CENTRO DE INVESTIGACIÓN
EN COMPUTACIÓN

Java Avanzado

Autor

Oscar Alejandro González Bustamante.

Lenguaje de Programación Java.

Interfaces Gráficas de Usuario con Java.


Interfaces Gráficas de Usuario con Java.

AWT y Swing.

Las AWT y SWING proveen componentes básicos para una GUI (Graphics User Interface - Interface Gráfica de Usuario) y son utilizados en las aplicaciones y los applets de Java. Una de las ventajas de usar AWT es que la interface es independiente de la plataforma o interface gráfica de la máquina. Esto nos asegura que lo que se vea en una computadora aparecerá igual en otra computadora.

Una estrategia para estudiar las AWT es dividirlas en :

- Componentes
- Contenedores
- Layouts (Administradores de diseño)
- Eventos

Componentes. Son clases o interfaces que permiten crear los objetos gráficos que componen una GUI tales como;botones, listas desplegables, cuadros de texto, casillas de verificación, botones de opción, campos de texto, etiquetas, menús, etc.

Contenedores. Son clases o interfaces que permiten crear objetos gráficos para contener a los componentes, tales como; paneles, cuadros de diálogo, marcos, ventanas, etc.

Layouts. Son clases o interfaces que permiten crear objetos de que administren el diseño, la distribución y colocación de los objetos componentes dentro de los objetos contenedores. Por ejemplo el **FlowLayout** distribuye los componentes de izquierda a derecha y de arriba a abajo, el **BorderLayout** los distribuye en cinco áreas geográficas; norte, sur, este, oeste y centro, etc.

Eventos. Son las clases o interfaces que permiten crear objetos que capturen y manejen los eventos. Un evento es una acción sobre algún componente, por ejemplo, clic a un botón, pulsar la tecla de enter en un botón, mover un elemento con las teclas de navegación, eventos especiales como los programados por tiempo, etc. Sin los eventos las GUI serían interfaces gráficas sin vida, y por lo tanto no serían muy útiles que digamos.

Cuando se empieza a utilizar SWING, se observa que Sun ha dado un paso importante adelante respecto al AWT. Ahora los componentes de la interface gráfica son Beans y utilizan el nuevo modelo de Delegación de Eventos de Java. Swing proporciona un conjunto completo de componentes, todos ellos *lightweight*, es decir, ya no se usan componentes *peer* dependientes del sistema operativo, ademas, SWING está totalmente escrito en Java. Todo esto conlleva una mayor funcionalidad en manos del programador, y en la posibilidad de mejorar en gran médida la apariencia cosmética de las interfaces gráficas de usuario.


Hay muchas ventajas que ofrece el SWING. Por ejemplo, la navegación con el teclado es automática, cualquier aplicación SWING puede utilizarce sin ratón, sin tener que escribir una línea de código adicional. Las etiquetas de información o "*tool tips*", se pueden crear con una sola línea de código. Además, con SWING la apariencia de la aplicación se adapta dinámicamente al sistema operativo y plataforma en que esté corriendo.

Los componentes Swing no soportan el modelo de Eventos de Propagación, sino solamente el modelo de eventos de Delegación incluido desde la versión JDK 1.1; por lo tanto si se van a utilizar componentes SWING, debe programar exclusivamente en el nuevo modelo, o dicho de otro forma, se recomienda al programador construir programas GUI mezclando lo menos posible SWING con AWT.

Los componentes

Los componentes son clases de objetos que permiten utilizar elementos gráficos para crear interfaces gráficas y están divididos en dos grandes grupos: los ***contenedores*** y los ***componentes***. Un componente, también denominado componente simple o atómico, es un objeto que tiene una representación gráfica, que se puede mostrar en la pantalla y con la que puede interactuar el usuario. Ejemplos de componentes son los botones, campos de texto, etiquetas, casillas de verificación, menús, etc.,.
En los lenguajes de Programación Orientada a Objetos como Java tenemos dos paquetes o conjuntos de clases principales agrupados en el paquete llamado AWT (Abstract Windows Tolkit) y en el paquete SWING, algunos de estos componentes del paquete AWT y del SWING están resumidos en la siguiente tabla 11-1:

Tipo de Componente		**Descripción**
AWT	**SWING**	
Button	JButton	Es un botón usado para recibir el clic del ratón
Canvas		Un lienzo o panel usado para dibujar
Checkbox	JCheckBox	Cuadro de verificación. Es un componente que le permite seleccionar un elemento
Choice	JComboBox	Es una lista desplegable de elementos estáticos.
Component		Es el padre de todos los componentes AWT, excepto de los componentes de tipo menú
Container		Es el padre de todos los contenedores
Dialog	JDialog	Es un cuadro de diálogo oon una ventana con titulo y bordes.
Frame	JFrame	Es un marco o ventana y es la clase base de todas las ventanas GUI con controles para ventana.
Label	JLabel	Etiqueta. Es una cadena de texto como componente.
List	JList	Un componente que contiene un conjunto dinámico de elementos.
Menu	JMenu	Es un elemento dentro de la barra de menú, el cual contiene un conjunto de elementos de tipo menú.
MenuItem	JMenuItem	Un elemento dentro de un menú.
Panel	JPanel	Una clase contenedora básica usado frecuentemente para crear diseños (layouts) complejos.
Scroollbar	JScrollBar	Un componente que permite al usuario hacer una



		selección dentro de un rango de valores.
ScroolPane	JScrollPane	Una clase contenedora que implementa un deslizador horizontal y vertical para un único componente hijo
TextArea	JTextArea	Un componente que permite al usuario introducir texto en un bloque o rectángulo.

Tabla 11-1 Resumen de algunas clases componentes del paquete java.awt y javax.swing

Contenedores.

Un contenedor es un componente al que se le incluirán uno o mas componentes o contenedores. Los contenedores permiten generar la estructura de una ventana y el espacio donde se muestra el resto de los componentes contenidos. En ella y que conforman la interfaz de usuario. Los contenedores de alto nivel mas utilizados se muestran en la siguiente tabla 11.2:

Clase	Utilidad
javax.swing.JFrame	Proporciona una ventana principal de aplicación con su funcionalidad normal (p.ej. Borde, título, menús) y un panel de contenido
javax.swing.JDialog	Proporciona una ventana de diálogo auxiliar en una aplicación, normalmente utilizada para pedir datos al usuario o configurar elementos.
javax.swing.JApplet	Implementa una ventana que aparece dentro de otra aplicación que normalmente es un navegador de internet.

Tabla 11-2. Clases contenedoras de alto nivel.

Jerarquía y tipos de los componentes gráficos.

Las clases gráficas se presentan organizadas en los siguientes grandes grupos:

- Clases básicas.
- Contenedores de alto nivel.
- Contenedores intermedios.
- Componentes atómicos.

Clases básicas.

Proporcionan el soporte y funcionalidad básica para el resto de los componentes. La raíz de todos los elementos gráficos en Java es la clase abstracta **java.awt.Component** y su subclase abstracta **java.awt.Container**. La clase **javax.swing.JComponent**, es una subclase de **java.awt.Container** y que es la clase base para prácticamente todos los componentes SWING.

Contenedores de alto nivel.


Los Frames

Se emplea para crear una ventana principal de una aplicación. Esta ventana posee un marco que incluye los controles de minimizar, restaurar/maximizar y cerrar. A este contenedor se le puede agregar una barra de menús (**javax.swing.JMenuBar**). Los componentes gráficos no se añaden directamente al **JFrame** sino a su panel de contenido. También el administrador de diseño (Layout) se le debe aplicar a este panel de contenido. Los métodos mas frecuentemente utilizados en un **JFrame** son:

```
f.setSize(420,450);  // tamaño de la ventana
f.setLocation(50,50); // posición de la esquina superior izquierda de la ventana
f.getContentPane().setLayout(objetoLayout); // establece el administrador de diseño
f.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE); // cerrar ventana
f.setTitle("Operadores de Java"); // pone texto en el título de la ventana
f.setJMenuBar(barraDeMenu); // agrega una barra de menú a la ventana
f.setVisible( true );  // hace visible o invisible la ventana
f.pack(); // asigna tamaño iniciales a los componentes
f.getContentPane().add(componente); // añade un componente a la ventana
f.addXXXXXListener(objetoListener ); // añade un oyente de eventos
```

El siguiente ejemplo utiliza algunos de los métodos anteriormente discutidos para el JFrame creando un objeto f de la clase **JFrame** en el método **public static void main(String [] args)**. Los comentarios que acompañan el código son explicativos de lo que se quiere hacer. Se le agregan dos eventos de ratón (**mouseEntered** y **mouseExited**) para cuando aparezca la ventana y se coloque el puntero de ratón por encima de la ventana y o cuando se coloque fuera de la misma, se escribirá un mensaje. Vea la figura 11.1.

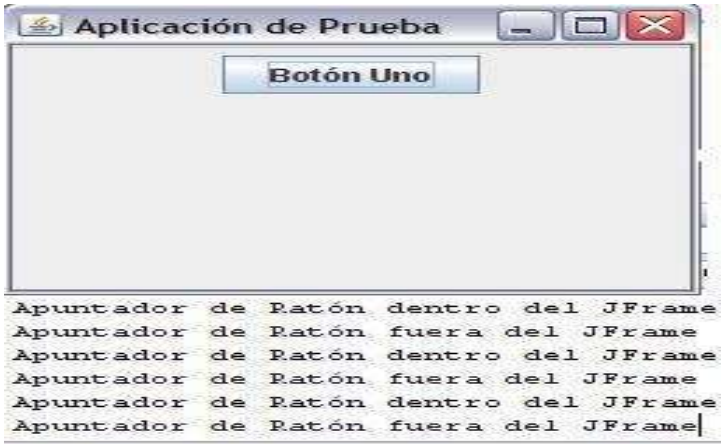


Figura 11-1. Ejecución del programa EjemploJFrame.java

Código del ejemplo de JFrame y sus métodos:

```
package ejemploswing;
import javax.swing.JFrame;
import javax.swing.JMenuBar;
import java.awt.FlowLayout;
import javax.swing.JButton;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
```

```
public class EjemploJFrame {

public static void main( String[] args ) {
   JFrame f = new JFrame();
   f.setLocation(50,50); // posición de la esquina superior izquierda de la ventana
   FlowLayout objetoLayout = new FlowLayout(); // administrador de diseño
   f.getContentPane().setLayout(objetoLayout); // establece el administrador de
diseño
   f.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE); //cerrar
ventana
   f.setTitle("Aplicación de Prueba"); // pone texto en el título de la ventana
   JMenuBar barraDeMenu = new JMenuBar(); // crea un objeto de tipo barra de menú
   f.setJMenuBar(barraDeMenu); // agrega una barra de menú a la ventana
   JButton boton1 = new JButton("Botón Uno"); // crea un botón
   f.getContentPane().add(boton1); // añade un componente a la ventana
   // añade un oyente al JFrame y lo maneja con una adaptador
  // para el Mouse cuando entre a la forma o cuando salga
   // lo hace todo mediante una clase interna anónima
  f.addMouseListener(new java.awt.event.MouseAdapter() {
              public void mouseEntered(java.awt.event.MouseEvent evt) {
                   formMouseEntered(evt); // invocación metodo delegado
              }
              public void mouseExited(java.awt.event.MouseEvent evt) {
                   formMouseExited(evt); // invocación método delegado
              }
         });

   f.pack(); // asigna tamaño iniciales a los componentes
   f.setVisible( true );  // hace visible o invisible la ventana
   f.setSize(250,200);  // tamaño de la ventana
 }// fin del main

// metodo delegado
    private static void formMouseExited(java.awt.event.MouseEvent evt) {
       // agrega tu codigo de manejo del evento aquí:
          System.out.println("Apuntador de Ratón fuera del JFrame");
     } // fin del método formMouseExited
// metodo delegado
    private static void formMouseEntered(java.awt.event.MouseEvent evt) {
      // agrega tu código de manejo del evento aquí:
          System.out.println("Apuntador de Ratón dentro del JFrame");
     } // fin del método formMouseEntered

}// fin de la clase EjemploJFrame
```

JDialog

La clase **javax.swing.JDialog** es la clase raíz de las ventanas secundarias que implementan cuadros de diálogo en SWING. Estas ventanas dependen de una ventana principal (o con marco, normalmente de la clase **JFrame**) y si la ventana principal se cierra, se iconiza o desiconiza, las ventanas secundarias realizan la misma operación de forma automática. Estas ventanas pueden ser modales o no modales, es

decir, limitan la interacción con la ventana principal si así se desea. El constructor más frecuentemente utilizado es:

```
// crea un Jdialog con la ventana de la que depende, un título y si es modal o no.
JDialog dialogo = new JDialog( frame, "Titulo del Jdialog", true );
```

La clase **JDialog** puede utilizar todos los métodos descritos para la clase **JFrame**.

Cuadros de diálogo estándar con la clase javax.swing.JOptionPane.

Esta clase se utiliza para crear los tipos de cuadros de diálogo más habituales, como los que permiten pedir un valor, mostrar un mensaje de error o advertencia, solicitar una confirmación, etc. Todos los cuadros de diálogo que implementa **JOptionPane** son modales (esto es, boquean la interacción del usuario con otras ventanas). La forma habitual de uso de la clase es mediante la invocación de alguno de sus métodos estáticos para crear cuadros de diálogo. Tienen el formato **show<Tipocuadro>Dialog**, donde el tipo puede ser:

- Message para informar al usuario con un mensaje, vea figura 11-2

```
//cuadro de mensaje
JOptionPane.showMessageDialog(ventana,
                "No se ha podido conectar a la base de datos", //mensaje
                "Error en conexión", // título
                JOptionPane.ERROR_MESSAGE); // icono
```

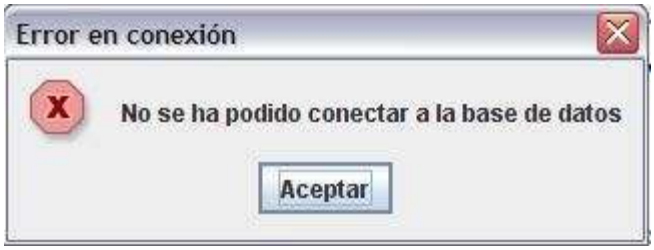


Figura 11-2. Cuadro de diálogo estandar de Mensaje

- Confirm para solicitar una confirmación al usuario con las posibles respuestas de si, no o cancelar, ver figura 11-3.

```
// cuadro de confirmación. Devuelve YES_OPTION , NO_OPTION
int confirma = JOptionPane.showConfirmDialog(ventana,
                "¿Desea cerrar el cuadro de Selección de archivos", //mensaje
                "Confirmar cerrar", // título
                JOptionPane.YES_NO_CANCEL_OPTION, //botones
                JOptionPane.INFORMATION_MESSAGE); // icono

if( confirma == JOptionPane.YES_OPTION ) // se compara confirmación
            System.exit( 0 );
```

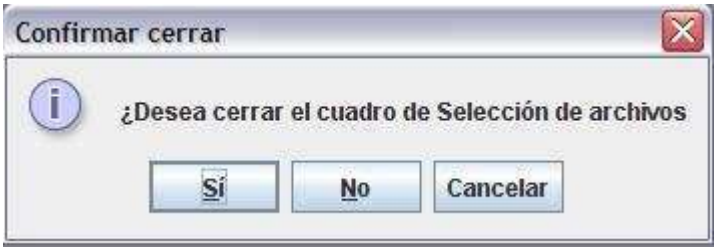



Figura 11-3. Cuadro de diálogo estandar de confirmación.

- Input para solicitar la entrada de un datos, ver figura 11-4.

```
//cuadro de entrada. Devuelve la cadena introducida o null si se cancela
String cadena = JOptionPane.showInputDialog(ventana,
                          "Dar N = ",  // mensaje
                          "Solicitud de un número", //título
                          JOptionPane.QUESTION_MESSAGE); // icono
int n = Integer.parseInt(cadena);  // conversion de cadena a entero
```

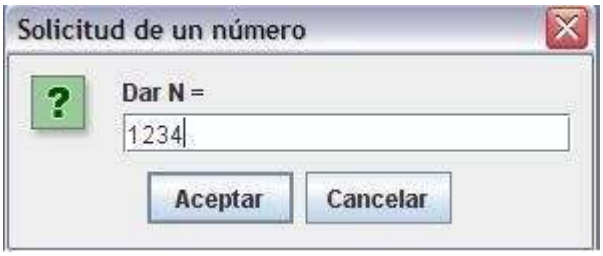


Figura 11-4. Cuadro de diálogo estándar de entrada.

- Option permite crear una ventana personalizada de cualquiera de los tipos anteriores, ver figura 11-5.

```
//Cuadro de opción personalizado, devuelve en  n el botón pulsado
String[] perrosFamosos = { "Snoopy","Super Can","Fido","Patan","Tribilín",
                           "Scoobe Doo","Pluto","Rintintín","Lasie" };

    int n = JOptionPane.showOptionDialog(null,
                "¿Cual es su perro favorito?", //mensaje
                "Mensaje de confirmación", //título
                JOptionPane.YES_NO_CANCEL_OPTION, // botones
                JOptionPane.QUESTION_MESSAGE,
                null,  // utiliza icono predeterminado
                perrosFamosos,
                perrosFamosos[0]); // botón determinado
       System.out.println( "Mi perro favorito es: " + perrosFamosos[n] );
```

Cuadros de diálogo estándar con la clase javax.swing.JFileChooser.

La clase **JFileChooser** tiene métodos como el **showOpenDialog(ventana)** y el **showSaveDialog(ventana)** que permiten la elección interactiva de un archivo o directorio, vea figura 11-6 y figura 11-7.

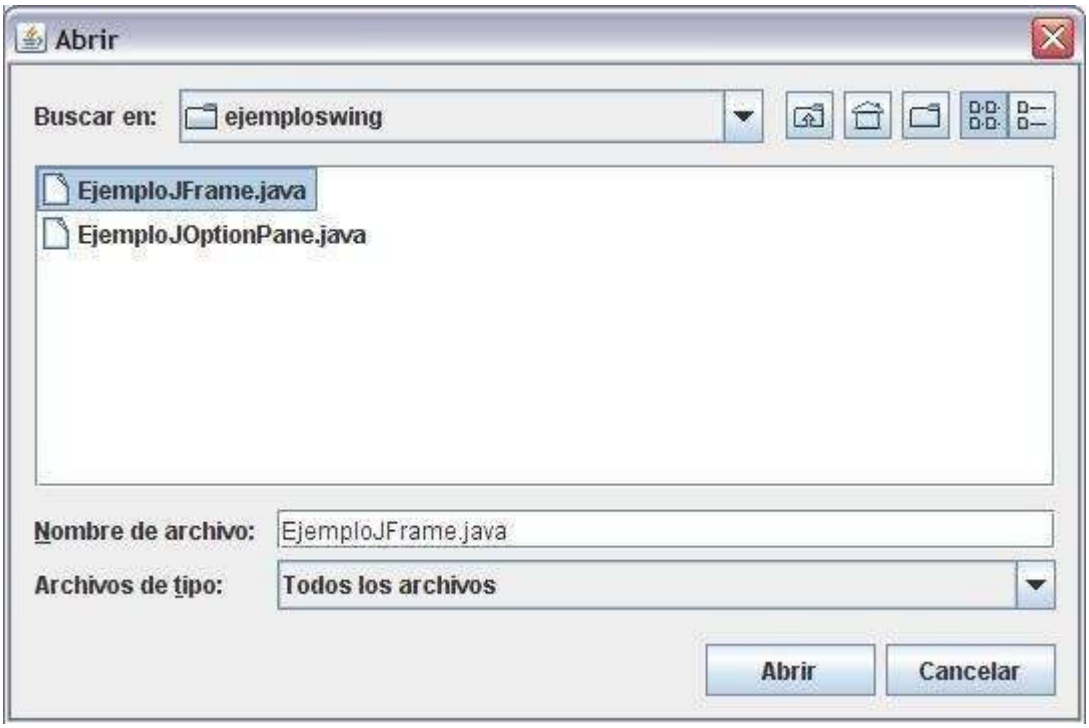


Figura 11-6. Cuadro de diálogo Abrir que produce el método showOpenDialog(ventana) de la clase javax.swing.JFileChooser.

