

# Introducción a los clientes ricos

## Índice

1	Introducción a AWT y Swing.....	2
1.1	AWT.....	2
1.2	Swing.....	5
2	Gestores de disposición.....	7
2.1	Tipos de gestores.....	7
2.2	Uso de los gestores.....	8
3	Modelo de eventos en Java.....	13
3.1	Oyentes para manejar eventos.....	14
3.2	Modos de definir un oyente.....	14
3.3	Uso de los "adapters".....	15
3.4	Diferencias entre AWT y Swing.....	19

En este tema veremos una introducción a lo que en algunos manuales se llaman "clientes ricos", es decir, clientes de aplicaciones que admiten una gran variedad de funcionalidades, debida a que la API para construirlos las ofrece. En concreto hablamos de la librería Swing de Java.

Para poder entenderla mejor, primero daremos un repaso a la librería AWT (la primera librería gráfica de Java), y su evolución a Swing, explicando los elementos básicos de una y otra para poder construir una aplicación.

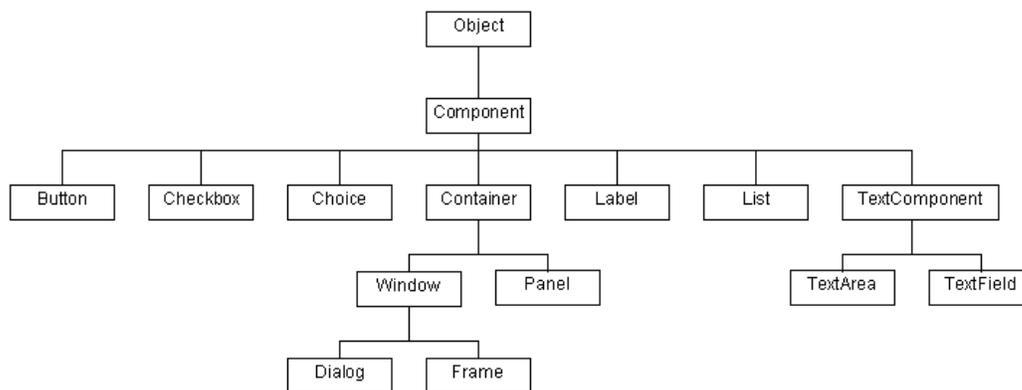
## 1. Introducción a AWT y Swing

### 1.1. AWT

**AWT** (*Abstract Windows Toolkit*) es la parte de Java que se emplea para construir **interfaces gráficas** de usuario. Este paquete ha estado presente desde la primera versión (la 1.0), aunque con la 1.1 sufrió un cambio notable. En la versión 1.2 se incorporó también a Java una librería adicional, **Swing**, que enriquece a AWT en la construcción de aplicaciones gráficas.

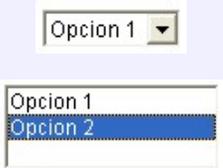
#### Controles de AWT

AWT proporciona una serie de **controles** (elementos gráficos como botones, listas, menús, etc), que podremos colocar en las aplicaciones visuales que implementemos. Dichos controles son subclases de la clase **Component**, y forman parte del paquete **java.awt**. Algunos de los controles más comunes son:



Estructura de clases de los controles más comunes

Pasamos a continuación a explicar estos y otros controles más detalladamente en la siguiente tabla:

<p><b>Component</b></p>	<p>La clase padre <i>Component</i> no se puede utilizar directamente. Es una clase abstracta, que proporciona algunos métodos útiles para sus subclases.</p>
<p><b>Botones</b></p> 	<p>Para emplear la clase <b>Button</b>, en el constructor simplemente indicamos el texto que queremos que tenga :</p> <pre>Button boton = new Button("Pulsame");</pre>
<p><b>Etiquetas</b></p> 	<p>Para utilizar <b>Label</b>, el uso es muy similar al botón: se crea el objeto con el texto que queremos darle:</p> <pre>Label etiq = new Label("Etiqueta");</pre>
<p><b>Areas de dibujo</b></p>	<p>La clase <b>Canvas</b> se emplea para heredar de ella y crear componentes personalizados. Accediendo al objeto <i>Graphics</i> de los elementos podremos darle la apariencia que queramos: dibujar líneas, pegar imágenes, etc:</p> <pre>Panel p = new Panel(); p.getGraphics().drawLine(0, 0, 100, 100); p.getGraphics().drawImage(...);</pre>
<p><b>Casillas de verificación</b></p> 	<p><b>Checkbox</b> se emplea para marcar o desmarcar opciones. Podremos tener controles aislados, o grupos de <i>Checkboxes</i> en un objeto <b>CheckboxGroup</b>, de forma que sólo una de las casillas del grupo pueda marcarse cada vez.</p> <pre>Checkbox cb = new Checkbox     ("Mostrar subdirectorios",     false); System.out.println ("Esta marcada: " +     cb.getState());</pre>
<p><b>Listas</b></p> 	<p>Para utilizar una <b>lista desplegable</b> (objeto <b>Choice</b>), se crea el objeto y se añaden, con el método <b>addItem(...)</b>, los elementos que queramos a la lista:</p> <pre>Choice ch = new Choice(); ch.addItem("Opcion 1"); ch.addItem("Opcion 2"); ... int i = ch.getSelectedIndex();</pre> <p>Para utilizar <b>listas fijas</b> (objeto <b>List</b>), en el constructor indicamos cuántos elementos son visibles. También podemos indicar si se permite seleccionar varios elementos a la vez. Dispone de muchos de los métodos que tiene <i>Choice</i> para añadir y consultar elementos.</p> <pre>List lst = new List(3, true); lst.addItem("Opcion 1"); lst.addItem("Opcion 2");</pre>

