema. El caracter elegido para el campo de contraseña es el típico asterisco. Ambos campos tienen 10 caracteres como máximo.

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

//creación del campo de nombre del usuario
TextField tfnombre=new TextField(10);

//creación del campo de password del usuario
TextField tfpassword=new TextField(10);

//establece el asterisco como caracter de eco
tfpassword.setEchoCharacter('*');
```

TextArea

Este componente permite editar varias líneas de texto llano (formato .txt). El constructor de *TextArea* permite especificar tanto el texto inicial que se va a introducir en el componente como el número de filas y columnas que va a admitir éste. Una vez creado el componente, el usuario puede editar el contenido como desee. Al igual que con *TextField*, este contenido se puede recuperar con el método *toString()*, heredado de la clase *Component*.
El método *appendText()* permite añadir texto al final del que ya existe en el área. El método *insertText()*, por su parte, sirve para insertar texto en una posición determinada.
El siguiente código de ejemplo crea un objeto de tipo *TextArea* e introduce texto en él:

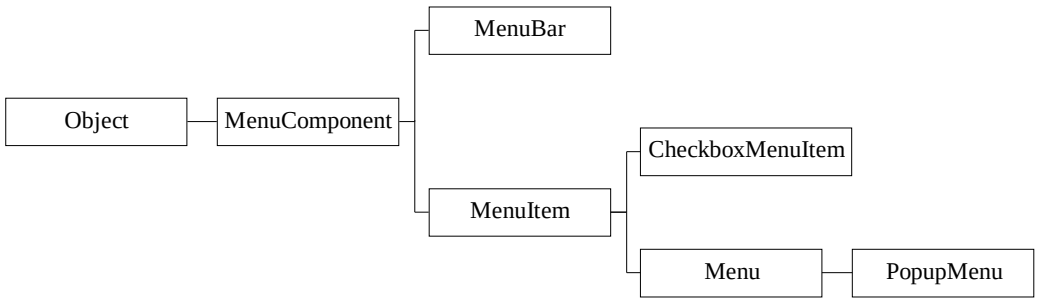
```
import java.awt.*;
...

String texto="Prueba de edición \n en varias líneas \n";

//40 columnas y 10 filas
TextArea taeditor=new TextArea(texto, 40, 10);

//añade el string al contenido anterior
taeditor.appendText("con este texto añadido");
```

Layout Managers

En lo que antecede, no hemos mencionado el problema de posicionar manualmente cada componente gráfico en el contenedor al que pertenece, ni el de adaptar esta posición a los cambios que el usuario vaya haciendo en el tamaño total o en las proporciones de la ventana.
Afortunadamente, AWT permite automatizar en gran medida este tipo de tareas, por medio de la interfaz *LayoutManager*. Todo objeto de tipo *Container* lleva asociado un objeto (el layout manager) que es una instancia de una clase que implementa esta interfaz. Cuando el usuario cambia las dimensiones del contenedor, se invoca automáticamente al layout manager para que vuelva a colocar los componentes gráficos en él incluidos. El layout manager asociado a un contenedor determinado se establece llamando al método *setLayout()*.
Java tiene algunos layouts predefinidos, como por ejemplo:
FlowLayout: Los componentes se van colocando de izquierda a derecha y de arriba hacia abajo, como las palabras en un editor de texto. Este es por definición.
BorderLayout: Se colocan cuatro componentes en los cuatro lados de la ventana y el resto en el centro.
GridLayout: Crea una cuadrícula de dimensiones especificadas y ajusta los componentes a ella.

Menúes

Como muestra la figura en la jerarquía de componentes, todos los componentes, excepto los componentes relacionados con los menúes, descienden de la clase Component del AWT. A causa de las restricciones de las distintas plataformas (como la imposibilidad de seleccionar el color de fondo de los menúes), todos los componentes relacionados con los menúes se han sacado fuera de la clase Component. En su lugar, los componentes de menúes descienden de la clase MenuComponent del AWT.



Cualquier ventana puede tener una barra de menú asociada, que será una instancia de la clase *ManuBar*. Esta puede contener varios objetos de la clase *Menu*, que puede, a su vez, contener objetos de tipo *MenuItem*. Veamos un sencillo ejemplo:

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

//crea la barra de menú
MenuBar mbmenu=new MenuBar();
...

//crea el primer menú y le coloca el nombre "Archivo"
Menu marchivo=new Menu("Archivo");

//crea el segundo menú y le coloca el nombre "Edición"
Menu medicion=new Menu("Edición");
...

//se añaden al menú Archivo las opciones concretas
marchivo.add(new MenuItem("Nuevo..."));
marchivo.add(new MenuItem("Abrir..."));
marchivo.add(new MenuItem("Cerrar"));
marchivo.add(new MenuItem("Guardar"));
marchivo.add(new MenuItem("Guardar como..."));
marchivo.add(new MenuItem("Salir"));
...

//se añaden las opciones al menú de Edición
medicion.add(new MenuItem("Deshacer"));
medicion.add(new MenuItem("Copiar"));
medicion.add(new MenuItem("cortar"));
medicion.add(new MenuItem("Pegar"));
...

//se introducen los menúes en la barra creada previamente
mbmenu.add(marchivo);
mbmenu.add(medicion);
```