Clase **EjemploFileChooser.java** que muestra el uso de un cuadro de diálogo de Abrir archivos.

```
import javax.swing.JFileChooser;
import java.io.File;

public class EjemploFileChosser {

    public static void main( String[] a ) {
        File fent= null;
        // se crea el selector de archivos
        JFileChooser selector = new JFileChooser();
        //solo posibilidad de seleccionar directorios
        selector.setFileSelectionMode( JfileChooser.FILES_AND_DIRECTORIES);
        //se muestra y se espera a que el usuario acepte o cancele la selección
        int opcion = selector.showOpenDialog(null); // cuadro Abrir archivo
        if ( opcion == JFileChooser.APPROVE_OPTION) {
            // se obtiene el archivo o directorio seleccionado
            fent = selector.getSelectedFile(); // devuelve un objeto File
            System.out.println("Nombre archivo:" + fent.getName() + "\n" +
              "Directorio padre: " + fent.getParent() + "\n" +
              "Ruta: " + fent.getPath() );
```

```
        }// fin del if
    }// fin del main
}// fin de la clase EjemploFileChosser
```

Clase **EjemploFileChooserSaveDialog.java** que muestra el uso de un cuadro de diálogo de Guardar archivos.

Figura 11-7. Cuadro de diálogo Guardar que produce el método showSaveDialog(ventana) de la clase javax.swing.JFileChooser.

```
import java.io.File;
import javax.swing.JFileChooser;

public class EjemploFileChooserSaveDialog {

     public static void main( String[] a ) {
        File fsal= null;
        // se crea el selector de archivos
        JFileChooser selector = new JFileChooser();
        //solo posibilidad de seleccionar directorios
        selector.setFileSelectionMode( JFileChooser.FILES_AND_DIRECTORIES);
        //se muestra y se espera a que el usuario acepte o cancele la selección
        int opcion = selector.showSaveDialog(null); // cuadro de Guardar archivo
        if ( opcion == JFileChooser.APPROVE_OPTION) {
            // se obtiene el archivo o directorio seleccionado
            fsal = selector.getSelectedFile(); // devuelve un objeto File
            System.out.println("Nombre archivo:" + fsal.getName() + "\n" +
              "Directorio padre: " + fsal.getParent() + "\n" +
              "Ruta: " + fsal.getPath() );
        }
```



```
        }// fin del main
    } // fin de la clase  EjemploFileChooserSaveDialog
```

Eventos.

Son las clases o interfaces que permiten crear objetos que capturen y manejen los eventos. Un evento es una acción sobre algún componente, por ejemplo, clic a un botón, pulsar la tecla de Enter en un botón, mover un elemento con las teclas de navegación, eventos especiales como los programados por tiempo, etc. Sin los eventos las GUI serían interfaces gráficas sin vida, y por lo tanto no serían muy útiles que digamos.
A continuación examinaremos un ejemplo en el lenguaje de programación Java sobre esto. Los eventos son objetos que describen que ha sucedido. Hay diferentes clases de eventos para describir diferentes categorías de acciones por parte del usuario.

Fuentes de eventos

Una fuente de un evento es el generador de un evento, así por ejemplo, el clic del ratón sobre un componente botón genera un **ActionEvent** con el botón como origen o fuente del evento, ver figura 11-2. La instancia de un **ActionEvent** es un objeto que contiene información acerca de los eventos que acaban de darse. Este contiene:

getActionCommand() - Devuelve el nombre del comando asociado con la acción.

getModifiers() - Regresa cualquier modificador que se haya dado durante la acción.

Manejadores de eventos

Un *manejador de evento* es un método que recibe un objeto de tipo evento y decide y procesa la interacción con el usuario.

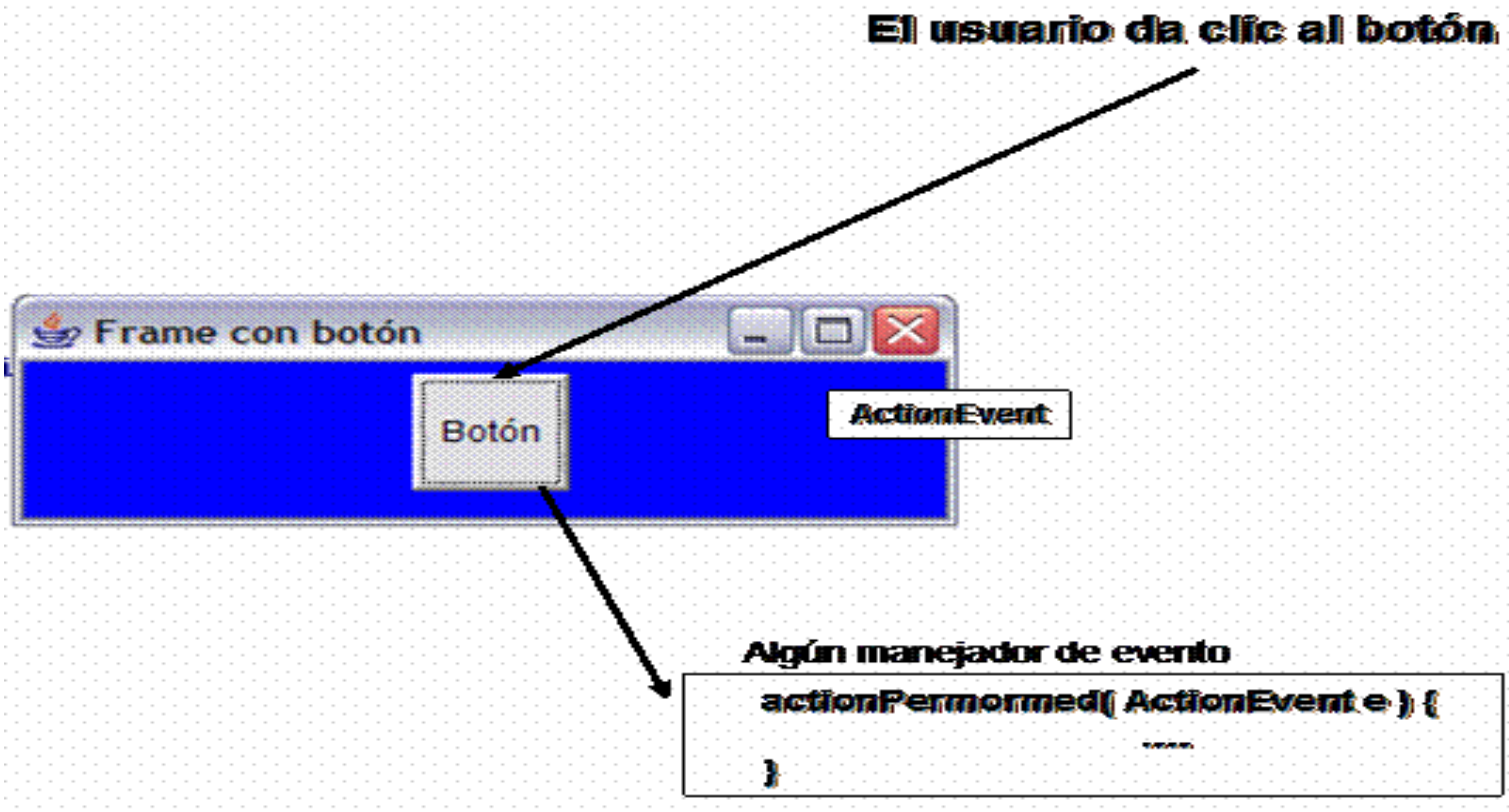


Figura 11-2. fuentes y manejadotes de eventos.

El modelo de delegación de eventos

Con este modelo los eventos son enviados a el componente desde donde el evento fue originado, pero cada componente propaga el evento a un o mas clases llamadas *oyentes* (*listeners*).

Los oyentes contienen *manejadores de eventos* que reciben y procesan el evento, ver figura 11-3. De esta forma, el manejador del evento puede ser un objeto separado del componente. Los oyentes son clases que implementan la interface **EventListener**.

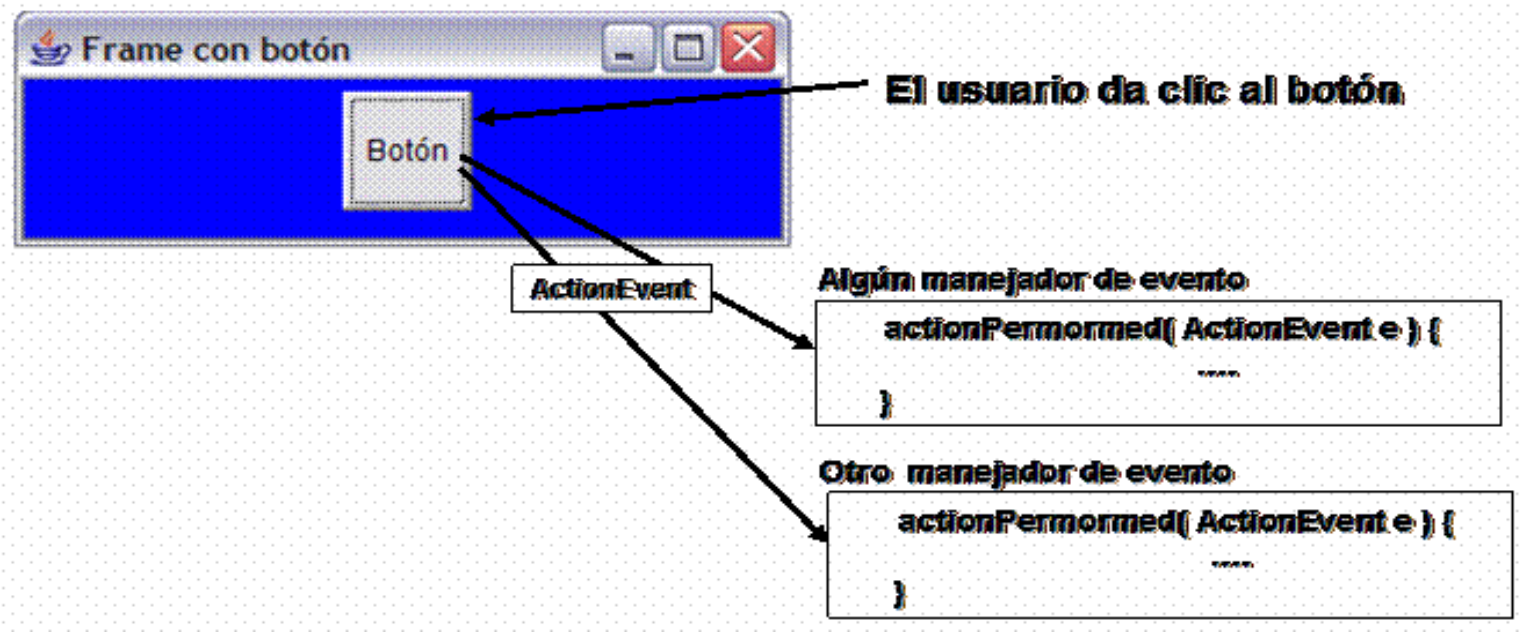


Figura 11-3. Delegación de eventos.

Los eventos son objetos que reportan solamente a los oyentes registrados. Cada evento tiene una interface oyente correspondiente que le indica cuales son los métodos adecuados que deben ser definidos dentro de la clase para recibir tales tipos de eventos. La clase que implementa la interface define todos esos métodos y pueden ser registrados como un oyente.

Los componentes que no tienen oyentes registrados no son propagados.

Por ejemplo, veamos el siguiente código de un simple Frame con un simple botón:

```
1. import java.awt.*;
2. import java.awt.event.*;
3.
4. /**
5. * <p>Título: PruebaBoton.java </p>
6. * <p>Descripción: Te enseña a usar delegación de eventos</p>
7. * <p>Copyright: Totalmente libre</p>
8. * <p>Empresa: El patito Feo Inc.</p>
9. * @author Oscar Alejandro González Bustamante
10.* @version 1.0
11.*/
12.
13.public class PruebaBoton
14.extends Frame {
15.Button button1 = new Button();
16.FlowLayout flowLayout1 = new FlowLayout();
17.
18.public PruebaBoton() {
19.try {
```

```
20.jbInit();
21.}
22.catch (Exception ex) {
23.ex.printStackTrace();
24.}
25.}
26.
27.void jbInit() throws Exception {
28.button1.setLabel("Botón");
29.button1.setActionCommand("¡Dame clic y verás que bonito!");
30.// registrar un oyente al botón
31.button1.addActionListener(new PruebaBoton_button1_actionAdapter(this));
32.
33.this.setBackground(Color.blue);
34.this.setTitle("Frame con botón");
35.this.setLayout(flowLayout1);
36.this.add(button1, null);
37.}
38.
39.public static void main(String[] args) {
40.PruebaBoton pruebaBoton = new PruebaBoton();
41.pruebaBoton.setSize(300, 100);
42.pruebaBoton.setLocation(300, 200);
43.pruebaBoton.setVisible(true);
44.pruebaBoton.button1.setSize(50, 50);
45.}
46.
47.void button1_actionPerformed(ActionEvent e) {
48.System.out.println("" + e.getActionCommand());
49.}
50.} // fin de la clase PruebaBoton
51.
52.// La clase PruebaBoton_button1_actionAdapter  es la clase manejadora en la
53.// cual el evento es delegado.
54.class PruebaBoton_button1_actionAdapter
55.implements java.awt.event.ActionListener {
56.PruebaBoton adaptee;
57.
58.PruebaBoton_button1_actionAdapter(PruebaBoton adaptee) {
59.this.adaptee = adaptee;
60.}
61.
62.public void actionPerformed(ActionEvent e) {
63.adaptee.button1_actionPerformed(e);
64.}
65.} // fin de la clase PruebaBoton_button1_actionAdapter
```

La figura 11-4 muestra la ejecución del programa anterior en JBuilder X (versión de prueba). Cuando el usuario le da clic al botón, se dispara el evento y escribe en la

ventana de mensajes " ¡Dame clic y verás que bonito!”.

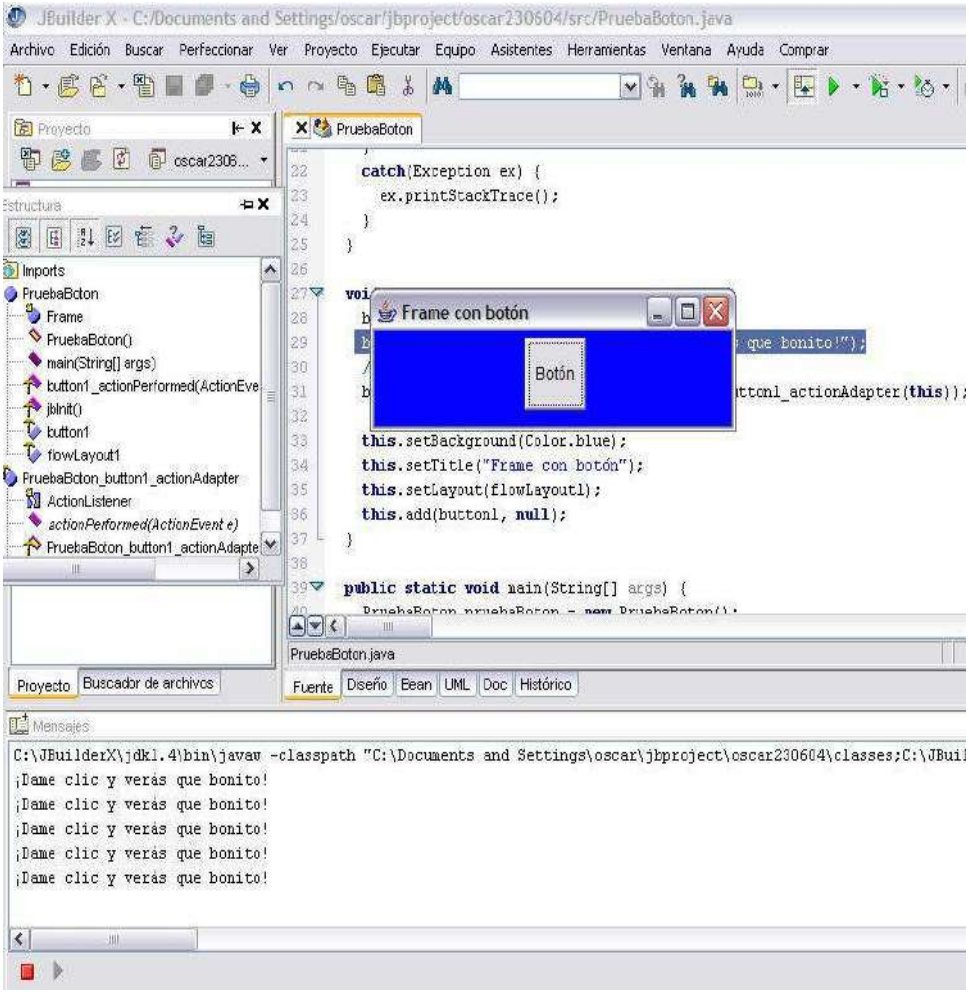


Figura 11-4. Salida del programa PruebaBoton.java.

Este ejemplo tiene las siguientes características:

La clase **Button** tiene un método **addActionListener (ActionListener)**

La interface **ActionListener** define un método **actionPerformed**, el cual recibe un **ActionEvent**.

Una vez creado, un objeto **Button** puede tener un objeto registrado como un oyente (listener) para el **ActionEvents** a través del método **addActionListener()**. El oyente registrado es instanciado desde una clase que implementa la interface **ActionListener**.

Cuando al el objeto **Button** se le da clic con el ratón, un **ActionEvent** es enviado. El **ActionEvent** es recibido a través de el método **actionPerformed()** para cualquier **ActionListener** que sea registrada sobre el objeto **Button** a través de su método **addActionListener()**.

El método **getActionCommand()** de la clase **ActionEvent** regresa el nombre del comando asociado con esta acción. En la línea 29 la acción del comando para este

botón es establecida con "¡Dame clic y verás que bonito!".

Categorías de eventos

Los eventos no son accidentalmente manejados. Los objetos que quieren ser oídos por un evento en particular sobre un componente GUI deben registrarse a si mismos con tal componente.

Cuando un evento ocurre, solo los objetos que fueron registrados recibirán el mensaje del evento ocurrido.

El modelo de delegación es bueno por la distribución del trabajo entre las clases.

Los eventos no tienen por que estar relacionados a un componente AWT. Este modelo de eventos provee soporte para JavaBeans.

La siguiente figura 11-5 muestra las categorías de los eventos.

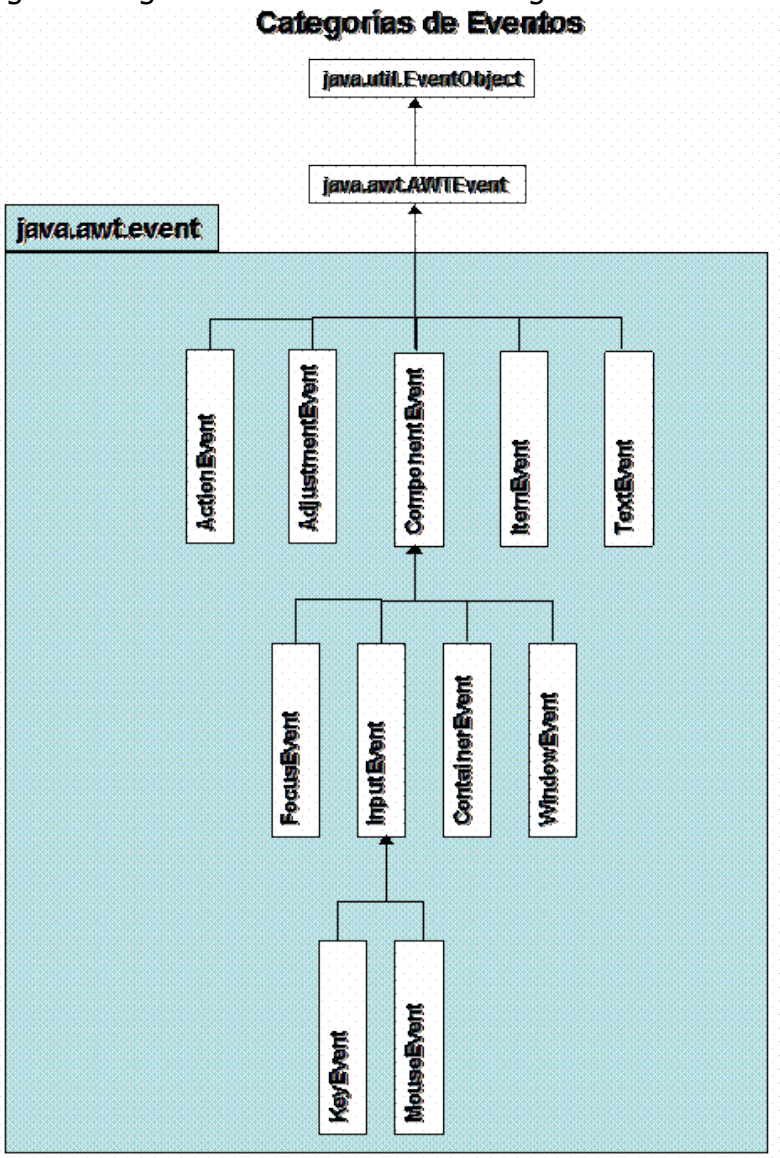


Figura 11-5 . Clases del java.awt.event

El mecanismo general para recibir eventos de los componentes ha sido descrito en el contexto de un simple tipo de evento. Muchas de las clases de eventos están en el paquete **java.awt.event**, pero otras existen en cualquier otra parte en el API.

Para cada categoría de eventos, hay una interface que tiene que ser implementada por la clase de objetos que quieren recibir los eventos. Esta interface demanda que uno o mas métodos sean definidos. Aquellos métodos son llamados cuando un suceso o evento en particular surge. La siguiente tabla 11-2 lista estas interfaces por categorias.

Tabla 11-2 Interfaces por categorías para manejo de eventos.

Categoría	Nombre de la Interface	Métodos
Action	ActionListener	actionPerformed(ActionEvent)
Item	ItemListener	itemStateChanged(ItemEvent)
Mouse	MouseListener	mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MourseEvent) mouseClicked(MouseEvent)
Mouse Motion	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
Key	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
Focus	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
Adjustment	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)
Component	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
Window	WindowListener	windowClosing(WindowEvent) windowOpenend(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
Container	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
Text	TextListener	textValueChanged(TextEvent)

Implementando múltiples interfaces

Los oyentes de eventos del paquete AWT permiten múltiples oyentes que pueden ser invocados por el mismo componente. Es posible programar código para manejar múltiples eventos en un solo método que lo maneje. Sin embargo, a veces un diseño dentro de una aplicación requiere muchas partes no relacionadas en el mismo programa para reaccionar a el mismo evento. Esto puede suceder si, por ejemplo, un sistema de ayuda sensible a contexto es agregado a un programa existente.

El mecanismo oyente permite agregar llamadas a un método **addXxxListener()** tantas veces como sea necesario, y usted puede especificar muchos diferentes oyentes como su diseño lo requiera. Todos estos oyentes registrados tienen sus métodos manejadores que los invocan cuando un evento suceda.

Adaptadoras de Eventos

Es mucho trabajo implementar todos los métodos en cada una de las interfaces oyentes, particularmente para el caso de la interface **MouseListener** y **WindowListener**.

Por ejemplo, la interface **MouseListener** declara los métodos siguientes:

- `public void mouseClicked( MouseEvent evento )`
- `public void mouseEntered( MouseEvent evento )`
- `public void mouseExited( MouseEvent evento )`
- `public void mousePressed( MouseEvent evento )`
- `public void mouseReleased( MouseEvent evento )`

Como una conveniencia, el lenguaje de programación Java provee de clases adaptadoras que implementan cada interface conteniendo mas de un método. Los métodos en estas clases adaptadoras están vacios.

Así, las clases oyentes que usted define pueden extenderse en clases adaptadoras y sobrescribir solo los métodos que usted necesite.

Ejemplo:

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

/**
  * Programa de Java que te enseña a utilizar clases adaptadoras
  * del paquete java.awt. Este demuestra el uso de los
  * multiples eventos de la clase MouseEvent  con
```



```
 * la clase adaptadora MouseAdapter
 * @autor Oscar A. González Bustamante
 * @version 1.0
 * Archivo: ManejadorClicRaton.java
 */

public class ManejadorClicRaton  extends MouseAdapter {

     // necesitamos el manejador mouseClick, asi que usamos
     // entonces el adaptador para evitar tener  que escribir  todos los
     //  métodos manejadores de eventos

 public  void  mouseClicked( MouseEvent  e ) {
    // Aquí programamos el evento del clic del ratón
   }
} // fin de la clase  ManejadorClicRaton
```

Hay que notar que esto es una clase, no una interface. Esto significa que es posible extender solamente a otra clase. Debido a que los oyentes son interfaces, usted puede implementar muchas otras mas clases.

Manejo de eventos usando clases internas

Es posible implementar manejadores de eventos como clases internas (como se ve en la línea 27, 34-39) . Esto permite el acceso a los datos privados de las clases externas (línea 38). vea figura 11-6. Por ejemplo:

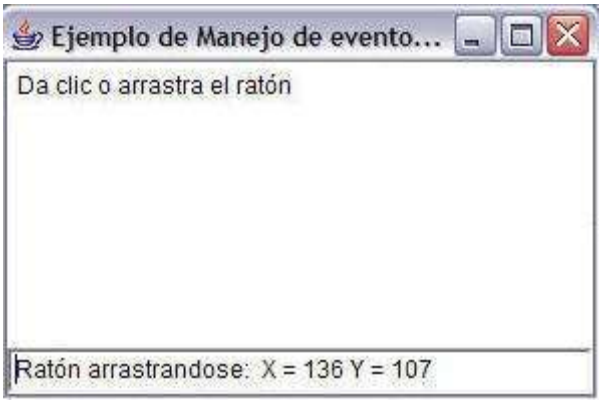


Figura 11-6. Eventos de ratón con clases internas.

```
1. import java.awt.*;
2. import java.awt.event.*;
3.
4. /**
5.   * Programa de Java que te enseña a utilizar clases adaptadoras pero como
6.   * clases internas.
7.   * @autor Oscar A. González Bustamante
8.   * @version 1.0
9.   * Archivo: PruebaInterna.java
10.  */
11.
12.public class PruebaInterna   {
13.      private  Frame  f;
14.      private  TextField  tf;
15.
16.    public  PruebaInterna() {
17.       f = new  Frame("Ejemplo de Manejo de eventos con clases internas");
18.       tf = new  TextField( 30 );
19.   }  // fin del constructor
20.
21. public  void  lanzaFrame() {
22.    Label   etiqueta  =  new  Label(" Da clic o arrastra el ratón ");
23.    // agrega componentes al Frame
24.    f.add( etiqueta, BorderLayout.NORTH );
25.    f.add( tf,  BorderLayout.SOUTH );
26.    // agrega un oyente que use una clase interna
27.    f.addMouseMotionListener( new  MiRatonseMueveOyente() );
28.    f.addMouseListener( new  ManejadorClicRaton() );
```


```
29.     // Tamaño del frame y hacerlo visible
30.     f.setSize( 300, 200 );
31.     f.setVisible( true );
32. } // fin del método lanzaFrame()
33.
34.     class MiRatonseMueveOyente extends MouseMotionAdapter {
35.         public void mouseDragged( MouseEvent e ) {
36.             String s = "Ratón arrastrandose: X = " + e.getX() +
37.                                 " Y = " + e.getY();
38.              tf.setText ( s );
39.         } // fin del método mouseDragged()
40.       }  // fin de la clase interna adaptadora MiRatonseMueveOyente
41.
42.   public static void main( String[] argumentos ) {
43.         PruebaInterna obj = new PruebaInterna();
44.        obj.lanzaFrame();
45.     } // fin del main()
46.} // fin de la clase  PruebaInterna
```

Manejo de eventos usando clases internas anónimas

Es posible incluir la definición completa de una clase en el ámbito de una expresión. Esta situación define lo que es conocido como clases internas anónimas y crea la instancia de una de ellas. Las clases anónimas internas son generalmente utilizadas con el manejo de eventos en AWT ; vea figura 11-7. por ejemplo:

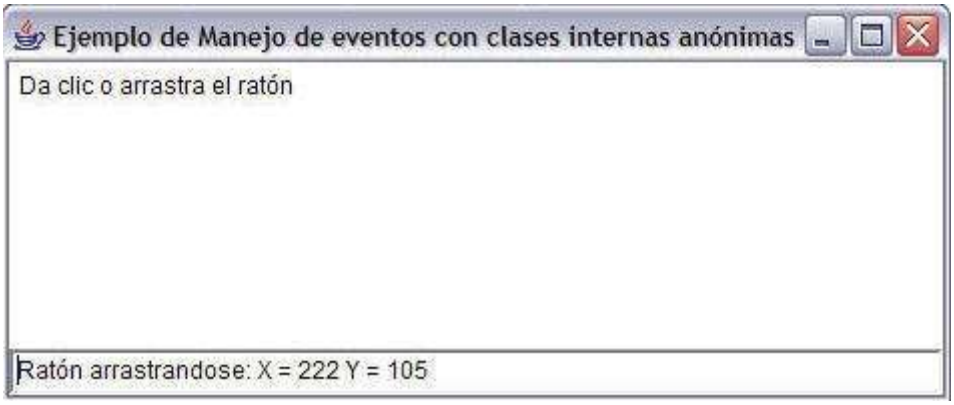


Figura 11-7. Clases internas anónimas para manejo de ratón.

```
1. import java.awt.*;
2. import java.awt.event.*;
3. /**
4.    * Programa de Java que te enseña a utilizar clases internas anónimas para
5.    * manejar eventos en las AWT
6.    * @autor Oscar A. González Bustamante
7.    * @version 1.0
8.    * Archivo: PruebaAnonima.java
9.    */
10.
11.public class PruebaAnonima   {
12.      private  Frame  f;
13.      private  TextField  tf;
14.
15.    public  PruebaAnonima() {
16.       f = new  Frame("Ejemplo de Manejo de eventos con clases internas
   anónimas");
17.       tf = new  TextField( 30 );
18.    }  // fin del constructor
19.
20. public  void  lanzaFrame() {
21.    Label   etiqueta  =  new  Label(" Da clic o arrastra el ratón ");
22.    // agrega componentes al Frame
23.    f.add( etiqueta, BorderLayout.NORTH );
24.    f.add( tf,  BorderLayout.SOUTH );
25.    // agrega un oyente que use una clase interna anónima
26.    f.addMouseMotionListener( new  MouseMotionAdapter()  {
27.           public  void  mouseDragged( MouseEvent  e ) {
28.               String  s = "Ratón arrastrandose: X = " + e.getX() +
29.                                  " Y = " + e.getY();
30.               tf.setText ( s );
31.          }
```



```
32.     } ); // <--  observe que hay que cerrar el paréntesis
33.     f.addMouseListener( new  ManejadorClicRaton() );
34.     // Tamaño del frame y hacerlo visible
35.     f.setSize( 300, 200 );
36.     f.setVisible( true );
37. } // fin del método lanzaFrame()
38.
39. public static void main( String[]  argumentos ) {
40.         PruebaAnonima   obj   =   new  PruebaAnonima();
41.         obj.lanzaFrame();
42.     } // fin del main()
43. } // fin de la clase  PruebaAnonima
```

Hay que hacer notar que la compilación de una clase anónima genera un archivo, el cual tiene el nombre de **PruebaAnonima$1.class** que es así para este caso en particular.


ANEXO A.

Ejemplos de componentes, contenedores y layouts con eventos.

JButton

En esta sección veremos ejemplos que hacen uso de los componentes, del paquete **java.awt** y **java.swing** los cuales puede escoger para crear su GUI. Como se dijo anteriormente en el punto 8.4 no es recomendable mezclar los componentes AWT y SWING , pero aquí nos interesa solo demostrar su uso y en algunos casos estan mezclados y en otros no.

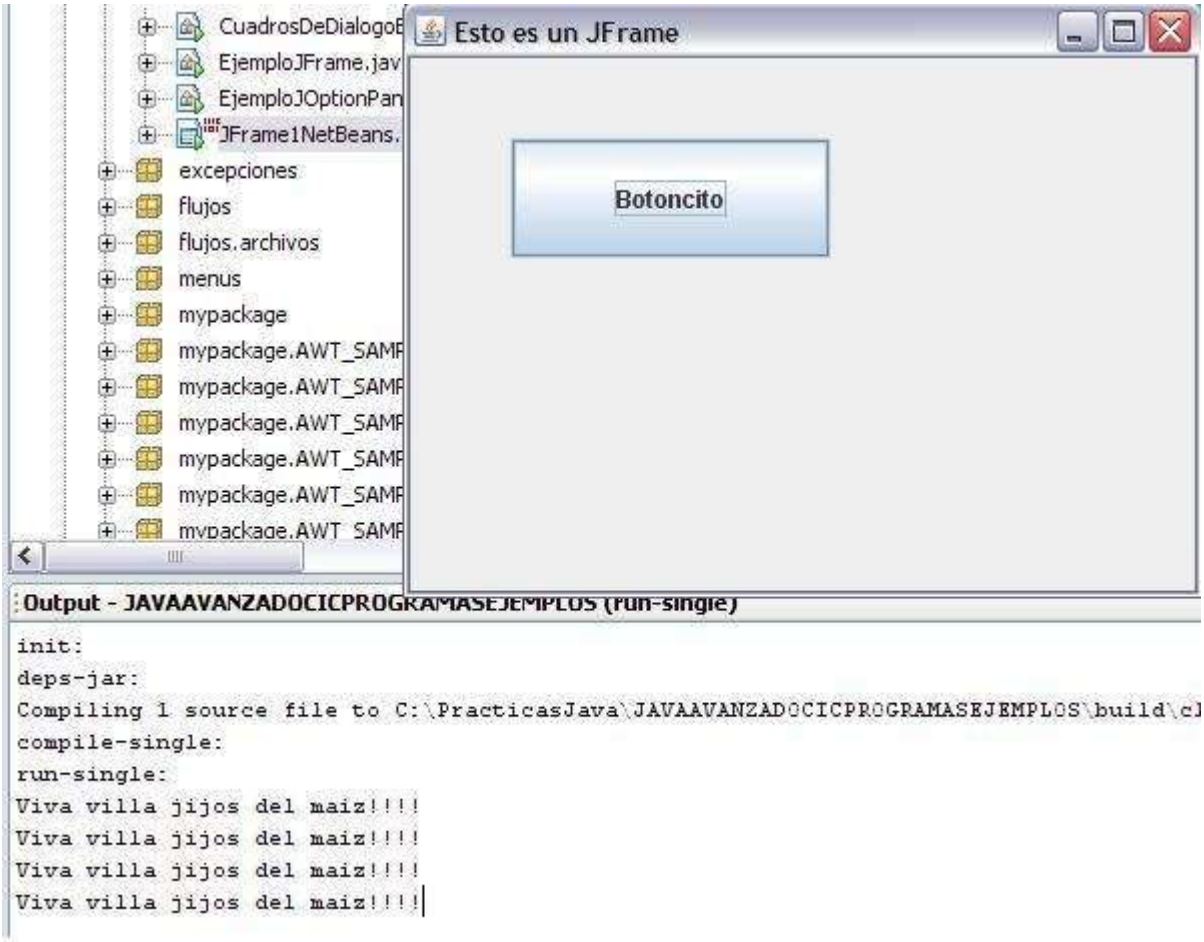
Ya nos es familiar que un objeto de la clase **JButton** sea un componente. Puede ser construído con una etiqueta de texto para informar al usuario sobre su uso. En el siguiente fragmento de código se crea un nuevo botón, se le da una etiqueta, una dimensión, y se le agrega un oyente mediante delegación de eventos, luego se agrega el componente al contenedor.