Ejemplos:

1. Simular la ventana de salida de Windows.

```
import java.awt.*;

class VentanaSalida extends Frame
     {
     //atributos de la clase (serían los componentes que tiene
     //el frame)
     private Panel panel1,panel2,panel22,panel3;
     private Label etiqueta;
```

```
private CheckboxGroup grupo;
private Checkbox opcion1,opcion2,opcion3;
private Button baceptar,bcancelar,bayuda;

VentanaSalida()
     {
     //variable de tipo font, para usarla en varias partes
     Font letra=new Font("Times new roman",Font.PLAIN,13);

     //todas las características necesarias para el frame
     this.setTitle("Cerrar Windows...");
     this.setBounds(200,200,400,200);
     this.setBackground(Color.lightGray);
     this.setResizable(false);
     //en vez de acomodar los componentes de acuerdo a como
     //lo da java por definición, utilizo un layout manager
     this.setLayout(new BorderLayout());

     //primer panel con una etiqueta (creación y agregado)
     panel1=new Panel(new FlowLayout(FlowLayout.LEFT));
     this.add(panel1, BorderLayout.NORTH);
     etiqueta=new Label("Confirme que desea:");
     etiqueta.setFont(letra);
     panel1.add(etiqueta);

     //segundo panel con otro panel dentro para poder
     //distribuir mejor los checkbox que tendrá (creación y
     //agregado)
     panel2=new Panel();
     this.add(panel2, BorderLayout.CENTER);
     panel22=new Panel(new GridLayout(3,1));
     grupo=new CheckboxGroup();
     opcion1=new Checkbox("apagar el sistema", grupo, true);
     opcion1.setFont(letra);
     opcion2=new Checkbox("reiniciar", grupo, false);
     opcion2.setFont(letra);
     opcion3=new Checkbox("reiniciar modo DOS", grupo, false);
     opcion3.setFont(letra);
     panel22.add(opcion1);
     panel22.add(opcion2);
     panel22.add(opcion3);
     panel2.add(panel22);

     //tercer panel con tres botones (creación y agregado)
     panel3=new Panel(new FlowLayout(FlowLayout.RIGHT,17,7));
     this.add(panel3, BorderLayout.SOUTH);
     baceptar=new Button("Aceptar");
     baceptar.setFont(letra);
     bcancelar=new Button("Cancelar");
     bcancelar.setFont(letra);
     bayuda=new Button("Ayuda");
```

```
          bayuda.setFont(letra);
          panel3.add(baceptar);
          panel3.add(bcancelar);
          panel3.add(bayuda);
          }
     }


class EjemploComponentes1
     {
     public static void main(String[] args)
          {
          VentanaSalida ventana=new VentanaSalida();
          ventana.setVisible(true);
          }
     }
```