```
boton = new javax.swing.JButton();

getContentPane().setLayout(null);

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Esto es un JFrame");// título del JFrame
boton.setText("Botoncito");// etiqueta del botón
boton.addActionListener(new java.awt.event.ActionListener() { // agrega oyente
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        botonActionPerformed(evt); // delegación del evento
    }
});

getContentPane().add(boton);// se agrega el componente al contenedor
boton.setBounds(50, 40, 160, 60);// se da tamaño y ubicación al botón
```



El método **actionPerformed()** es el que implementa la interface **ActionListener**, la cual es registrada como un oyente, y el invocada cuando el bóton es presionado por un clic del ratón. La acción es delegada al método **butonActionPerformed()**

```
    private void botonActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        System.out.println("Viva villa jijos del maiz!!!!");
    }
```

La cadena "Viva Villa jijos del maiz!!!" pudo haberse establecido con el método:

```
boton.setActionCommand("Viva Villa jijos del maiz!!!");
```

y luego recuperado con el método

`e.getActionCommand()`

dentro del método

`botonActionPerformed()`

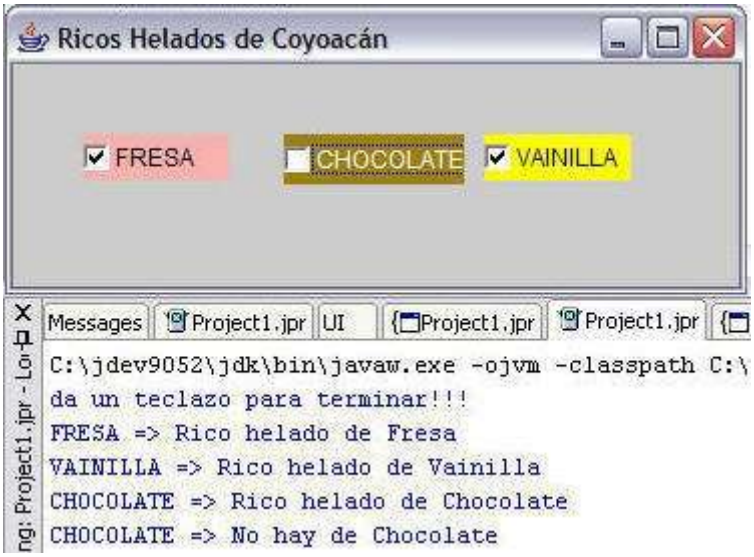
```
1. package ejemploswing;
2. /**
3.    * Programa de Java que te enseña a utilizar componentes
4.    * del paquete java.awt. Este demuestra el uso de los
5.    * objetos de la clase Button.
6.    * @autor Oscar A. González Bustamante
7.    * @version 1.0
8.    * Archivo: JFrame1NetBeans.java
9.    */
10.public class JFrame1NetBeans extends javax.swing.JFrame {
11.
12.    /** Creates new form JFrame1NetBeans */
13.    public JFrame1NetBeans() {
14.        initComponents();
15.        setSize(400,300);// tamaño del JFrame
16.        setLocation(200,200);// ubicación esquina superior izq.
17.    }
18.
19.    /** This method is called from within the constructor to
20.     * initialize the form.
21.     * WARNING: Do NOT modify this code. The content of this method is
22.     * always regenerated by the Form Editor.
23.     */
24.    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
25.    private void initComponents() {
26.        boton = new javax.swing.JButton();
27.
28.        getContentPane().setLayout(null);
29.
30.        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
31.        setTitle("Esto es un JFrame"); // título del JFrame
32.        boton.setText("Botoncito");// etiqueta del botón
33.        boton.addActionListener(new java.awt.event.ActionListener() { // agrega
   oyente
34.            public void actionPerformed(java.awt.event.ActionEvent evt) {
35.                botonActionPerformed(evt); // delegación del evento
36.            }
37.        });
38.
39.        getContentPane().add(boton);// se agrega el componente al contenedor
40.        boton.setBounds(50, 40, 160, 60); // se da tamaño y ubicación al botón
41.
42.        pack();
43.    }// </editor-fold>
44.
45.    private void botonActionPerformed(java.awt.event.ActionEvent evt) {
46.// TODO add your handling code here:
47.        System.out.println("Viva villa jijos del maiz!!!!");
```


```
48.     }
49.
50.     /**
51.      * @param args the command line arguments
52.      */
53.     public static void main(String args[]) {
54.         java.awt.EventQueue.invokeLater(new Runnable() {
55.             public void run() {
56.                 new JFrame1NetBeans().setVisible(true);
57.             }
58.         });
59.     }// fin del método main
60.
61.     // Variables declaration - do not modify
62.     private javax.swing.JButton boton;
63.     // End of variables declaration
64.
65.}// fin de la clase JFrame1NetBeans
```


CheckBox

El componente **Checkbox** del paquete **java.awt** nos provee de un simple dispositivo de encendido/apagado con una etiqueta de texto a un lado. Veamos el siguiente ejemplo:

Si se selecciona el **Checkbox** correspondiente desplegará el sabor del helado correspondiente, si se quita la marca de verificación o la palomita escribirá que no hay ese sabor correspondiente.




```
1. package mypackage1;
2. import javax.swing.JFrame;
3. import java.awt.Dimension;
4. import java.awt.Color;
5. import java.awt.Checkbox;
6. import java.awt.Rectangle;
7. import java.awt.event.ItemListener;
8. import java.awt.event.ItemEvent;
9. import java.io.*;
10./**
11.  * Programa de Java que te enseña a utilizar componentes
12.  * del paquete java.awt. Este demuestra el uso de los
13.  * objetos de la clase Checkbox.
14.  * @autor Oscar A. González Bustamante
15.  * @version 1.0
16.  * Archivo: RicosHelados.java
17.  */
18.public class RicosHelados extends JFrame
19.{
20.  private Checkbox fresa = new Checkbox();  // se crean los componentes
   Checkbox
```

```
21.  private Checkbox chocolate = new Checkbox();
22.  private Checkbox vainilla = new Checkbox();
23.
```

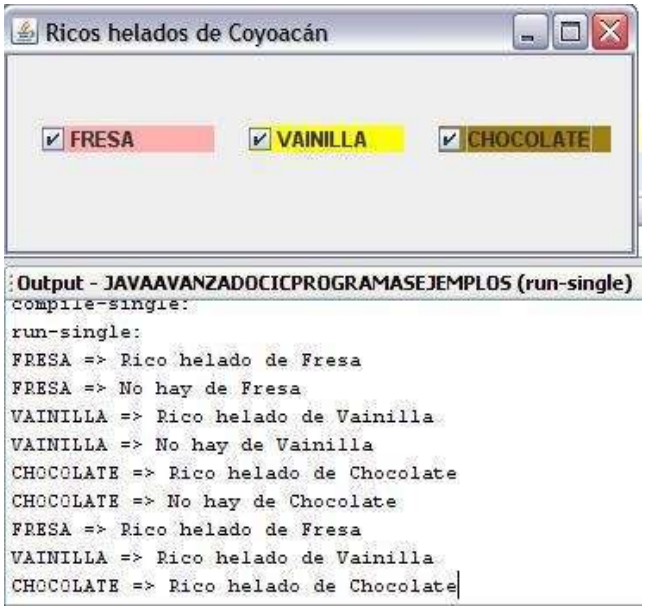
```
24.  public RicosHelados()
25.  {
26.    try
27.    {
28.      jbInit();
29.    }
30.    catch(Exception e)
31.    {
32.      e.printStackTrace();
33.    }
34.
35.  }
36.
37.  private void jbInit() throws Exception
38.  {
39.    this.getContentPane().setLayout(null);
40.    this.setSize(new Dimension(370, 145));
41.    this.setTitle("Ricos Helados de Coyoacán"); // titulo del JFrame
42.    this.setBackground(Color.cyan);
43.    fresa.setLabel("FRESA"); // se establecen la etiquetas
44.    fresa.setBounds(new Rectangle(35, 35, 73, 23)); // tamaño del componente
45.    fresa.setBackground(Color.pink); // color del fondo
46.    fresa.addItemListener(new ItemListener() // se agrega un oyente
47.      {
48.        public void itemStateChanged(ItemEvent e)
49.        {
50.          fresa_itemStateChanged(e);  // se delega el evento
51.        }
52.      });
53.    chocolate.setLabel("CHOCOLATE");
54.    chocolate.setBounds(new Rectangle(135, 35, 90, 25));
55.    chocolate.setForeground(Color.white);
56.    chocolate.setBackground(new Color(162, 126, 12));
57.    chocolate.addItemListener(new ItemListener()
58.      {
59.        public void itemStateChanged(ItemEvent e)
60.        {
61.          chocolate_itemStateChanged(e);
62.        }
63.      });
64.    vainilla.setLabel("VAINILLA");
65.    vainilla.setBounds(new Rectangle(235, 35, 73, 23));
66.    vainilla.setBackground(Color.yellow);
67.    vainilla.addItemListener(new ItemListener()
68.      {
69.        public void itemStateChanged(ItemEvent e)
70.        {
71.          vainilla_itemStateChanged(e);
72.        }
```

```
73.       });
74.     this.getContentPane().add(vainilla, null);  // se agregan los componentes
75.     this.getContentPane().add(chocolate, null);
76.     this.getContentPane().add(fresa, null);
77.   }
78.
79.   // se invoca este método en caso de que escojan fresa
80.   private void fresa_itemStateChanged(ItemEvent e)
81.   {
82.     String sabor = "No hay de Fresa";
83.     if ( e.getStateChange() == ItemEvent.SELECTED )
84.     {
85.       sabor = "Rico helado de Fresa";
86.     }
87.      System.out.println( e.getItem() + " => " + sabor );
88.   }
89.
90.// se invoca este método en caso de que escojan chocolate
91.   private void chocolate_itemStateChanged(ItemEvent e)
92.   {
93.    String sabor = "No hay de Chocolate";
94.     if ( e.getStateChange() == ItemEvent.SELECTED )
95.     {
96.       sabor = "Rico helado de Chocolate";
97.     }
98.      System.out.println( e.getItem() + " => " + sabor );
99.   }
100.
101.// se invoca este método en caso de que escojan vainilla
102.   private void vainilla_itemStateChanged(ItemEvent e)
103.   {
104.    String sabor = "No hay de Vainilla";
105.     if ( e.getStateChange() == ItemEvent.SELECTED )
106.     {
107.       sabor = "Rico helado de Vainilla";
108.     }
109.     System.out.println( e.getItem() + " => " + sabor );
110.   }
111.
112.  public static void main( String args[] ) throws IOException
113.{
114.     RicosHelados  rh = new  RicosHelados();
115.     rh.setVisible( true );
116.     System.out.println("da un teclazo para terminar!!!");
117.     System.in.read();
118.}
119.} // fin de la clase RicosHelados
```

La selección o la no selección de un **Checkbox** es enviada a la interface **ItemListener**. El **ItemEvent** es enviado conteniendo el método **getStatedChange()**, el cual regresa **ItemEvent.DESELECTED** o un **ItemEvent.SELECTED**, según sea el caso. El método **getItem()** regresa la etiqueta del objeto **Checkbox** como un **String** con la etiqueta que representa a ese **Checkbox**.

JcheckBox

La clase **JCheckBox** del **javax.swing** se conporta muy similar al **CheckBox** del **java.awt**, vea la figura A continuación tenemos la versión pero ahora con swing. Cheque el lector que aquí los eventos los controlamos con la clase **java.awt.event.ActionEvent** y no con la interface **ItemListener.**




```
1. package ejemploswing;
2.
3.
4. public class RicosHeladosCoyoacán extends javax.swing.JFrame {
5.
6.     /** Creates new form RicosHeladosCoyoacán */
7.     public RicosHeladosCoyoacán() {
8.         initComponents();
9.         setSize(370, 145);
10.         setLocation(100,200);
11.     }
```

```
12.
13.     /** This method is called from within the constructor to
14.      * initialize the form.
15.      * WARNING: Do NOT modify this code. The content of this method is
16.      * always regenerated by the Form Editor.
17.      */
18.     // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
19.     private void initComponents() {
20.         fresa = new javax.swing.JCheckBox();
21.         vainilla = new javax.swing.JCheckBox();
22.         chocolate = new javax.swing.JCheckBox();
23.
24.         getContentPane().setLayout(null);
25.
26.         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
27.         setTitle("Ricos helados de Coyoac\u00e1n");
28.         fresa.setBackground(java.awt.Color.pink);
29.         fresa.setText("FRESA");
30.            fresa.setBorder(javax.swing.BorderFactory.createEmptyBorder(0,  0,  0,
    0));
31.         fresa.setMargin(new java.awt.Insets(0, 0, 0, 0));
32.         fresa.setName("null");
33.         fresa.addActionListener(new java.awt.event.ActionListener() {
34.             public void actionPerformed(java.awt.event.ActionEvent evt) {
35.                 fresaActionPerformed(evt);
36.             }
37.         });
38.
39.         getContentPane().add(fresa);
40.         fresa.setBounds(20, 40, 100, 15);
41.         vainilla.setBackground(java.awt.Color.yellow);
42.         vainilla.setText("VAINILLA");
43.         vainilla.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0,
    0));
44.         vainilla.setMargin(new java.awt.Insets(0, 0, 0, 0));
45.         vainilla.addActionListener(new java.awt.event.ActionListener() {
46.             public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
47.                 vainillaActionPerformed(evt);
48.             }
49.         });
50.         getContentPane().add(vainilla);
51.        vainilla.setBounds(140, 40, 90, 15);
52.         chocolate.setBackground(new java.awt.Color(162, 126, 12));
53.         chocolate.setText("CHOCOLATE");
54.           chocolate.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0,
    0, 0));
55.         chocolate.setMargin(new java.awt.Insets(0, 0, 0, 0));
56.         chocolate.addActionListener(new java.awt.event.ActionListener() {
57.             public void actionPerformed(java.awt.event.ActionEvent evt) {
58.                 chocolateActionPerformed(evt);
59.             }
60.         });
61.
62.         getContentPane().add(chocolate);
63.         chocolate.setBounds(250, 40, 100, 15);
64.
65.         pack();
66.     }// </editor-fold>
67.
68.     private void chocolateActionPerformed(java.awt.event.ActionEvent evt) {
69.// TODO add your handling code here:
70.          String sabor = "No hay de Chocolate";
71.        if ( evt.getActionCommand().equals(chocolate.getText()) )
72.         {
73.            if( chocolate.isSelected() ) sabor = "Rico helado de Chocolate";
74.         }
75.         System.out.println( evt.getActionCommand() + " => " + sabor );
76.     }
77.
78.     private void vainillaActionPerformed(java.awt.event.ActionEvent evt) {
79.// TODO add your handling code here:
80.          String sabor = "No hay de Vainilla";
81.        if ( evt.getActionCommand().equals(vainilla.getText()) )
```

```
82.         {
83.            if( vainilla.isSelected() ) sabor = "Rico helado de Fresa";
84.         }
85.         System.out.println( evt.getActionCommand() + " => " + sabor );
86.     }
87.
88.     private void fresaActionPerformed(java.awt.event.ActionEvent evt) {
89. // TODO add your handling code here:
90.         String sabor = "No hay de Fresa";
91.         if ( evt.getActionCommand().equals(fresa.getText()) )
92.         {
93.            if( fresa.isSelected() ) sabor = "Rico helado de Fresa";
94.         }
95.         System.out.println( evt.getActionCommand() + " => " + sabor );
96.     }
97.         /**
98.      * @param args the command line arguments
99.      */
100.     public static void main(String args[]) {
101.         java.awt.EventQueue.invokeLater(new Runnable() {
102.             public void run() {
103.                 new RicosHeladosCoyoacán().setVisible(true);
104.             }
105.         });
106.     }
107.
108.     // Variables declaration - do not modify
109.     private javax.swing.JCheckBox chocolate;
110.     private javax.swing.JCheckBox fresa;
111.     private javax.swing.JCheckBox vainilla;
112.     // End of variables declaration
113.  }// fin de la clase RicosHeladosCoyoacán
```

CheckboxGroup – Botones de opción

Los **ChecboxGroup** proveen el agrupamiento de múltiples elementos **Checkbox** como un conjunto con exclusión mutua, de modo tal que un solo **Checkbox** en el conjunto tendrá en todo momento el valor de **true** .

El **Checkbox** con el valor **true** es el que actualmente este seleccionado. Se puede crear cada **Checkbox** de un grupo usando el constructor que toma un argumento adicional, un **CheckboxGroup**. Este objeto CheckboxGroup conecta el elemento Checkbox dentro del conjunto. La apariencia de cada elemento **Checkbox** agregado a el grupo es convertido a un botón de opción o "botón de radio".

El siguiente ejemplo tenemos un contenedor **JFrame** con cuatro opciones AM, FM, TV, VIDEO mutuamente excluyentes. Por default se establece como **true** a la opción TV. Cuando el usuario selecciona una de estas cuatro opciones se dispara el evento que escribe un mensaje por la consola, ver figura.


```
1. package mypackage1;
2. import javax.swing.JFrame;
3. import java.awt.Dimension;
4. import java.awt.CheckboxGroup;
5. import java.awt.Checkbox;
6. import java.awt.Rectangle;
7. import java.io.*;
8. import java.awt.event.ItemListener;
9. import java.awt.event.ItemEvent;
10./**
11.  * Programa de Java que te enseña a utilizar componentes
```



```
12.  * del paquete java.awt. Este demuestra el uso de los
13.  * objetos de la clase CheckboxGroup.
14.  * @autor Oscar A. González Bustamante
15.  * @version 1.0
16.  * Archivo: MiChecboxGroup.java
17.  */
18.public class MiCheckboxGroup extends JFrame
19.{
20.  private CheckboxGroup cbg = new CheckboxGroup();  // se crea el CheckboxGroup
21.  private Checkbox am = new Checkbox("AM",cbg, false); // constructores de los
   items
22.  private Checkbox fm = new Checkbox("FM",cbg, false);
23.  private Checkbox tv = new Checkbox("TV",cbg, true); // opción seleccionada
24.  private Checkbox video = new Checkbox("VIDEO",cbg, false);
25.  public MiCheckboxGroup()   {
26.    try
27.    {
28.      jbInit();
29.    }
30.    catch(Exception e)
31.    {
32.      e.printStackTrace();
33.    }
34.
35.  }
36.
37.  private void jbInit() throws Exception
38.  {
39.    this.getContentPane().setLayout(null);
40.    this.setSize(new Dimension(400, 300));
41.    this.setTitle("Botones de radio mutuamente excluyentes");
42.
43.    am.setBounds(new Rectangle(75, 85, 73, 23));
44.    am.addItemListener(new ItemListener() // se agrega un oyente al Checkbox am
45.      {
46.        public void itemStateChanged(ItemEvent e) // si cambia su estado
47.        {
48.          am_itemStateChanged(e);  // se dispara el evento por delegación
49.        }
50.      });
51.
52.    fm.setBounds(new Rectangle(120, 85, 75, 25));
53.    fm.addItemListener(new ItemListener()
54.      {
55.        public void itemStateChanged(ItemEvent e)
```

```
56.         {
57.           fm_itemStateChanged(e);
58.         }
59.       });
60.
61.     tv.setBounds(new Rectangle(160, 85, 73, 23));
62.     tv.addItemListener(new ItemListener()
63.       {
64.         public void itemStateChanged(ItemEvent e)
65.         {
66.           tv_itemStateChanged(e);
67.         }
68.       });
69.
70.     video.setBounds(new Rectangle(205, 85, 73, 23));
71.     video.addItemListener(new ItemListener()
72.       {
73.         public void itemStateChanged(ItemEvent e)
74.         {
75.           video_itemStateChanged(e);
76.         }
77.       });
78.     this.getContentPane().add(video, null);  // se agregan los componetes al
    JFrame
79.   this.getContentPane().add(tv, null);
80.    this.getContentPane().add(fm, null);
81.     this.getContentPane().add(am, null);
82.   }
83.
84.// se maneja el evento del Checkbox  am
85.   private void am_itemStateChanged(ItemEvent e)
86.   {
87.    String modo = "No hay AM";
88.     if ( e.getStateChange() == ItemEvent.SELECTED )
89.     {
90.       modo = "MODO AMPLITUD MODULADA";
91.     }
92.      System.out.println( e.getItem() + " => " + modo );
93.   }
94.
95.   private void fm_itemStateChanged(ItemEvent e)
96.   {
97.     String modo = "No hay FM";
98.     if ( e.getStateChange() == ItemEvent.SELECTED )
99.     {
100.       modo = "MODO FRECUENCIA MODULADA";
101.     }
102.      System.out.println( e.getItem() + " => " + modo );
103.   }
104.
105.   private void tv_itemStateChanged(ItemEvent e)
```

```
106.  {
107.  String modo = "No hay TV";
108.    if ( e.getStateChange() == ItemEvent.SELECTED )
109.    {
110.      modo = "MODO TELEVISIÓN";
111.    }
112.     System.out.println( e.getItem() + " => " + modo );
113.  }
114.
115.  private void video_itemStateChanged(ItemEvent e)
116.  {
117.    String modo = "No hay VIDEO";
118.    if ( e.getStateChange() == ItemEvent.SELECTED )
119.    {
120.      modo = "MODO VIDEO";
121.    }
122.     System.out.println( e.getItem() + " => " + modo );
123.  }
124.  public static void main( String args[] ) throws IOException
125.{
126.    MiCheckboxGroup  micbg = new  MiCheckboxGroup();
127.    micbg.setVisible( true );
128.    System.out.println("da un teclazo para terminar!!!");
129.    System.in.read();
130.    System.out.println("Fin del programa");
131.   } // fin del main()
132.} // fin de la clase MiCheckboxGroup
```

JRadioButton y ButtonGroup – Botones de opción.

El siguiente ejemplo es similar al anterior **JFrame** con cuatro opciones AM, FM, TV, VIDEO mutuamente excluyentes pero aquí utilizamos las clases **javax.swing.ButtonGroup** y **javax.swing.JRadioButton** . Por default se establece como **true** a la opción TV. Cuando el usuario selecciona una de estas cuatro opciones se dispara el evento que escribe un mensaje por la consola, ver figura.



```
1. package ejemploswing;
2. import java.awt.event.ItemEvent;
3.
4. public class MiJCheckBoxRadio extends javax.swing.JFrame {
5.
6.     /** Creates new form MiJCheckBoxRadio */
7.     public MiJCheckBoxRadio() {
8.         initComponents();
9.         setSize(400,300);// tamaño del JFrame
10.         setLocation(50,300);// ubicación en pantalla
11.     }
12.
13.
14.     // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
15.     private void initComponents() {
16.         cbg = new javax.swing.ButtonGroup(); // grupo de botones
17.         am = new javax.swing.JRadioButton(); // botones de radio
18.         fm = new javax.swing.JRadioButton();
19.         tv = new javax.swing.JRadioButton();
20.         video = new javax.swing.JRadioButton();
21.
22.         getContentPane().setLayout(null); // layout null
23.
24.         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
25.         setTitle("Botones de radio excluyen mutuamente");
26.         cbg.add(am); // agrega el boton de radio am al grupo de botones
27.         am.setText("AM");
28.         am.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0, 0));
29.         am.setMargin(new java.awt.Insets(0, 0, 0, 0));
30.         am.addItemListener(new java.awt.event.ItemListener() {
31.             public void itemStateChanged(java.awt.event.ItemEvent evt) {
32.                 amItemStateChanged(evt);
33.             }
34.         });
35.
```

```
36.         getContentPane().add(am);
37.         am.setBounds(30, 80, 60, 15);
38.
39.         cbg.add(fm);// agrega el boton de radio fm al grupo de botones
40.         fm.setText("FM");
41.         fm.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0, 0));
42.         fm.setMargin(new java.awt.Insets(0, 0, 0, 0));
43.         fm.addItemListener(new java.awt.event.ItemListener() {
44.             public void itemStateChanged(java.awt.event.ItemEvent evt) {
45.                 fmItemStateChanged(evt);
46.             }
47.         });
48.
49.         getContentPane().add(fm);
50.         fm.setBounds(120, 80, 70, 15);
51.
52.         cbg.add(tv);// agrega el boton de radio tv al grupo de botones
53.         tv.setText("TV");
54.         tv.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0, 0));
55.         tv.setMargin(new java.awt.Insets(0, 0, 0, 0));
56.         tv.addItemListener(new java.awt.event.ItemListener() {
57.             public void itemStateChanged(java.awt.event.ItemEvent evt) {
58.                 tvItemStateChanged(evt);
59.             }
60.         });
61.
62.         getContentPane().add(tv);
63.         tv.setBounds(220, 80, 70, 15);
64.
65.         cbg.add(video);// agrega el boton de radio video al grupo de botones
66.         video.setText("VIDEO");
67.         video.setBorder(javax.swing.BorderFactory.createEmptyBorder(0, 0, 0,
    0));
68.         video.setMargin(new java.awt.Insets(0, 0, 0, 0));
69.         video.addItemListener(new java.awt.event.ItemListener() {
70.             public void itemStateChanged(java.awt.event.ItemEvent evt) {
```

```
71.                 videoItemStateChanged(evt);
72.             }
73.         });
74.
75.         getContentPane().add(video);
76.         video.setBounds(320, 80, 120, 15);
77.
78.         pack();
79.     }// </editor-fold>
80.
81.     private void videoItemStateChanged(java.awt.event.ItemEvent evt) {
82. // TODO add your handling code here:
83.          String modo = "No hay VIDEO";
84.       if ( evt.getStateChange() == ItemEvent.SELECTED )
85.       {
86.         modo = "MODO VIDEO";
87.       }
88.       System.out.println( video.getText() + " => " + modo );
89.     }
90.
91.     private void tvItemStateChanged(java.awt.event.ItemEvent evt) {
92. // TODO add your handling code here:
93.          String modo = "No hay TV";
94.       if ( evt.getStateChange() == ItemEvent.SELECTED )
95.       {
96.         modo = "MODO TELEVISIÓN";
97.       }
98.       System.out.println( tv.getText() + " => " + modo );
99.     }
100.
101.     private void fmItemStateChanged(java.awt.event.ItemEvent evt) {
102. // TODO add your handling code here:
103.          String modo = "No hay FM";
104.       if ( evt.getStateChange() == ItemEvent.SELECTED )
105.       {
106.         modo = "MODO FRECUENCIA MODULADA";
```

```
107.        }
108.        System.out.println( fm.getText() + " => " + modo );
109.    }
110.
111.    private void amItemStateChanged(java.awt.event.ItemEvent evt) {
112.// TODO add your handling code here:
113.        String modo = "No hay AM";
114.        if ( evt.getStateChange() == ItemEvent.SELECTED )
115.        {
116.          modo = "MODO AMPLITUD MODULADA";
117.        }
118.        System.out.println( am.getText() + " => " + modo );
119.    }
120.
121.
122.    public static void main(String args[]) {
123.        java.awt.EventQueue.invokeLater(new Runnable() {
124.            public void run() {
125.                new MiJCheckBoxRadio().setVisible(true);
126.            }
127.        });
128.    }
129.
130.    // Variables declaration - do not modify
131.    private javax.swing.JRadioButton am;
132.    private javax.swing.ButtonGroup cbg;
133.    private javax.swing.JRadioButton fm;
134.    private javax.swing.JRadioButton tv;
135.    private javax.swing.JRadioButton video;
136.    // End of variables declaration
137.
138.} // fin de la clase  MiJCheckBoxRadio
```

Choice

Un **Choice** es un componente que nos provee de una simple lista de selección de un elemento el cual será seleccionada una entrada.

Cuando se hace clic en el objeto **Choice**, este despliega una lista de elementos que pueden ser añadidos a ella. Estos elementos agregados son objetos **String**.

Se hace uso de la interface **ItemListener** para observar los cambios en la lista **Choice**. Los detalles del manejo del evento son los mismos que para el caso del **Checkbox**.

Es siguiente código ejemplifica el uso de una lista **Choice** con elementos de algunas universidades en México (ver figura anterior) los cuales al seleccionarlos despliega texto por la consola y establece un valor nuevo para la etiqueta.

```
1. package mypackage1;
2. import javax.swing.JFrame;
3. import java.awt.Dimension;
4. import java.awt.Choice;
5. import java.awt.Rectangle;
6. import java.awt.Color;
7. import java.awt.event.ItemListener;
8. import java.awt.event.ItemEvent;
9. import java.io.*;
10.import java.awt.Label;
11.import javax.swing.JTree;
12./**
13.  * Programa de Java que te enseña a utilizar componentes
14.  * del paquete java.awt. Este demuestra el uso de los
15.  * objetos de la clase Choice.
16.  * @autor Oscar A. González Bustamante
17.  * @version 1.0
```


```
18.   * Archivo: ChoiceUniversidades.java
19.   */
20.public class ChoiceUniversidades extends JFrame
21.{
22.  private static String escuela = null;
23.  private Choice lista = new Choice(); // se crea la lista desplegable o Choice
24.  private Label le = new Label();
25.  public ChoiceUniversidades()   {
26.    try
27.    {
28.      jbInit();
29.    }
30.    catch(Exception e)
31.    {
32.      e.printStackTrace();
33.    }
34.  } // fin del constructor
35.    private void jbInit() throws Exception
36.  {
37.    this.getContentPane().setLayout(null);
38.    this.setSize(new Dimension(412, 134));
39.    this.setTitle("Universidades Mexicanas de prestigio y renombre");
40.    lista.setBounds(new Rectangle(90, 20, 175, 25));
41.    lista.setBackground(Color.blue); // color de la lista
42.    lista.setForeground(Color.yellow);  // color de la letra en la lista
43.
44.    lista.addItemListener(new ItemListener()
45.      {
46.        public void itemStateChanged(ItemEvent e)
47.        {
48.          lista_itemStateChanged(e);
49.        }
50.      });
51.    le.setText("hola");
52.    le.setBounds(new Rectangle(10, 65, 380, 30));
53.    le.setBackground(Color.white);
54.    lista.addItem("UNAM"); // agregamos los elementos a la lista
55.    lista.addItem("IPN");
56.    lista.addItem("UAM");
57.    lista.addItem("CHAPINGO");
58.    lista.addItem("UVM");
59.    lista.addItem("UNITEC");
60.    this.getContentPane().add(le, null);
61.    this.getContentPane().add(lista, null); // agregamos la lista al JFrame
62.
63.  }
64.
65.   public static void main( String args[] ) throws IOException
66.   {
67.    ChoiceUniversidades  cu = new  ChoiceUniversidades();
```



```
68.     cu.setVisible( true );
69.     System.out.println("da un teclazo para terminar!!!");
70.     System.in.read();
71.     System.out.println("Fin del programa");
72.    } // fin del main()
73.
74.// manejamos el evento de la lista Choice
75.  private void lista_itemStateChanged(ItemEvent e)
76.  {
77.       if( "UNAM".equals( e.getItem() ) )  // obtenemos el elemento seleccionado
78.       {
79.          escuela = new String("UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO");
80.       }
81.       else if ( "IPN".equals( e.getItem() ) )
82.       {
83.          escuela = new String("INSTITUTO POLITÉCNICO NACIONAL");
84.       }
85.       else if ( "UAM".equals( e.getItem() ) )
86.       {
87.          escuela = new String("UNIVERSIDAD AUTÓNOMA METROPOLITANA");
88.       }
89.       else if ( "CHAPINGO".equals( e.getItem() ) )
90.       {
91.          escuela = new String("UNIVERSIDAD DE CHAPINGO");
92.       }
93.       else if ( "UVM".equals( e.getItem() ) )
94.       {
95.          escuela = new String("UNIVERSIDAD DEL VALLE DE MÉXICO");
96.       }
97.       else if ( "UNITEC".equals( e.getItem() ) )
98.       {
99.          escuela = new String("UNIVERSIDAD TECNOLÓGICA DE MÉXICO");
100.      }
101.
102.      le.setText( escuela );  // establecemos nuevo valor a la etiqueta
103.      System.out.println("==> "+ escuela );
104.    } // fin del metodo que maneja el evento

105.} // fin de la clase ChoiceUniversidades
```

Canvas

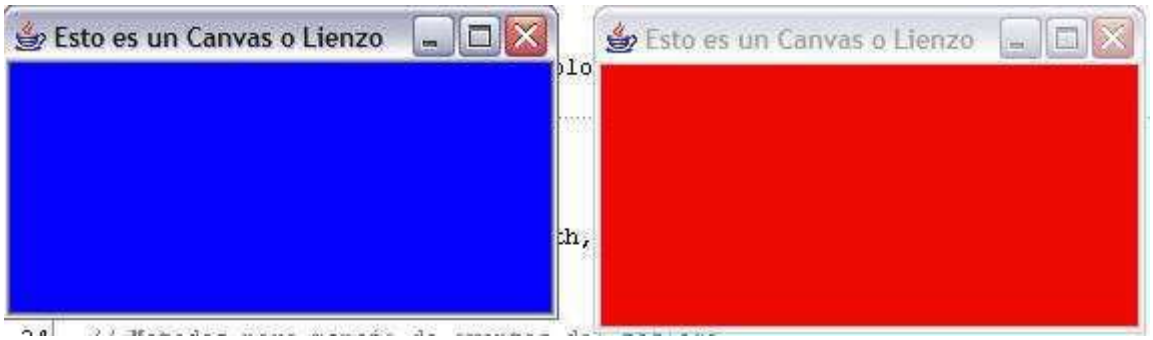
Un **Canvas** (lienzo) provee de un espacio en blanco o coloreado por cualquier color de fondo. Tiene un tamaño preferido de 0 por 0, a menos que se especifique un tamaño usando **setSize()**. Para especificar el tamaño, hay que colocarlo en un administrador de diseño que especifique el tamaño.

Se puede usar ese espacio para dibujar, escribir texto, o recibir del teclado o del ratón alguna entrada. Para dibujar utilizamos ampliamente los métodos de **java.awt.Graphics**. Generalmente un **Canvas** además de proveer de un espacio general para el dibujo nos proporciona un área de trabajo para personalizar un componente.

El **Canvas** puede "oir" todos los eventos que son aplicables a todo componente en general. En particular, podría desearse agregar objetos como un **KeyListener**, **MouseMotionListener**, o un **MouseListener** al mismo, para permitir la respuesta a algún tipo de entrada por parte del usuario.

Nota: Para recibir eventos de teclado en un Canvas, es necesario llamar al método requestFocus() de el Canvas. Si esto no es hecho, es generalmente imposible redireccionar las teclas pulsadas al Canvas, En lugar de que las teclas pulsadas se pierdan enteramente en otro componente.

El siguiente ejemplo cambia el color del Canvas cada ves que una tecla es presionada.




```
1. package mypackage1;
2. import java.io.*;
3. import java.awt.event.*;
4. import java.awt.*;
5. /**
6.    * Programa de Java que te enseña a utilizar componentes
7.    * del paquete java.awt. Este demuestra el uso de los
8.    * objetos de la clase Canvas. Con eventos del teclado
9.    * @autor Oscar A. González Bustamante
10.   * @version 1.0
11.   * Archivo: CanvasLienzo.java
12.   */
13.public class CanvasLienzo extends Canvas implements KeyListener
14.{
15.  private int index;
16.  Color  colores[] = { Color.red, Color.green, Color.blue };
17.
18.  public  void paint( Graphics g )
19.  {
20.    g.setColor( colores[ index ] );
```


```
21.    g.fillRect( 0, 0, getSize().width, getSize().height );
22.  }
23.
24.  // Metodos para manejo de eventos del teclado
25.  public void keyTyped( KeyEvent ke )
26.  {
27.    index++;
28.    if ( index == colores.length )
29.    {
30.        index = 0;
31.    }
32.    repaint();
33.  }
34.  // Metodos de teclado no utilizados en este programa
35.  public void keyPressed( KeyEvent ke )  { }
36.  public void keyReleased( KeyEvent ke ) { }
37.
38.  public static void main( String args[] ) throws IOException
39.   {
40.    Frame  f = new  Frame("Esto es un Canvas o Lienzo");
41.    CanvasLienzo cl = new CanvasLienzo();
42.    cl.setSize(150, 150 );
43.    f.setLayout( new BorderLayout() );
44.    f.add( cl, BorderLayout.CENTER );
45.    cl.requestFocus(); //solicitud del foco o enfoque
46.    cl.addKeyListener( cl );
47.    f.pack();
48.    f.setVisible( true );
49.  } // fin del main()
50.} // fin de la clase CanvasLienzo
```

TextField

Un objeto **TextField** es una línea simple de texto en un cuadro de entrada, por ejemplo:

debido a que solo puede tener una línea, un **ActionListener** puede ser informado, usando **actionPerformed()**, cuando la tecla del Enter o Return sea presionada. Puede agregar otros componentes oyentes si así lo desea.

Se puede utilizar el componente **TextField** para enmascarar teclas como en el caso de una contraseña o para ignorar algunas teclas del teclado. El siguiente ejemplo crea el código para que el campo de texto *Nombre* ignore los dígitos y que la contraseña aparezca enmascarada con un asterisco.

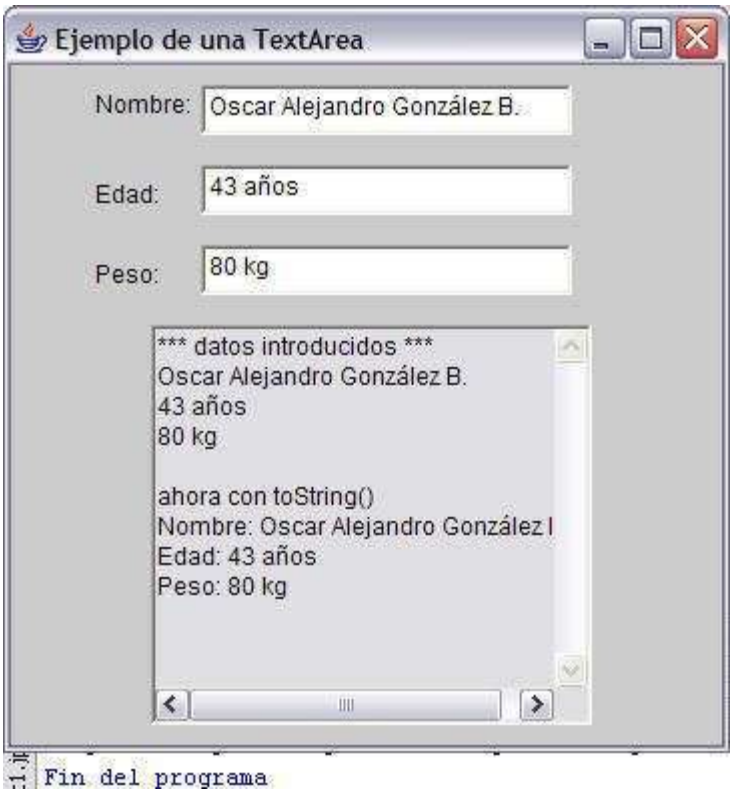
```
1. package mypackage1;
2. import javax.swing.JFrame;
3. import java.awt.Dimension;
4. import java.awt.TextField;
5. import java.awt.Rectangle;
6. import java.awt.Label;
7. import java.awt.event.*;
8. /**
9.   * Programa de Java que te enseña a utilizar componentes
10.  * del paquete java.awt. Este demuestra el uso de los
11.  * objetos de la clase TextField.
12.  * @autor Oscar A. González Bustamante
13.  * @version 1.0
14.  * Archivo: ComponenteTextField.java
15.  */
16.public class ComponenteTextField extends JFrame
17.{
18.   // se construyen los campos de texto
19.  private TextField tfnombre = new TextField("Nombre del Web");
20.  private TextField tfurl = new TextField( );
21.  private TextField tfpassw = new TextField( );
22.  // se construyen sus etiquetas
23.  private Label labnombre = new Label("Nombre:");
```

```
24.  private Label laburl = new Label();
25.  private Label labpassw = new Label();
26.
27.  public ComponenteTextField()
28.  {
29.    try
30.    {
31.      jbInit();
32.    }
33.    catch(Exception e)
34.    {
35.      e.printStackTrace();
36.    }
37.
38.  } // fin del constructor
39.
40.  private void jbInit() throws Exception
41.  {
42.    this.getContentPane().setLayout(null); // layout por default
43.    this.setSize(new Dimension(388, 220)); // dimensión del JFrame
44.    this.setTitle("Ejemplo del componente TextField"); // titulo del JFrame
45.    tfnombre.setBounds(new Rectangle(160, 20, 180, 25)); // tamaño del
   TextField
46.    tfnombre.addKeyListener ( new  QuitaDigitos() ); // quita los digitos al
   campo tfnombre
47.    tfurl.setText("URL del Web"); // su texto por default
48.    tfurl.setBounds(new Rectangle(160, 65, 150, 25));
49.    tfpassw.setBounds(new Rectangle(160, 115, 145, 30));
50.    tfpassw.setEchoChar('*'); // caracter de enmascaramiento
51.    labnombre.setBounds(new Rectangle(25, 20, 130, 30));
52.    laburl.setText("Dirección o URL:");
53.   laburl.setBounds(new Rectangle(25, 65, 100, 30));
```


```
54.     labpassw.setText("Contraseña:");
55.     labpassw.setBounds(new Rectangle(25, 115, 100, 30));
56.     // se agregan los componentes
57.     this.getContentPane().add(labpassw, null);
58.     this.getContentPane().add(laburl, null);
59.     this.getContentPane().add(labnombre, null);
60.     this.getContentPane().add(tfpassw, null);
61.     this.getContentPane().add(tfurl, null);
62.     this.getContentPane().add(tfnombre, null);
63.   }
64.
65.  class QuitaDigitos  extends  KeyAdapter
66.   {
67.    public void keyPressed( KeyEvent e )
68.     {
69.       char c = e.getKeyChar();
70.       if ( Character.isDigit( c ) )
71.       {
72.            e.consume();
73.       }
74.     }
75.   } // fin de la clase interna  QuitaDigitos
76.
77.    public static void main( String args[] )
78.   {
79.     ComponenteTextField  ctf = new  ComponenteTextField();
80.     ctf.setVisible( true );
81.     System.out.println("Fin del programa");
82.   } // fin del main()
83. } // fin de la clase ComponenteTextField
```

TextArea

Una **TextArea** "área de texto" es un objeto para manejar texto capturado desde el un dispositivo de entrada y este puede manejar múltiples líneas y múltiples columnas. Se puede establecer ue sea de solo lectura, usando el método **setEditable(boolean)**. Este desplegará unas barras de deslizamiento verticales y horizontales.



El oyente se puede especificar con **addTextListener()** el cual recibe la notificación de las teclas pulsadas de la misma manera que lo hace un **TextField**.

Se puede en agregar oyentes en general, a una área de texto, pero debido a que es multilínea, el presionar la tecla de Enter pone otro caracter en el buffer. Si se necesita reconocer "una entrada completa", se puede poner un botón de aplicar o Enviar seguido del área de texto para permitir al usuario indicar esto.