2. Realizar un programa que permita hacer reservas de turnos de un mes determinado para un médico.

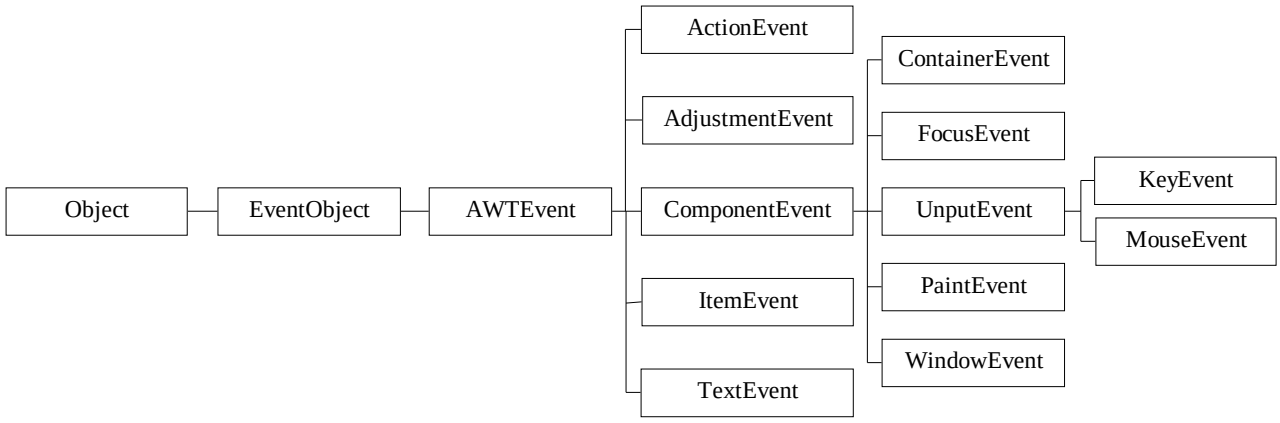
```
import java.awt.*;

class VentanaTurnoMedico extends Frame
     {
     private MenuBar menu;
     private Menu turnos,pacientes,gral;
     private Panel panel,panel1,panel2,panel3;
     private Label etiq1,etiq2,etiq3,etiq4,etiq5;
     private TextField nombpac,hora,minuto;
     private List obsoc;
     private Choice dias;

     VentanaTurnoMedico()
          {
          this.setTitle("Turnos del mes de Mayo...");
          this.setBounds(100,100,600,350);
          this.setForeground(new Color(50,50,150));
          this.setFont(new Font("Arial",Font.BOLD,15));
          this.setResizable(false);

          //creación y agregado del menú
          menu=new MenuBar();
          turnos=new Menu("Turnos");
          turnos.add(new MenuItem("Nuevo..."));
          turnos.add(new MenuItem("Abrir..."));
          turnos.add(new MenuItem("Guardar"));
          menu.add(turnos);
          pacientes=new Menu("Pacientes");
          pacientes.add(new MenuItem("Buscar"));
          pacientes.add(new MenuItem("Agregar"));
```

```
pacientes.add(new MenuItem("Eliminar"));
menu.add(pacientes);
gral=new Menu("General");
gral.add(new MenuItem("Backup"));
gral.add(new MenuItem("Control"));
gral.add(new MenuItem("-"));
gral.add(new MenuItem("Salir"));
menu.add(gral);
this.setMenuBar(menu);

//panel principal que contiene los componentes
panel=new Panel(new FlowLayout(FlowLayout.LEFT,70,35));
this.add(panel);
//primer panel con etiqueta y campo de texto (creación y
//agregado)
panel1=new Panel(new FlowLayout(FlowLayout.LEFT,10,0));
panel.add(panel1);
etiq1=new Label("Nombre del Paciente:");
panel1.add(etiq1);
nombpac=new TextField(35);
panel1.add(nombpac);

//segundo panel con etiqueta y lista (creación y
//agregado)
panel2=new Panel(new FlowLayout(FlowLayout.LEFT,10,0));
panel.add(panel2);
etiq2=new Label("Obra Social:");
panel2.add(etiq2);
obsoc=new List(5,false);
obsoc.add("CSS     ");
obsoc.add("OSDE    ");
obsoc.add("DOCTHOS ");
obsoc.add("OSPLAD  ");
obsoc.add("Otros...");
panel2.add(obsoc);

//tercer panel con etiquetas, lista desplegable y campos
//de texto (creación y agregado)
panel3=new Panel(new FlowLayout(FlowLayout.LEFT,10,0));
panel.add(panel3);
etiq3=new Label("Día de la Semana:");
panel3.add(etiq3);
dias=new Choice();
dias.addItem("Lunes    ");
dias.addItem("Martes   ");
dias.addItem("Miércoles");
dias.addItem("Jueves   ");
dias.addItem("Viernes  ");
panel3.add(dias);
etiq4=new Label("Hora:");
panel3.add(etiq4);
```

```
          hora=new TextField("00",2);
          panel3.add(hora);
          etiq5=new Label(":");
          panel3.add(etiq5);
          minuto=new TextField("00",2);
          panel3.add(minuto);
          }
     }

class EjemploComponentes2
     {
     public static void main(String[] args)
          {
          VentanaTurnoMedico ventana=new VentanaTurnoMedico();
          ventana.setVisible(true);
          }
     }
```