```
1. package mypackage1;
2. import javax.swing.JFrame;
3. import java.awt.Dimension;
4. import java.awt.Label;
5. import java.awt.Rectangle;
6. import java.awt.TextArea;
7. import java.awt.TextField;
8. import java.awt.event.ActionListener;
9. import java.awt.event.ActionEvent;
10.import java.lang.String;
11./**
```


```
12.  * Programa de Java que te enseña a utilizar componentes
13.  * del paquete java.awt. Este demuestra el uso de los
14.  * objetos de la clase TextArea. Utiliza una clase llamada
15.  * Persona que esta en el mismo paquete
16.  * @autor Oscar A. González Bustamante
17.  * @version 1.0
18.  * Archivo: TextAreaPersona.java
19.  */
20.public class TextAreaPersona extends JFrame
21.{
22.  private Label labNombre = new Label();
23.  private Label labEdad = new Label();
24.  private Label labPeso = new Label();
25.  private TextField tfNombre = new TextField();
26.  private TextField tfEdad = new TextField();
27.  private TextField tfPeso = new TextField();
28.  private TextArea ta = new TextArea();  // se construye un TextArea
29.  private String  texto = null;
30.
31.  public TextAreaPersona()
32.  {
33.    try
34.    {
35.      jbInit();
36.    }
37.    catch(Exception e)
38.    {
39.      e.printStackTrace();
40.    }
41.
42.  }
43.
44.  private void jbInit() throws Exception
45.  {
46.    this.getContentPane().setLayout(null);
47.    this.setSize(new Dimension(363, 374));
48.    this.setTitle("Ejemplo de una TextArea");
49.    // etiquetas
50.    labNombre.setBounds(new Rectangle(40, 5, 50, 30));
51.    labNombre.setText("Nombre:");
52.    labEdad.setText("Edad:");
53.    labEdad.setBounds(new Rectangle(40, 50, 60, 30));
54.    labPeso.setText("Peso:");
55.    labPeso.setBounds(new Rectangle(40, 90, 110, 30));
56.
57.    tfNombre.setBounds(new Rectangle(95, 10, 185, 25));
58.    tfEdad.setBounds(new Rectangle(95, 50, 185, 25));
59.    tfPeso.setBounds(new Rectangle(95, 90, 185, 25));
60.    tfPeso.addActionListener(new ActionListener()
61.      {
```


```
62.         public void actionPerformed(ActionEvent e)
63.         {
64.           tfPeso_actionPerformed(e);
65.         }
66.       });
67.     texto = new String( "*** datos introducidos ***\n");
68.     ta.setText( texto ); // establece texto inicial al TextArea
69.     ta.setBounds(new Rectangle(70, 130, 220, 200));  // dimensión del TextArea.
70.     ta.setEditable(false); // se inhibe la entrada por teclado al TextArea
71.     this.getContentPane().add(ta, null);
72.     this.getContentPane().add(tfPeso, null);
73.     this.getContentPane().add(tfEdad, null);
74.     this.getContentPane().add(tfNombre, null);
75.     this.getContentPane().add(labPeso, null);
76.     this.getContentPane().add(labEdad, null);
77.     this.getContentPane().add(labNombre, null);
78.   }
79.
80.   private void tfPeso_actionPerformed(ActionEvent e)
81.   {  // construyo una persona con los valores de los TextField
82.      Persona pe = new Persona( tfNombre.getText(),
83.                                tfEdad.getText(),
84.                                tfPeso.getText() );
85.      texto += tfNombre.getText() + "\n";
86.      texto += tfEdad.getText() + "\n";
87.      texto += tfPeso.getText() + "\n\n";
88.      texto += "ahora con toString() \n";
89.      texto += pe.toString();
90.      // se agregan estos valores al TextArea
91.      ta.setText( texto );
92.
93.
94.   }
95.
96.    public static void main( String args[] )
97.    {
98.     TextAreaPersona  tap = new  TextAreaPersona();
99.     tap.setVisible( true );
100.     System.out.println("Fin del programa");
101.    } // fin del main()


102.} // fin de la clase TextAreaPersona
```


```
1. package mypackage1;
2. /**
3.    * Programa de Java que utiliza TextAreaPersona.
4.    * @autor Oscar A. González Bustamante
5.    * @version 1.0
6.    * Archivo: Persona.java
7.    */
8. class Persona {
9.     String   nombre;
10.    String      edad;
11.    String   peso;
12.
13.     Persona( String nom, String edad, String peso) {
14.       nombre = nom;
15.       this.edad = edad;
16.       this.peso = peso;
17.     } // fin del constructor
18.
19.      Persona () { }
20.
21.      public String  toString() {
22.           String s = new String ( "Nombre: " + nombre + "\n" +
23.                                   "Edad: " + edad + "\n" +
24.                                   "Peso: " + peso + "\n" );
25.              return s;
26.       }
27.
28.} // fin de la clase Persona
```

List

Una **List** es una "lista" que presenta varias opciones de texto que son desplegadas en una área que permite seleccionar varios elementos al mismo tiempo y es posible recorrerla mediante barras de desplazamiento.

El numero de argumentos del constructor define el tamaño preferido de la lista en términos del número de renglones visibles. Como ha de suponerse un administrador de diseño "Layout" puede sobreescribir este valor. El argumento booleano indica que la lista debe permitir al usuario hacer múltiples selecciones.

Cuando un elemento es seleccionado o deseleccionado de la lista, **java.awt** envía una instancia de **ItemEvent** a la lista. Cuando el usuario hace doble clic sobre el elemento en una lista desplazable, un **ActionEvent** es generado por la lista en modo de selección simple y en modo de selección múltiple. Los elementos son seleccionados desde la lista de acuerdo a las convenciones de la plataforma. Para un entorno UNIX Motif, un simple clic ilumina una entrada de la lista, pero debe hacerse doble clic para disparar la acción de la lista.

El siguiente ejemplo muestra una lista con platillos mexicanos que al seleccionarlos despliega por la consola lo que el comensal quiere comer:

```
1. package mypackage1;
2. import javax.swing.JFrame;
3. import java.awt.Dimension;
4. import java.awt.List;
5. import java.awt.Rectangle;
6. import java.awt.Color;
7. import java.awt.Label;
8. import java.awt.event.ActionListener;
9. import java.awt.event.ActionEvent;
10./**
11.  * Programa de Java que te enseña a utilizar componentes
12.  * del paquete java.awt. Este demuestra el uso de los
13.  * objetos de la clase List.
14.  * @autor Oscar A. González Bustamante
15.  * @version 1.0
16.  * Archivo: ListEj.java
17.  */
```


```
18.public class ListEj extends JFrame
19.{
20.  private List lis = new List(); // se instancia la lista
21.  private Label lab = new Label();
22.  private String comida = null;
23.
24.  public ListEj()
25.  {
26.    try
27.    {
28.      jbInit();
29.    }
30.    catch(Exception e)
31.    {
32.      e.printStackTrace();
33.    }
34.
35.  }
36.
37.  private void jbInit() throws Exception
38.  {
39.    this.getContentPane().setLayout(null);
40.    this.setSize(new Dimension(226, 169));
41.    this.setTitle("Ejemplo de una List");
42.    lis.setBounds(new Rectangle(60, 25, 100, 85)); //tamaño de la lista
43.    lis.setBackground(Color.orange); // color de fondo de la lista
44.    lis.setForeground(Color.blue); // color de la letra de la lista
45.    lis.setMultipleMode( false ); // modo de multiples opciones a escoger
46.    lis.addActionListener(new ActionListener()
47.      {
48.        public void actionPerformed(ActionEvent e)
49.        {
50.          lis_actionPerformed(e);
51.        }
52.      });
53.    lab.setText("Rica comida mexicana ¡ummm rico!");
54.    lab.setBounds(new Rectangle(35, 5, 175, 25));
55.    lis.add("Tacos"); // se agregan los elementos a la lista
56.    lis.add("Chiles Enogada");
57.    lis.add("Tortas");
58.    lis.add("Enchiladas");
59.    lis.add("Sopes");
60.    lis.add("Pambazos");
61.    lis.add("Tamales");
62.    lis.add("Burritos");
63.    this.getContentPane().add(lab, null);
64.    this.getContentPane().add(lis, null);
65.  }
66.
67.   public static void main( String args[] )
```

```
68.    {
69.     ListEj  le = new  ListEj();
70.     le.setVisible( true );
71.     System.out.println("Fin del programa");
72.    } // fin del main()
73.
74. // se maneja el evento de la lista
75.  private void lis_actionPerformed(ActionEvent e)
76.  {
77.
78.      System.out.println("Yo quiero comer: " +  e.getActionCommand()  );
79.
80.  }

81.} //fin de la clase ListEj
```

ScrollPane

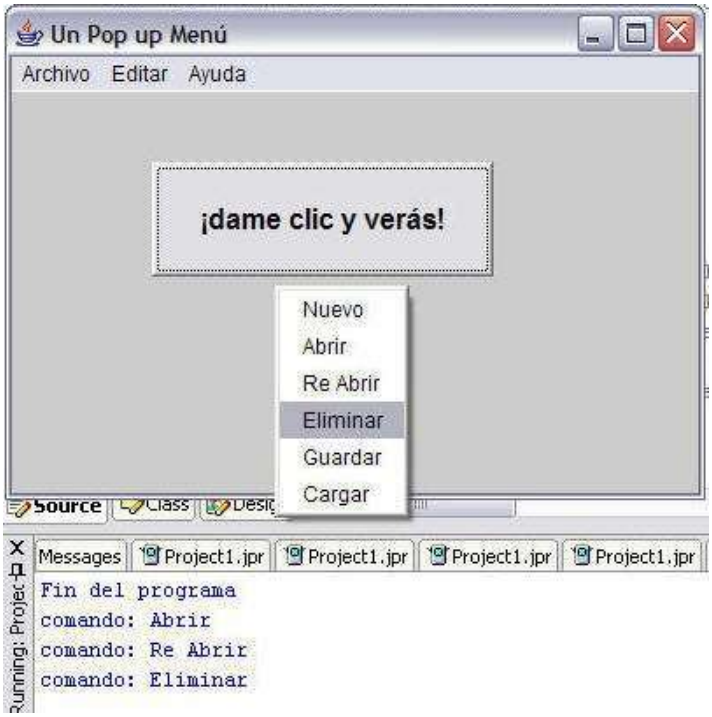
Un **ScroollPane** provee de un contenedor general que no puede ser usado como un componente libre. Siempre esta asociado con un contenedor (por ejemplo, un **Frame**). Provee un área de visualización dentro de una región grande y con barras de deslizamiento para manipularla.

El **ScroollPane** crea y maneja las barras de desplazamiento y las contiene en un solo componente. No es posible controlar el administrador de diseño que usa, en lugar de esto se puede agregar a un Panel para que el scrooll pane, configure el administrador de diseño, y coloque los componentes en dicho panel.

Generalmente, no es posible manejar eventos en un **ScroollPane**; los eventos son manejados a través de los componentes que se contengan.

MenuBar

Un **Menu** es un componente diferente a otros componentes porque no se puede agregar a un Menu a los contenedores comunes. Solo se puede agregar menús a un "menu container". Se puede comenzar un árbol de menú poniendo un **MenuBar** "una barra de menú" en un **Frame**, usando el método **setMenuBar()**. Desde este punto, se pueden agregar menús a la barra de menú y menús o elementos de menú dentro del menú.



Los menús Pop-up son una excepción porque estos aparecen como ventanas flotantes y no requieren un administrador de diseño, pero es importante agregar el menu Pop-up al contenedor padre, de lo contrario no funciona.

Los Help menu (menús de Ayuda) pueden ser implementados en un **MenuBar** mediante el uso del método **setHelpMenu(Menu)**. Se debe agregar el menú a ser tratado como un Help menu a la barra de menu; entonces será tratado en la misma forma que un Help menu para la plataforma en que se este trabajando. Para X/Motif-type systems, este será colocado como una entrada de menú al final derecho de la barra de menú.

El siguiente programa maneja estos diversos tipos de menus y al ejecutarlo despliega un **JFrame** con un N, y un Help menu. Al darle clic al botón desplegará un menú **popupMenu.**


```
1. package mypackage1;
2. import javax.swing.JFrame;
3. import java.awt.Dimension;
4. import java.awt.Button;
5. import java.awt.Rectangle;
6. import java.awt.event.ActionListener;
7. import java.awt.event.ActionEvent;
8. import java.awt.PopupMenu;
9. import java.awt.MenuItem;
10.import java.awt.Font;
11.import java.awt.Color;
12.import java.awt.MenuBar;
13.import java.awt.Menu;
14./**
15.  * Programa de Java que te enseña a utilizar componentes
16.  * del paquete java.awt. Este demuestra el uso de los
17.  * objetos de la clase PopupMenu.
18.  * @autor Oscar A. González Bustamante
19.  * @version 1.0
20.  * Archivo: PopUpMenu.java
21.  */
22.
23.public class PopUpMenu extends JFrame {
24. String [] elementos = {"Nuevo", "Abrir","Re Abrir","Eliminar", "Guardar",
   "Cargar", "Salir" };
25.  private Button b = new Button();
26.  private PopupMenu popupMenu1 = new PopupMenu();  // se instancia un PopupMenu
27.  private MenuItem menuItem1 = new MenuItem(); // se instancian los elementos
28.  private MenuItem menuItem2 = new MenuItem(); // del PopupMenu
29.  private MenuItem menuItem3 = new MenuItem();
30.  private MenuItem menuItem4 = new MenuItem();
31.  private MenuItem menuItem5 = new MenuItem();
32.  private MenuItem menuItem6 = new MenuItem();
33.  private MenuBar menuBar1 = new MenuBar();
34.  private Menu  a = new Menu ( "Archivo");
35.  private Menu  e = new Menu ( "Editar");
36.  private Menu  h = new Menu ( "Ayuda" );
37.
38.  public PopUpMenu()
39.  {
40.    try
41.    {
42.      jbInit();
43.    }
44.    catch(Exception e)
45.    {
46.      e.printStackTrace();
```

```
47.     }
48.
49.   }
50.
51.   private void jbInit() throws Exception
52.   {
53.     this.getContentPane().setLayout(null);
54.     this.setSize(new Dimension(400, 282));
55.     this.setTitle("Un Pop up Menú");
56.     this.setBackground(Color.blue);
57.     b.setLabel("¡dame clic y verás!");
58.     b.setBounds(new Rectangle(80, 60, 195, 65));
59.     b.setActionCommand("¡dame clic y verás! ");
60.     b.setFont(new Font("Tahoma", 1, 16));
61.     b.addActionListener(new ActionListener() // oyente al botón
62.       {
63.         public void actionPerformed(ActionEvent e)
64.         {
65.           b_actionPerformed(e);
66.         }
67.       });
68.
69.     popupMenu1.setLabel("Un popup"); // se establecen las etiquetas
70.     menuItem1.setLabel( elementos[0] );
71.     menuItem2.setLabel( elementos[1] );
72.     menuItem3.setLabel( elementos[2] );
73.     menuItem4.setLabel( elementos[3] );
74.     menuItem5.setLabel( elementos[4] );
75.     menuItem6.setLabel( elementos[5] );
76.     popupMenu1.addActionListener( new ActionListener() // oyente al botón para
    el poppupMenu
77.       {
78.         public void actionPerformed(ActionEvent e)
79.         {
80.           popupMenu1_actionPerformed(e);
81.         }
82.       });
83.     this.getContentPane().add(b, null); // se agrega el botón al contenedor
84.     popupMenu1.add(menuItem1);
85.     popupMenu1.add(menuItem2);
86.     popupMenu1.add(menuItem3);
87.     popupMenu1.add(menuItem4);
88.     popupMenu1.add(menuItem5);
89.     popupMenu1.add(menuItem6);
90.
91.     menuBar1.add( a ); // se agrega el menu Archivo al MenuBar
92.     menuBar1.add( e ); // sea agrega el menu Editar al MenuBar
93.     menuBar1.setHelpMenu( h );  // agrega un menu de ayuda al MenuBar
94.     // agregar el PopupMenu al Contendor padre JFrame si no no funciona
95.     // al agregarlo tambien se agregan todos sus items.
```

```
96.     this.getContentPane().add(popupMenu1);
97.
98.      }
99.
100.   public static void main( String args[] )
101.    {
102.     PopUpMenu  pum = new  PopUpMenu();
103.     pum.setVisible( true ); // hace visible al JFrame
104.     pum.setMenuBar( pum.menuBar1 );
105.
106.     System.out.println("Fin del programa");
107.    } // fin del main()
108.
109.  private void b_actionPerformed(ActionEvent e)
110.  {
111.      popupMenu1.show( b , 70, 70);  // muestra  PopupMenu
112.   }
113.
114.     private void popupMenu1_actionPerformed(ActionEvent e)
115.  {
116.    String item = null;
117.     int i;
118.     // maneja el evento de ver cual MenuItem fue seleccionado.
119.      for ( i=0; i < elementos.length; ++i )
120.       if ( e.getActionCommand().equals( elementos[i] ) )
121.            item = new  String ( elementos[i] );
122.
123.       System.out.println("comando: " + item );
124.
125.     }


126.} // fin de la clase PopUpMenu
```

Creando un MenuBar

Un componente **MenuBar** es un menú horizontal. Puede solamente agregarse a un objeto **Frame**, y forma la raíz de todos los árboles menú. Un **Frame** despliega un **MenuBar** a la vez. Sin embargo, se pueden cambiar el **MenuBar** en base al estado de el programa así que diferentes menús pueden aparecer en distintos puntos; por ejemplo:

El **MenuBar** no soporta oyentes. Como parte normal del comportamiento de un menú, los eventos anticipados que ocurran en la región de la barra de menú son procesados automáticamente.

```
1. package mypackage1;
2. import javax.swing.JFrame;
3. import java.awt.Dimension;
4. import java.awt.MenuBar;
5. import java.awt.Menu;
6. /**
7.   * Programa de Java que te enseña a utilizar componentes
8.   * del paquete java.awt. Este demuestra el uso de los
9.   * objetos de la clase MenuBar.
10.  * @autor Oscar A. González Bustamante
11.  * @version 1.0
12.  * Archivo: CreandoMenuBar.java
13.  */
14.public class CreandoMenuBar extends JFrame
15.{
16.  private MenuBar mb1 = new MenuBar(); // se crea un objeto MenuBar
17.  private Menu menu1 = new Menu(); // se crea un objeto Menu
18.
19.
20.  public CreandoMenuBar()
21.  {
22.    try
23.    {
24.      jbInit();
25.    }
26.    catch(Exception e)
27.    {
28.      e.printStackTrace();
29.    }
30.
31.  }
32.
33.  private void jbInit() throws Exception
34.  {
35.    this.getContentPane().setLayout(null);
```

```
36.     this.setSize(new Dimension(400, 300));
37.     this.setTitle("Creando Un Menu Bar");
38.     mb1.setName( "Soy un MenuBar");
39.    // se hace esto para que aparezca la barra de menu vacia
40.     menu1.setLabel(" ");
41.     mb1.add(menu1);
42.
43.   }
44.  public static void main( String args[] )
45.   {
46.     CreandoMenuBar cmb = new CreandoMenuBar();
47.     cmb.setMenuBar( cmb.mb1 );   // se agrega el MenuBar al JFrame
48.     cmb.setVisible ( true );
49.
50.   } // fin del main()


51.} // fin de la clase CreandoMenuBar
```

Es posible crear entradas a una barra de menú agregando objetos de la clase **Menu**. Los componentes **Menu** proveen a la barra de menú elementos de tipo menú pull-down " menús desplegables ".

```
private MenuBar mb1 = new MenuBar();
private Menu  a = new Menu ( "Archivo");
private Menu  e = new Menu ( "Editar");
private Menu  h = new Menu ( "Ayuda" );
mb1.add( a ); // se agrega el menu Archivo al MenuBar
mb1.add( e ); // sea agrega el menu Editar al MenuBar
mb1.setHelpMenu( h );  // agrega un menu de ayuda al MenuBar
```

Puede agregarse un **ActionListener** a un objeto **Menu**, pero este es poco común. Normalmente, se usan menús para desplegar y controlar los menús items, " elementos de menú " los cuales son descritos en la siguiente sección.

Creando un MenuItem

Un componente **MenuItem** son los nodos hoja de una árbol de menú. Son agregados a un menú para completarlo; por ejemplo:


```
1. package mypackage1;
2. import javax.swing.JFrame;
3. import java.awt.Dimension;
4. import java.awt.MenuBar;
5. import java.awt.Menu;
6. import java.awt.MenuItem;
7. /**
8.   * Programa de Java que te enseña a utilizar componentes
9.   * del paquete java.awt. Este demuestra el uso de los
10.  * objetos de la clase MenuItem.
11.  * @autor Oscar A. González Bustamante
12.  * @version 1.0
13.  * Archivo: CreandoMenuItem.java
14.  */
15.public class CreandoMenuItem extends JFrame
16.{
17.  private MenuBar mb1 = new MenuBar(); // se crea un objeto MenuBar
18.  private Menu a = new Menu("Archivo"); // se crean  objetos Menu
19.  private Menu e = new Menu("Editar");
20.  private Menu h = new Menu("Ayuda");
21.  MenuItem mi1 = new MenuItem("Nuevo"); // se crean objetos MenuItem
22.  MenuItem mi2 = new MenuItem("Guardar");
23.  MenuItem mi3 = new MenuItem("Cargar");
24.  MenuItem mi4 = new MenuItem("Salir");
25.
26.  public CreandoMenuItem()
27.  {
28.    try
29.    {
30.      jbInit();
31.    }
32.    catch(Exception e)
```

```
33.     {
34.       e.printStackTrace();
35.     }
36.
37.   }
38.
39.   private void jbInit() throws Exception
40.   {
41.     this.getContentPane().setLayout(null);
42.     this.setSize(new Dimension(400, 300));
43.     this.setTitle("Creando Un Menu Bar");
44.     mb1.setName( "Soy un MenuBar");
45.    // se hace esto para que aparezca la barra de menu vacia
46.    mb1.add( a ); // se agregan los objetos Menu al MenuBar
47.    mb1.add( e );
48.    mb1.setHelpMenu( h ); // se agrega el objeto menu Help a MenuBar
49.    a.add( mi1 ); // agregando los objetos MenuItem al objeto Menu
50.    a.add( mi2 );
51.    a.add( mi3 );
52.    a.addSeparator(); // agrega un separador entre los MenuItem
53.    a.add( mi4 );
54.}
55.  public static void main( String args[] )
56.   {
57.     CreandoMenuItem cmi = new CreandoMenuItem();
58.     cmi.setMenuBar( cmi.mb1 );  // se agrega el MenuBar al JFrame
59.     cmi.setVisible ( true );
60.
61.   } // fin del main()


62.} // fin de la clase CreandoMenuItem
```

Usualmente, puede agregarse un **ActionListener** a un objeto **MenuItem** para proveerle comportamiento a los menús.

Creando un CheckboxMenuItem

Un **CheckboxMenuItem** es un elemento de menú que puede ser verificado, así que puede tener selecciones de tipo (encendido o apagado) en sus menús; por ejemplo:


```
1. package mypackage1;
2. import javax.swing.JFrame;
3. import java.awt.Dimension;
4. import java.awt.MenuBar;
5. import java.awt.Menu;
6. import java.awt.MenuItem;
7. import java.awt.CheckboxMenuItem;
8. /**
9.   * Programa de Java que te enseña a utilizar componentes
10.  * del paquete java.awt. Este demuestra el uso de los
11.  * objetos de la clase ChecboxMenuItem.
12.   * @autor Oscar A. González Bustamante
13.   * @version 1.0
14.  * Archivo: CreandoChecboxMenuItem.java
15.   */
16.public class CreandoCheckboxMenuItem extends JFrame
17.{
18.  private MenuBar mb1 = new MenuBar(); // se crea un objeto MenuBar
19.  private Menu a = new Menu("Archivo"); // se crean  objetos Menu
20.  private Menu e = new Menu("Editar");
21.  private Menu h = new Menu("Ayuda");
22.  MenuItem mi1 = new MenuItem("Nuevo"); // se crean objetos MenuItem
23.  MenuItem mi2 = new MenuItem("Guardar");
24.  MenuItem mi3 = new MenuItem("Cargar");
25.  MenuItem mi4 = new MenuItem("Salir");
26.  // creando el objeto CheckboxMenuItem
27.  CheckboxMenuItem mi5 = new CheckboxMenuItem("Reiniciar");
28.
29.  public CreandoCheckboxMenuItem()
30.  {
31.    try
32.    {
33.      jbInit();
```

```
34.     }
35.     catch(Exception e)
36.     {
37.       e.printStackTrace();
38.     }
39.
40.   }
41.
42.   private void jbInit() throws Exception
43.   {
44.     this.getContentPane().setLayout(null);
45.     this.setSize(new Dimension(400, 300));
46.     this.setTitle("Creando Un Menu Bar");
47.     mb1.setName( "Soy un MenuBar");
48.     mb1.add( a ); // se agregan los objetos Menu al MenuBar
49.    mb1.add( e );
50.    mb1.setHelpMenu( h ); // se agrega el objeto menu Help a MenuBar
51.    a.add( mi1 ); // agregando los objetos MenuItem al objeto Menu
52.    a.add( mi2 );
53.    a.add( mi3 );
54.    a.addSeparator(); // agrega un separador entre los MenuItem
55.    a.add( mi4 );
56.     a.add( mi5 ); // se agrega el CheckboxMenuItem al menu Archivo
57.}
58.  public static void main( String args[] )
59.   {
60.     CreandoCheckboxMenuItem ccbmi = new CreandoCheckboxMenuItem();
61.     ccbmi.setMenuBar( ccbmi.mb1 );  // se agrega el MenuBar al JFrame
62.     ccbmi.setVisible ( true );
63.
64.   } // fin del main()
65.} // fin de la clase CreandoCheckboxMenuItem
```

Se debería monitorear el **CheckboxMenuItem** utilizando la interface **ItemListener**. El método **itemStateChanged** es invocado cuando el estado del checkbox es modificado.

Por ultimo deseo agregar un ejemplo realizado por un alumno que tiene todo lo referente a menús visto en esta sección, pero el utiliza mucho las clases del paquete **javax.swing**, en lugar de las de **java.awt**, pero es esencialmente lo mismo, veamos:


Menu con Swing
Archivo
Editar
Nuevo
Abrir
Cerrar
Radio Boton A
Radio Boton B
Check Box A
Check Box B
Submenu
Submenu 1
Submenu 2
: Nuevo
: Radio Boton A
: Buscar
: Copiar
: Radio Boton B
k box: Check Box A
k box: Check Box B
: Submenu 1


```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.JMenu;
4. import javax.swing.JMenuItem;
5. import javax.swing.JCheckBoxMenuItem;
6. import javax.swing.JRadioButtonMenuItem;
7. import javax.swing.ButtonGroup;
8. import javax.swing.JMenuBar;
9. import javax.swing.JTextArea;
10.import javax.swing.JScrollPane;
11.import javax.swing.JFrame;
12./**
13.  * Programa de Java que te enseña a utilizar componentes
14.  * del paquete java.awt. Este demuestra el uso de los
15.  * objetos de la clase ChecboxMenuItem.
16.  * @autor Gilardoni Jara
17.  * @version 1.0
18.  * Reviso: Oscar Alejandro González Bustamante
19.  * Archivo: MenuSwing.java
20.  */
21.public class MenuSwing extends JFrame
22.                      implements ActionListener, ItemListener {
23.    JTextArea salida;
24.    JScrollPane barrascroll;
25.    String newline = "\n";
26.
27.    public MenuSwing() {
28.        JMenuBar menuBar;
29.        JMenu menu, submenu;
30.        JMenuItem menuItem;
31.        JRadioButtonMenuItem radioBotom;
32.        JCheckBoxMenuItem checkBotom;
33.
34.        addWindowListener(new WindowAdapter() {
35.            public void windowClosing(WindowEvent e) {
36.                System.exit(0);
37.            }
38.        });
39.
40.        //adiciono los componentes , usando  BorderLayout.
41.        Container contentPane = getContentPane();
42.        salida = new JTextArea(5, 30);
43.        salida.setEditable(false);
44.        barrascroll = new JScrollPane(salida);
45.        contentPane.add(barrascroll, BorderLayout.CENTER);
46.
47.        //creo la barra del menu.
48.        menuBar = new JMenuBar();
49.        setJMenuBar(menuBar);
```

```
50.
51.         //Primer Item del menu.
52.         menu = new JMenu("Archivo");
53.         menuBar.add(menu);
54.
55.         //Primeros Grupos del Primer item del menu
56.         menuItem = new JMenuItem("Nuevo");
57.         menuItem.addActionListener(this);
58.         menu.add(menuItem);
59.         menuItem = new JMenuItem("Abrir");
60.         menuItem.addActionListener(this);
61.         menu.add(menuItem);
62.         menuItem = new JMenuItem("Cerrar");
63.         menuItem.addActionListener(this);
64.         menu.addSeparator();
65.         menu.add(menuItem);
66.         //radio boton del primer menu
67.         menu.addSeparator();
68.         ButtonGroup group = new ButtonGroup();
69.         radioBotom = new JRadioButtonMenuItem("Radio Boton A");
70.         radioBotom.setSelected(true);
71.         group.add(radioBotom);
72.         radioBotom.addActionListener(this);
73.         menu.add(radioBotom);
74.         radioBotom = new JRadioButtonMenuItem("Radio Boton B");
75.         group.add(radioBotom);
76.         radioBotom.addActionListener(this);
77.         menu.add(radioBotom);
78.
79.         //check box del primer menu
80.         menu.addSeparator();
81.         checkBotom = new JCheckBoxMenuItem("Check Box A");
82.         checkBotom.addItemListener(this);
83.         menu.add(checkBotom);
84.         checkBotom = new JCheckBoxMenuItem("Check Box B");
85.         checkBotom.addItemListener(this);
86.         menu.add(checkBotom);
87.
88.         //submenu
89.         menu.addSeparator();
90.         submenu = new JMenu(" Submenu");
91.         menuItem = new JMenuItem("Submenu 1 ");
92.         menuItem.addActionListener(this);
93.         submenu.add(menuItem);
94.         menuItem = new JMenuItem("Submenu 2");
95.         menuItem.addActionListener(this);
96.         submenu.add(menuItem);
97.         menu.add(submenu);
98.
```

```
99.         //Segundo menu de la barra
100.        menu = new JMenu("Editar");
101.        menuBar.add(menu);
102.
103.         //Grupo del segundo menu
104.        menuItem = new JMenuItem("Buscar");
105.        menuItem.addActionListener(this);
106.        menu.add(menuItem);
107.        menuItem = new JMenuItem("Copiar");
108.        menuItem.addActionListener(this);
109.        menu.add(menuItem);
110.    }
111.
112.    public void actionPerformed(ActionEvent e) {
113.        JMenuItem source = (JMenuItem)(e.getSource());
114.        String s =  "Ud. Presiono el Menu: " + source.getText();
115.
116.        salida.append( s + newline);
117.    }
118.
119.    public void itemStateChanged(ItemEvent e) {
120.        JMenuItem source = (JMenuItem)(e.getSource());
121.        String s = " Ud. Presiono el check box: " + source.getText();
122.                    salida.append(s + newline);
123.    }
124.
125.    public static void main(String[] args) {
126.        MenuSwing window = new MenuSwing();
127.
128.        window.setTitle("Menu con Swing");
129.        window.setSize(400, 300);
130.        window.setVisible(true);
131.    }
132.} // fin de la clase MenuSwing
```

Contenedores

Usted agrega componentes GUI a un contenedor utilizando el método **add()**. Hay dos tipos principales de contenedores: la clase **Window** y **Panel**.

Una ventana (**Window**) es un rectángulo que flota libremente en la pantalla. Hay dos tipos de ventanas. El **Frame** y el **Dialog**. Un **Frame** es una ventana con una barra de titulo y esquinas que pueden ser cambiadas de tamaño. Un **Dialog** es una simple ventana que no puede tener una barra de menú y aunque se puede mover no puede ser cambiada de tamaño.

El Panel debe estar contenido dentro de otro contenedor, o dentro de una ventana de un navegador Web. El Panel es una área rectangular dentro de la cual se puede colocar otros componentes. Se debe colocar un Panel dentro de una ventana de tipo Window (o dentro de una subclase de Window) para ser desplegado.

Posicionamiento y tamaño de los componentes en un contenedor.

La posición y el tamaño de un componente dentro de un contenedor es determinado por un administrador de diseño (layout manager). Un contenedor mantiene una referencia a una instancia en particular de un administrador de diseño. Cuando el contenedor necesita posicionar un componente, este invoca a el administrador de diseño para dicho fin. La misma delegación sucede cuando es decido el tamaño de un componente. El administrador de diseño toma todo el control sobre todos los componentes dentro del contenedor. Es responsable del cálculo y definición del tamaño preferido para el objeto en el contexto del tamaño actual de la pantalla.

Debido a que el administrador de diseño generalmente es responsable del tamaño y posición de los componentes dentro de su contenedor, usted generalmente no debería intentar establecer el tamaño o la posición de dichos componentes. Pero si lo tratara de hacer (usando métodos como setLocation(), setSize() o setBounds()), el administrador de diseño puede sobreescribir su decisión.

Si insiste en controlar el tamaño o la posición de los componentes sin utilizar los administradores de diseño, entonces puede deshabilitar al administrador de diseño mediante la llamada al siguiente método en su contenedor:

```
contenedor.setLayout(null);
```

Después de este paso, usted debe ahora usar **setLocation()**, **setSize()**, o **setBounds()** en todos los componentes para ubicarlos dentro del contendor.

Hacer esto resulta en un programa dependiente de la plataforma así que habrá diferencias entre diferentes sistemas basados en ventanas y en tamaños de tipos de fuentes. Una mejor solución sería crear una nueva subclase de **LayoutManager**.

Ejemplos de contenedores

Los Frames

Un **Frame** es una subclase de **Window**. Es una ventana con un titulo y puede cambiarse de tamaño en las esquinas. **Frame** hereda estas características de la clase **Container** así que se puede agregar componentes a un **Frame** utilizando el método **add()**. El administrador de diseño para un **Frame** omisión es el **BorderLayout**. Se puede cambiar el administrador de diseño utilizando el método **setLayout()**.


El constructor **Frame (String)** crea un nuevo objeto **Frame** invisible con el título especificado por el argumento **String**. Usted puede agregar todos los componentes a el **Frame** mientras este sea invisible.

El siguiente programa crea un **Frame** que tiene un titulo, tamaño y color de fondo específicos:

```
1. package oscar;
2. import java.awt.*;
3. import java.io.*;
4. /**
5.  * <p>Título: EjemploFrame.java</p>
6.  * <p>Descripción: Te enseña un contendor Frame  </p>
7.  * <p>Copyright: Es libre </p>
8.  * <p>Empresa: Patito feo </p>
9.  * @author Oscar Alejandro González Bustamante
10. * @version 1.0
11. */
12.
13.
14.public class  EjemploFrame {
15.   private Frame f;
16.
17.  public EjemploFrame() {
18.    f = new Frame(" Hola esto es un Frame ");
19.  }
20. public void lanzaelFrame() {
21.    f.setSize(200, 200);
22.    f.setBackground( Color.red );
23.    f.setLocation( new Point( 250, 250 ) );
24.    f.setVisible( true );
25. }
26.
27.  public static void main( String[] argumentos ) throws IOException {
28.    EjemploFrame  ventanaGUI = new  EjemploFrame();
29.    ventanaGUI.lanzaelFrame();
30.    for ( int i = 250 ; i > 0; i-- )
31.       ventanaGUI.f.setLocation( 250 - i , 250 - i );
32.
33.
34.    System.out.println( " Dar un teclazo para terminar");
35.    System.in.read();
36.    ventanaGUI.f.dispose();
37.    System.out.println("Chao chao bye bye !!");
38.    System.exit(0);
39.  }


40.}// fin de la clase EjemploFrame
```


Los Paneles

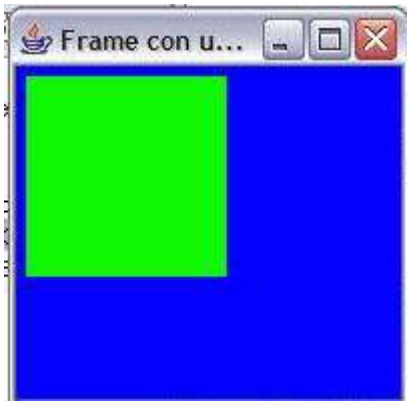
Un Panel es como un Frame, y provee del espacio para que el programador coloque cualquier componente GUI, incluyendo también otros paneles. Cada panel hereda de la clase Container todos sus métodos y atributos, y puede tener su propio administrador de diseño.

El siguiente programa crea un pequeño panel de color verde, y lo agrega a un frame.