Eventos

Jerarquía de Eventos



Los eventos de *Java* pueden ser de alto y bajo nivel. Los *eventos de alto nivel* se llaman también *eventos semántico*s, porque la acción de la que derivan tiene un significado en sí misma, en el contexto de las interfaces gráficas de usuario. Los *eventos de bajo nivel* son las acciones elementales que hacen posible los eventos de alto nivel; son los que se producen con las operaciones elementales con el mouse, teclado, containers y windows

Son eventos de alto nivel los cuatro que tienen que ver con: hacer click sobre botones o elegir comandos en menúes (ActionEvent), cambiar valores en barras de desplazamiento (AdjustmentEvent), elegir valores (ItemEvent) y cambiar el texto (TextEvent).

Las seis clases de eventos de bajo nivel son los eventos relacionados con componentes (ComponentEvent), con los containers (ContainerEvent), con presionar teclas (KeyEvent), con mover, arrastrar, presionar y soltar con el mouse (MouseEvent), con obtener o perder el foco (FocusEvent) y con las operaciones con ventanas (WindowEvent).

Eventos de AWT

AWTEvent es el padre de todos los eventos de AWT que se encuentra en el paquete *java.awt.event*.

Cuando un componente produce cualquier tipo de evento (como hacer click sobre él, mover el mouse, utilizar el teclado, etc.) llama a un método determinado de la clase que implemente *EventListener* y le pasa un objeto que describe ese evento para que realice las acciones correspondientes.

La diferencia entre los EventSources y EventListeners es que:

- el *EventSource* (fuente de evento) es el Component que recibe la acción del usuario y que produce un evento, por ejemplo, un botón que el usuario le hizo click;
- un *EventListener* (escuchador de evento) es un objeto que implementa una interfaz *Listener*, y que por ello dispone de métodos para gestionar los eventos que se pasan como argumentos.

Para que un componente produzca un evento es necesario utilizar el método add() correspondiente para cada tipo de evento. Sería el método del EventSource que permite registrar uno o más EventListeners que están interesados en un tipo de eventos concretos.

<fuente-evento>.add<metodo-escuchador-evento>(<escuchador-evento>)

Componentes y Eventos
La siguiente tabla muestra los componentes del AWT, con los eventos específicos de cada uno de ellos y la interfaz Listener o el Adapter correspondiente al evento, así como una breve explicación sobre qué consiste cada tipo de evento:

Component	Eventos generados	Interfaz Listener - Adapter	Significado
Button	ActionEvent	ActionListener	Hacer click en el botón
Checkbox	ItemEvent	ItemListener	Seleccionar o deseleccionar un ítem.
Choice	ItemEvent	ItemListener	Seleccionar o deseleccionar un ítem.
Component	ComponentEvent	ComponentListener ComponentAdapter	Mover, cambiar el tamaño, mostrar u ocultar un componente.
	FocusEvent	FocusListener FocusAdapter	Obtener o perder el foco.
	KeyEvent	KeyListener KeyAdapter	Presionar o soltar una tecla.
	MouseEvent	MouseListener MouseAdapter	Presionar o soltar un botón del mouse; entrar o salir de un componente; mover o arrastrar el mouse (recordar que este evento posee dos Listeners)
		MouseMotionListener MouseMotionAdapter	
Container	ContainerEvent	ContainerListener ContainerAdapter	Añadir o eliminar un componente de un container.
List	ActionEvent	ActionListener	Hacer doble click sobre un ítem de la lista.
	ItemEvent	ItemListener	Seleccionar o deseleccionar un ítem.
MenuItem	ActionEvent	ActionListener	Seleccionar un ítem de un menú.
Scrollbar	AdjustementEvent	AdjustementListener	Cambiar el valor del scrollbar.
TextComponent	TextEvent	TextListener	Cambiar el texto.
TextField	ActionEvent	ActionListener	Terminar de editar un texto presionando Enter.
Window	WindowEvent	WindowListener WindowAdapter	Acciones sobre la ventana: abrir, cerrar, iconizar, restablecer e iniciar el cierre.

Los Adapter son clases derivadas de Object que contienen definiciones vacías de los métodos correspondientes a las interfaces EventListener que declaran más de un método. Son una ayuda para no tener que definir un gran número de métodos vacíos.

Ejemplo:

1. **Hacer una herramienta para graficar tres tipos de figuras simples y que se les pueda cambiar el color.**

```
//clase Dialogo
import java.awt.*;

class Dialogo extends Dialog
     {
```

```
     private Frame padre;

     Dialogo(Frame parent, String title, boolean modal)
          {
          //creación del objeto invocando a la clase heredada
          super(parent, title, modal);
          padre=parent;
          }

     public void mostrarMensaje(String mensaje)
          {
          this.setBounds(250,200,300,100);
          this.add(new Label(mensaje));
          //entrega del foco al dialogo
          this.requestFocus();
          this.setVisible(true);
          //entrega nuevamente del foco al padre (Frame)
          padre.requestFocus();
          }
     }

//clase Lienzo
import java.awt.*;

class Lienzo extends Canvas
     {
     private int num,x,y,an,al;

     public void pintar(int num,int x,int y,int an,int al)
          {
          this.num=num;
          this.x=x;
          this.y=y;
          this.an=an;
          this.al=al;
          }

     //creación del mismo método que la clase heredada para que la
     //actualización sea de otra manera (sobreescritura)
     public void update(Graphics g)
          {
          //los tres pasos necesarios para actualizar el lienzo
          g.setColor(this.getBackground());
          g.fillRect(0, 0, this.getWidth(), this.getHeight());
          g.setColor(this.getForeground());
          //se dibuja un arco (1), un ovalo (2) o un rectángulo (3)
          if(num==1)
               g.drawArc(x,y,an,al,x*2,y*2);
          else
               if(num==2)
                    g.drawOval(x,y,an,al);
```

```
                else
                     g.drawRect(x,y,an,al);
          }
     }

//clase EscuchadorBotones
import java.awt.event.*;
import java.awt.*;
//se implementa la interfaz ActionListener para los botones
class EscuchadorBotones implements ActionListener
     {
     public void actionPerformed(ActionEvent evento)
          {
          //ver quién es la fuente del evento
          if(evento.getSource() instanceof Button)
               {
               //verificación de la fuente y envío de tarea
               Button bot=(Button)evento.getSource();
               Panel pane1=(Panel)bot.getParent();
               Panel pane=(Panel)pane1.getParent();
               VentanaDibujo vent=(VentanaDibujo)pane.getParent();
               vent.dibujar(bot);
               }
          else
               if(evento.getSource() instanceof MenuItem)
                    {
                    //verificación de la fuente y envío de tarea
                    MenuItem mi=(MenuItem)evento.getSource();
                    Menu m=(Menu)mi.getParent();
                    MenuBar mb=(MenuBar)m.getParent();
                    VentanaDibujo vent=(VentanaDibujo)
                                        mb.getParent();
                    vent.barraMenu(mi);
                    }
          }
     }

//clase EscuchadorItems
import java.awt.event.*;
import java.awt.*;

//se implementa la interfaz ItemListener para los ítems
class EscuchadorItems implements ItemListener
     {
     public void itemStateChanged(ItemEvent evento)
          {
          //ver quién es la fuente del evento
          if(evento.getSource() instanceof Choice)
               {
               //verificación de la fuente y envío de tarea
               Choice bot=(Choice)evento.getSource();
```

```
                Panel pane1=(Panel)bot.getParent();
                Panel pane=(Panel)pane1.getParent();
                VentanaDibujo vent=(VentanaDibujo)pane.getParent();
                vent.cambiarColor(bot);
                }
           }
     }

//clase EscuchadorVentanas
import java.awt.event.*;

//se implementa la interfaz WindowAdapter para las ventanas
class EscuchadorVentanas extends WindowAdapter
     {
     public void windowClosing(WindowEvent evento)
           {
           //ver quién es la fuente del evento
           if(evento.getSource() instanceof Dialogo)
                {
                //determinación de evento y cierre (deja de
                //ser visible)
                Dialogo d=(Dialogo)evento.getWindow();
                d.setVisible(false);
                }
           else
                {
                //salida del programa (termina la ejecución actual
                //del Java Virtual Machine)
                System.out.println("Cierre de la ventana...");
                System.exit(0);
                }
           }
     }

//clase EjemploEventos
import java.awt.*;

class VentanaDibujo extends Frame
     {
     private MenuBar menu;
     private Menu ayuda;
     private MenuItem acerca;
     private Lienzo canvas;
     private Panel panel,panel1,panel2,panel3,panel4,panel5;
     private CheckboxGroup grupo;
     private Checkbox opcion1,opcion2,opcion3;
     private TextField puntoX,puntoY,ancho,alto;
     private Label etiq1,etiq2,etiq3,etiq4;
     private Button boton;
     private Choice color;
     private Dialogo dialogo;
```

```
VentanaDibujo()
     {
     this.setTitle(":: H E R R A M I E N T A   P A R A   D I
                    B U J A R ::");
     this.setBounds(100,100,600,350);
     this.setBackground(Color.lightGray);
     this.setResizable(false);
     this.setLayout(new GridLayout(1,2));
     //los dos pasos para generar un evento con la ventana
     EscuchadorVentanas manven1=new EscuchadorVentanas();
     this.addWindowListener(manven1);

     //componentes
     menu=new MenuBar();
     ayuda=new Menu("Ayuda");
     acerca=new MenuItem("Acerca de...");
     ayuda.add(acerca);
     menu.add(ayuda);
     this.setMenuBar(menu);
     //los dos pasos para generar un evento con el menuitem