```
1. package oscar;
2.
3. import javax.swing.*;
4. import java.awt.*;
5. import java.io.*;
6.
7. /**
8.  * <p>Título:FrameConPanel.java </p>
9.  * <p>Descripción: Te enseña a crear un panel dentro de un Frame </p>
10. * <p>Copyright:es libre</p>
11. * <p>Empresa: El patito feo Inc.</p>
12. * @author Oscar Alejandro González Bustamante.
13. * @version 1.0
14. */
15.
16.public class FrameConPanel extends JFrame {
17.  Panel panel1 = new Panel();
18.  FlowLayout flowLayout1 = new FlowLayout();
19.  FlowLayout flowLayout2 = new FlowLayout();
20.
21.  public FrameConPanel() {
22.    super( "Frame con un Panel");
23.    try {
24.      jbInit();
25.    }
26.    catch(Exception ex) {
27.      ex.printStackTrace();
28.    }
29.  }
30.
31.  void jbInit() throws Exception {
32.    this.getContentPane().setLayout(flowLayout2);
33.    panel1.setLayout(flowLayout1);
34.    panel1.setBackground(Color.green);
35.    panel1.setLocale(java.util.Locale.getDefault());
36.    this.getContentPane().setBackground(Color.blue);
37.    flowLayout1.setAlignment(FlowLayout.LEFT);
38.    flowLayout1.setHgap(50);
39.    flowLayout1.setVgap(50);
40.    flowLayout2.setAlignment(FlowLayout.LEFT);
41.    this.getContentPane().add(panel1, null);
42.  }
43.
44.  public static void main(String[] args) throws IOException {
45.    FrameConPanel frameConPanel = new FrameConPanel();
46.    frameConPanel.setLocation(100,100);
47.    frameConPanel.setSize(200,200);
48.    frameConPanel.setVisible( true );
49.    System.out.println("Dar un teclazo!!");
```

```
50.     System.in.read();
51.     frameConPanel.dispose();
52.     System.out.println("Adios");
53.   }
54.} // fin de la clase FrameConPanel
```

El resultado al ejecutar el programa con el comando **java FrameConPanel** lo muestra la siguente figura.



Layouts

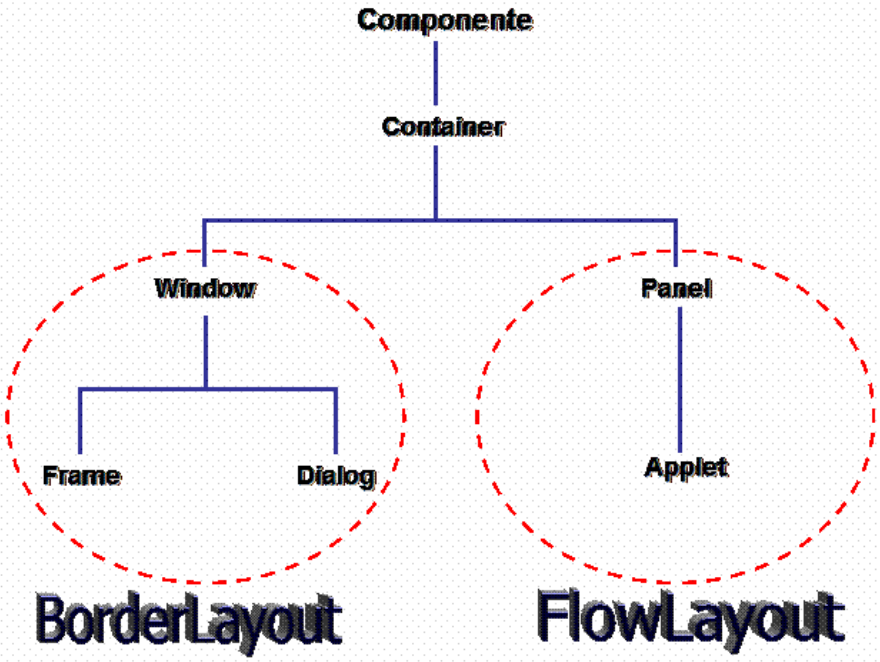
La distribución de los componentes en un contenedor usualmente esta controlada con un administrador de diseño (layout manager). Cada contenedor (como un Panel o un Frame) tiene por omisión un administrador de diseño asociado con este, el cual puede ser cambiado invocando al método **setLayout()**.

Cada administrador de diseño es responsable de definir el tamaño de cada componente en sus contenedores.

Los siguientes administradores de diseño son incluidos en el lenguaje de programación Java:

- **FlowLayout** - Es el administrador de diseño por default en el **Panel** y en el **Applet**. Coloca los componentes de izquierda a derecha y de arriba hacia abajo.
- **BorderLayout** - Es el administrador de diseño por default de **Window**, **Dialog** y **Frame**. Coloca los componentes en 5 regiones, Norte, Sur, Este, Oeste y Centro.
- **GridLayout** - Es un administrador de diseño que provee flexibilidad para colocar componentes a manera de celdillas de una hoja de cálculo.
- **CardLayout** - Es raramente usado, y coloca los componentes en capas, donde cada capa es como una carta de naipes, de ahí su nombre.
- **GridBagLayout** - Utilizado para distribuir los componentes como dentro de unas celdillas pero de diferente tamaño, es parecido al GridLayout.

La siguiente figura muestra los administradores de diseño (layout managers).




Ejemplos con Layouts

El FlowLayout

Es el administrador de diseño por default para la clase Panel y los componentes son agregados de izquierda a derecha. Su alineación por omisión es centrada y permite colocar los componentes a sus tamaños preferidos. Con los argumentos de el constructor de FlowLayout se permite definir los componentes para que fluyan de izquierda a derecha o de derecha a izquierda. También es posible especificar los espacios mínimos de separación entre cada componente. aquí tenemos los constructores:

FlowLayout()

FlowLayout(int align)

FlowLayout(int align, int hgap, int vgap)

Los valores para la alineación deben ser **FlowLayout.LEFT**, **FlowLayout.RIGHT**, o **FlowLayout.CENTER**, así por ejemplo:

setLayot(new FlowLayout(FlowLayout.RIGHT, 20, 40));

crea un administrador de diseño de tipo FlowLayout con una alineación centrada y con las unidades de espacios mínimos especificados, por default da 5 para espacio horizontal y 5 para el vertical.

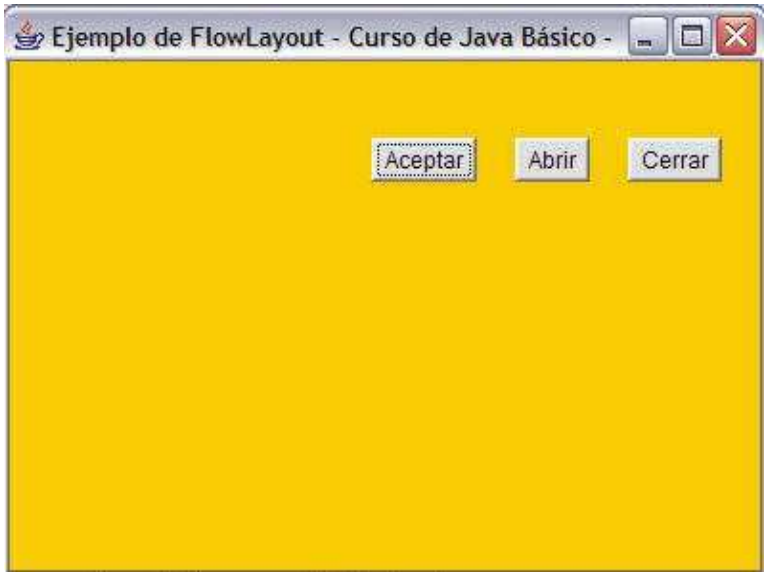
Veamos el siguiente ejemplo:

```
1. package oscar230604;
2.
3. import java.awt.*;
4. import java.awt.event.*;
5.
6. /**
7.  * <p>Título: EjemploFlowLayout.java </p>
8.  * <p>Descripción: Te enseña a usar el FlowLayout</p>
9.  * <p>Copyright: Totalmente libre</p>
10. * <p>Empresa: El patito Feo Inc.</p>
11. * @author Oscar Alejandro González Bustamante
12. * @version 1.0
13. */
14.
15.public class EjemploFlowLayout extends Frame {
16.   Button button1 = new Button();
17.   Button button2 = new Button();
18.   Button button3 = new Button();
19.   FlowLayout flowLayout1 = new FlowLayout();
20.
```


```
21.  public EjemploFlowLayout() {
22.    try {
23.      jbInit();
24.    }
25.    catch(Exception ex) {
26.      ex.printStackTrace();
27.    }
28.  }
29.
30.  void jbInit() throws Exception {
31.    button1.setLabel("Aceptar");
32.    button1.setLocale(java.util.Locale.getDefault());
33.    button1.addActionListener(new
   EjemploFlowLayout_button1_actionAdapter(this));
34.    button2.setLabel("Abrir");
35.    button3.setLabel("Cerrar");
36.    this.setSize( 400, 300 );
37.    this.setBackground(Color.orange);
38.    this.setResizable(true);
39.    this.setTitle("Ejemplo de FlowLayout - Curso de Java Básico -");
40.    this.setLayout(flowLayout1);
41.    flowLayout1.setAlignment(FlowLayout.RIGHT);
42.    flowLayout1.setHgap(20);
43.    flowLayout1.setVgap(40);
44.    this.add(button1, null);
45.    this.add(button2, null);
46.    this.add(button3, null);
47.  }
48.
```

```
49.  public static void main(String[] args) {
50.    EjemploFlowLayout ejemploFlowLayout = new EjemploFlowLayout();
51.    ejemploFlowLayout.setVisible( true );
52.  }
53.
54.  void button1_actionPerformed(ActionEvent e) {
55.
56.  }
57.}
58.
59.class EjemploFlowLayout_button1_actionAdapter implements
   java.awt.event.ActionListener {
60.  EjemploFlowLayout adaptee;
61.
62.  EjemploFlowLayout_button1_actionAdapter(EjemploFlowLayout adaptee) {
63.    this.adaptee = adaptee;
64.  }
65.  public void actionPerformed(ActionEvent e) {
66.    adaptee.button1_actionPerformed(e);
67.  }
68.} // fin de la clase EjemplodeFlowLayout
69.
```

En la figura siguiente podemos ver que al ejecutar el programa con el comando **java EjemploFlowLayout** nos presenta una ventana con tres botones con los espacios entre cada botón de 20 unidades por 40 unidades y con alineación hacia la derecha.



El BorderLayout

Es el administrador de diseño por default de la clase Frame. Los componentes son agregados a 5 regiones especificas dentro de la ventana o frame:

NORTH ocupa la parte de arriba

EAST ocupa el lado derecho

SOUTH ocupa la parte inferior

WEST ocupa la parte izquierda

CENTER ocupa la parte central

Cuando ajustamos verticalmente el tamaño de la ventana o frame, las regiones EAST, WEST y CENTER son ajustadas.

Cuando ajustamos horizontalmente el tamaño de la ventana o frame, las regiones NORT, SOUTH y CENTER son ajustadas.

Cuando añadimos botones a las posiciones relativas los botones no cambian si la ventana es cambiada de tamaño, pero los tamaños de los botones si cambian.

El siguiente constructor crea un administrador de diseño de tipo BorderLayout sin espacios entre sus componentes:

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

Usando el siguiente constructor podemos indicarle los espacios entre los componentes especificados por hgap y vgap:

```
BorderLayout( int hgap, int vgap );
```

Se deben agregar los componentes en las regiones específicas respetando mayúsculas y minúsculas ya que no es lo mismo **add(boton, BorderLayout.CENTER)** que **add(boton, BorderLayout.center)** en el administrador de diseño, o de otra forma no serán visibles. Si se quiere evitar esto se puede usar **add (boton, "center")**.

Si se deja una región sin utilizar, esta se comportará como si se hubiera preferido un tamaño de 0 x 0. La región **CENTER** seguirá apareciendo como fondo cuando incluso si no tiene componentes.


Solo se puede agregar un solo componente por cada una de las cinco regiones. Si se trata de agregar mas de una, solo la última agregada será la visible. Cuando deseamos agregar mas componentes por región usaremos la clase Panel, lo cual veremos después.

```
1. package oscar230604;
2.
3. import java.awt.*;
4. import java.awt.event.*;
5.
6. /**
7.  * <p>Título: EjemploBorderLayout.java </p>
8.  * <p>Descripción: Te enseña a usar el FlowLayout</p>
9.  * <p>Copyright: Totalmente libre</p>
10. * <p>Empresa: El patito Feo Inc.</p>
11. * @author Oscar Alejandro González Bustamante
12. * @version 1.0
13. */
14.
15.public class EjemploBorderLayout extends Frame {
16.   Button button1 = new Button();
17.   Button button2 = new Button();
18.   Button button3 = new Button();
19.   BorderLayout borderLayout1 = new BorderLayout();
20.   Button button4 = new Button();
21.   TextArea textArea1 = new TextArea();
22.
23.   public EjemploBorderLayout() {
24.     try {
25.       jbInit();
26.     }
27.     catch(Exception ex) {
28.       ex.printStackTrace();
29.     }
30.   }
31.
32.   void jbInit() throws Exception {
33.     button1.setLabel("Aceptar");
34.     button1.setLocale(java.util.Locale.getDefault());
35.     button1.addActionListener(new
   EjemploBorderLayout_button1_actionAdapter(this));
36.     button2.setLabel("Abrir");
37.     button3.setLabel("Cerrar");
38.     this.setSize( 400, 300 );
39.     this.setBackground(Color.cyan);
40.     this.setResizable(true);
41.     this.setTitle("Ejemplo de BorderLayout - Curso de Java Básico -");
42.     this.setLayout(borderLayout1);
43.     button4.setBackground(Color.red);
44.     button4.setLabel("Aplicar");
```

```
45.    borderLayout1.setHgap(20);
46.    borderLayout1.setVgap(40);
47.    textArea1.setColumns(20);
48.    textArea1.setEditable(false);
49.    textArea1.setEnabled(true);
50.    textArea1.setFont(new java.awt.Font("Arial", 3, 12));
51.    textArea1.setLocale(java.util.Locale.getDefault());
52.    textArea1.setSelectionEnd(20);
53.    textArea1.setSelectionStart(20);
54.    textArea1.setText("\t Hola esto es un TextArea  \n " +
55.                      "\t dentro de un Frame con un \n " +
56.                      "\t BorderLayout en la región \n " +
57.                      "\t CENTER");
58.    textArea1.setVisible(true);
59.    this.add(button1, BorderLayout.NORTH);
60.    this.add(button2, BorderLayout.WEST);
61.    this.add(button3, BorderLayout.SOUTH);
62.    this.add(button4, BorderLayout.EAST);
63.    this.add(textArea1, BorderLayout.CENTER);
64.  }
65.
66.  public static void main(String[] args) {
67.    EjemploBorderLayout ejemploBorderLayout = new EjemploBorderLayout();
68.    ejemploBorderLayout.setVisible( true );
69.  }
70.
71.  void button1_actionPerformed(ActionEvent e) {
72.
73.  }
74.}
75.
76.class EjemploBorderLayout_button1_actionAdapter implements
   java.awt.event.ActionListener {
77.  EjemploBorderLayout adaptee;
78.
79.  EjemploBorderLayout_button1_actionAdapter(EjemploBorderLayout adaptee) {
80.    this.adaptee = adaptee;
81.  }
82.  public void actionPerformed(ActionEvent e) {
83.    adaptee.button1_actionPerformed(e);
84.  }
85.} // fin de EjemploBorderLayout
86.
```

En la figura siguiente podemos ver que al ejecutar el programa con el comando **java EjemploBorderLayout** nos presenta una ventana con cuatro botones y un área de texto en la región **CENTER**.


Ejemplo de BorderLayout - Curso de Java Básí...
Aceptar
Abrir
Hola esto es un TextArea dentro de un Frame con un BorderLayout en la región CENTER
Aplicar
Cerrar


El GridLayout

Este administrador de diseño proporciona flexibilidad para colocar componentes en celdillas de izquierda a derecha y de arriba a abajo en una rejilla al estilo de una hoja electrónica de cálculo con filas y columnas. Por ejemplo un GridLayout con tres renglones y dos columnas puede ser creado mediante la sentencia new GridLayout (3, 2) lo que se traduce en 6 celdas.

Con un administrador de tipo GridLayout siempre se ignoran los tamaños preferidos para cada componente. El ancho de todas celdas es idéntico y es determinado mediante la división del ancho disponible sobre el número de columnas, así como el alto de todas las celdas es también determinado mediante el alto disponible entre el número de renglones.

El orden en el cual los componentes son agregados a la rejilla es determina la celda que ocupa. Los renglones de celdas son llenados de izquierda a derecha, como un texto de una página que es llenada con líneas de de arriba a abajo.

Veamos el siguiente ejemplo.

```
1. package oscar230604;
2.
3. import java.awt.*;
4. import javax.swing.*;
5.
6. /**
7.  * <p>Título: EjemploGridLayout.java </p>
8.  * <p>Descripción: Te enseña a usar el FlowLayout</p>
9.  * <p>Copyright: Totalmente libre</p>
10. * <p>Empresa: El patito Feo Inc.</p>
11. * @author Oscar Alejandro González Bustamante
12. * @version 1.0
13. */
14.
15.
16.public class EjemploGridLayout extends Frame {
17.  GridLayout gridLayout1 = new GridLayout();
18.  Button button1 = new Button();
19.  Button button2 = new Button();
20.  Button button3 = new Button();
21.  Button button4 = new Button();
22.  Button button5 = new Button();
23.  Button button6 = new Button();
24.
25.  public EjemploGridLayout() {
26.    try {
27.      jbInit();
28.    }
29.    catch(Exception ex) {
```

```
30.       ex.printStackTrace();
31.     }
32.   }
33.
34.   void jbInit() throws Exception {
35.     button1.setLabel("UNO");
36.     gridLayout1.setColumns(2);
37.     gridLayout1.setHgap(20);
38.     gridLayout1.setRows(3);
39.     gridLayout1.setVgap(40);
40.     this.setLayout(gridLayout1);
41.     button2.setLabel("DOS");
42.     button3.setLabel("TRES");
43.     button4.setLabel("CUATRO");
44.     button5.setLabel("CINCO");
45.     button6.setLabel("SEIS");
46.     this.setBackground(Color.red);
47.     this.setTitle("Ejemplo de Frame con GridLayout");
48.     this.add(button1, null);
49.     this.add(button2, null);
50.     this.add(button3, null);
51.     this.add(button4, null);
52.     this.add(button5, null);
53.     this.add(button6, null);
54.   }
55.
56.   public static void main(String[] args) {
57.     EjemploGridLayout ejemploGridLayout = new EjemploGridLayout();
58.     ejemploGridLayout.setSize( 400, 400 );
59.     ejemploGridLayout.setVisible ( true );
60.   }
61. } // fin de EjemploGridLayout
62.
```

En la figura siguiente podemos ver que al ejecutar el programa con el comando **java EjemploGridLayout** nos presenta una ventana con 3 filas por 2 columnas o 6 celdas botones son colocados en el orden en que se van agregando al contenedor y todos tienen los espacios especificados de 20 unidades en horizontal y 40 en vertical.

CENTRO DE INVESTIGACIÓN
EN COMPUTACIÓN

Java Avanzado