     EscuchadorBotones manbot1=new EscuchadorBotones();
     acerca.addActionListener(manbot1);

     canvas=new Lienzo();
     canvas.setBackground(Color.white);
     this.add(canvas);

     panel=new Panel(new FlowLayout(FlowLayout.LEFT,100,10));
     this.add(panel);
     grupo=new CheckboxGroup();
     opcion1=new Checkbox("Arco", grupo, true);
     opcion2=new Checkbox("Ovalo", grupo, false);
     opcion3=new Checkbox("Rectángulo", grupo, false);
     panel.add(opcion1);
     panel.add(opcion2);
     panel.add(opcion3);
     panel1=new Panel();
     panel.add(panel1);
     puntoX=new TextField("100",3);
     panel1.add(puntoX);
     etiq1=new Label("-> coordenada x");
     panel1.add(etiq1);
     panel2=new Panel();
     panel.add(panel2);
     puntoY=new TextField("100",3);
     panel2.add(puntoY);
     etiq2=new Label("-> coordenada y");
     panel2.add(etiq2);
     panel3=new Panel();
     panel.add(panel3);
     ancho=new TextField("100",3);
```

```
        panel3.add(ancho);
        etiq3=new Label("-> ancho");
        panel3.add(etiq3);
        panel4=new Panel();
        panel.add(panel4);
        alto=new TextField("100",3);
        panel4.add(alto);
        etiq4=new Label("-> alto");
        panel4.add(etiq4);
        panel5=new Panel();
        panel.add(panel5);
        boton=new Button("DIBUJAR");
        panel5.add(boton);
        color=new Choice();
        color.add("NEGRO");
        color.add("ROJO");
        color.add("AZUL");
        color.add("AMARILLO");
        panel5.add(color);
        //los dos pasos para generar un evento con el botón
        EscuchadorBotones manbot2=new EscuchadorBotones();
        boton.addActionListener(manbot2);
        //los dos pasos para generar un evento con el ítem
        EscuchadorItems manite=new EscuchadorItems();
        color.addItemListener(manite);

        dialogo=new Dialogo(this, "Acerca de...", true);
        //los dos pasos para generar un evento con el diálogo
        EscuchadorVentanas manven2=new EscuchadorVentanas();
        dialogo.addWindowListener(manven2);
        }

    //método de la operación que dispara el evento del menuitem
    void barraMenu(MenuItem mi)
        {
        if(mi.getLabel().equals("Acerca de..."))
            {
            dialogo.mostrarMensaje("HERRAMIENTA  GRAFICA  VERSION
                        1.0");
            }
        //mensaje al método update()
        canvas.repaint();
        }

    //método de la operación que dispara el evento del botón
    void dibujar(Button bot)
        {
        if(bot.getLabel().equals("DIBUJAR"))
            {
            if(opcion1.getState())
                {
```

```
                    canvas.pintar(1,aValor(puntoX.getText()),
                                   aValor(puntoY.getText()),
                                   aValor(ancho.getText()),
                                   aValor(alto.getText()));
                    }
               else
                    if(opcion2.getState())
                         {
                         canvas.pintar(2,aValor(puntoX.getText()),
                                        aValor(puntoY.getText()),
                                        aValor(ancho.getText()),
                                        aValor(alto.getText()));
                         }
                    else
                         {
                         canvas.pintar(3,aValor(puntoX.getText()),
                                        aValor(puntoY.getText()),
                                        aValor(ancho.getText()),
                                        aValor(alto.getText()));
                         }
               }
          //mensaje al método update()
          canvas.repaint();
          }

     //método que convierte una cadena a un entero
     int aValor(String texto)
          {
          int num=0,val=1;
          for(int i=texto.length()-1; i>=0; i--)
               {
               num=num+
                    Character.getNumericValue(texto.charAt(i))*val;
               val=val*10;
               }
          return num;
          }

     //método de la operación que dispara el evento del ítem
     void cambiarColor(Choice ch)
          {
          if(ch.getSelectedItem().equals("NEGRO"))
               {
               canvas.setForeground(Color.black);
               }
          else
               if(ch.getSelectedItem().equals("ROJO"))
                    {
                    canvas.setForeground(Color.red);
                    }
               else
```

```
                    if(ch.getSelectedItem().equals("AZUL"))
                         {
                         canvas.setForeground(Color.blue);
                         }
                    else
                        if(ch.getSelectedItem().equals("AMARILLO"))
                              {
                              canvas.setForeground(Color.yellow);
                              }
          //mensaje al método update()
          canvas.repaint();
          }
     }

//clase principal
class EjemploEventos
     {
     public static void main(String[] args)
          {
          VentanaDibujo ventana=new VentanaDibujo();
          ventana.setVisible(true);
          }
     }
```

Bibliografía utilizada

- Biblioteca Técnica de Programación. “Cómo Programar en JAVA”. Prensa Técnica. 1997.
- http://members.tripod.com/Juan_Antonio/descarga/tutorjava.zip
- http://www1.ceit.es/Asignaturas/Informat2/Clases/Clases9899/Clase08/JavaAWTpart1/
- http://asignaturas.deusto.es/labinfii/Apuntes/Tema8Trans.pdf
- http://tejo.usal.es/~fgarcia/docencia/poo/02-03/trabajos/S2T4.pdf
- “Java en Acción”. Programación IV. Ing. Manuel Vega Ulloa.
- Java™ 2 Platform, Standard Edition, v 1.3.1 - API (Application Programming Interface) Specification.