<p><b>Cuadros de texto</b></p> 	<p>Al trabajar con <b>TextField</b> o <b>TextArea</b>, se indica opcionalmente en el constructor el número de columnas (y filas en el caso de <i>TextArea</i>) que se quieren en el cuadro de texto.</p> <pre>TextField tf = new TextField(30); TextArea ta = new TextArea(5, 40); ... tf.setText("Hola"); ta.appendText("Texto 2"); String texto = ta.getText();</pre>
<p><b>Diálogos, ventanas y contenedores</b></p>	<p>Se tiene la clase <b>Frame</b> para crear ventanas principales de aplicaciones, y la clase <b>Dialog</b> para crear subventanas que dependan de ventanas principales:</p> <pre>Frame f = new Frame(); Dialog dlg = new Dialog();</pre> <p>Son lo que se llama <i>contenedores de nivel superior</i>. También se tienen contenedores intermedios, como la clase <b>Panel</b>, que nos permiten organizar los controles dentro de los contenedores superiores.</p> <p>Para añadir elementos (botones, listas, etiquetas, etc) en un contenedor, basta con llamar a su método <i>add</i> correspondiente:</p> <pre>Frame f = new Frame(); Panel p = new Panel();  Button b = new Button(); Label l = new Label();  p.add(b); p.add(l); f.add(p);</pre>
<p><b>Menús</b></p> 	<p>Para utilizar menús, se emplea la clase <b>MenuBar</b> (para definir la barra de menú), <b>Menu</b> (para definir cada menú), y <b>MenuItem</b> (para cada opción en un menú). Un menú podrá contener a su vez submenús (objetos de tipo <i>Menu</i>). También está la clase <b>CheckboxMenuItem</b> para definir opciones de menú que son casillas que se marcan o desmarcan.</p> <pre>MenuBar mb = new MenuBar(); Menu m1 = new Menu ("Menu 1"); Menu m11 = new Menu ("Menu 1.1"); Menu m2 = new Menu ("Menu 2"); MenuItem mil = new MenuItem ("Item 1.1"); MenuItem mill=new MenuItem ("Item 1.1.1"); CheckboxMenuItem mi2 =     new CheckboxMenuItem("Item 2.1"); mb.add(m1); mb.add(m2); m1.add(mil); m1.add(m11); m11.add(mill);</pre>

	<pre>m2.add(mi2);</pre> <p>Mediante el método <code>setMenuBar(...)</code> de <code>Frame</code> podremos añadir un menú a una ventana:</p> <pre>Frame f = new Frame(); f.setMenuBar(mb);</pre>
--	---

## 1.2. Swing

Anteriormente se ha visto una descripción de los controles *AWT* para construir aplicaciones visuales. En cuanto a estructura, no hay mucha diferencia entre los controles proporcionados por *AWT* y los proporcionados por *Swing*: éstos se llaman, en general, igual que aquéllos, salvo que tienen una "J" delante; así, por ejemplo, la clase `Button` de *AWT* pasa a llamarse `JButton` en *Swing*, y en general la estructura del paquete `javax.swing` es muy similar a la que tiene `java.awt` (aunque hay más controles, y se ofrecen más funcionalidades en *Swing*).

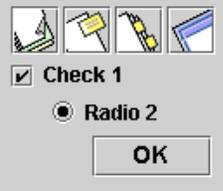
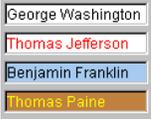
Pero yendo más allá de la estructura, existen importantes diferencias entre los componentes *Swing* y los componentes *AWT*:

- Los componentes *Swing* están escritos sin emplear código nativo, con lo que ofrecen más versatilidad multiplataforma (podemos dar a nuestra aplicación un aspecto que no dependa de la plataforma en que la estemos ejecutando).
- Los componentes *Swing* ofrecen más capacidades que los correspondientes *AWT*: los botones pueden mostrar imágenes, hay más facilidades para modificar la apariencia de los componentes, existen más componentes diferentes que en *AWT*, etc.
- Al mezclar componentes *Swing* y componentes *AWT* en una aplicación, se debe tener cuidado de emplear contenedores *AWT* con elementos *Swing*, puesto que los contenedores pueden solapar a los elementos (se colocan encima).

### Controles de Swing

Los controles en *Swing* tienen en general el mismo nombre que los de *AWT*, con una "J" delante. Así, el botón en *Swing* es `JButton`, la etiqueta es `JLabel`, etc. Hay algunas diferencias, como por ejemplo `JComboBox` (el equivalente a `Choice` de *AWT*), y controles nuevos. Vemos aquí un listado de algunos controles:

<b>JComponent</b>	La clase padre para los componentes <i>Swing</i> es <code>JComponent</code> , paralela al <code>Component</code> de <i>AWT</i> .
<b>Botones</b>	Se tienen botones normales ( <code>JButton</code> ), de verificación ( <code>JCheckBox</code> ), de radio ( <code>JRadioButton</code> ), etc, similares a los <code>Button</code> , <code>Checkbox</code> de <i>AWT</i> , pero con más posibilidades (se pueden añadir imágenes, etc).

	
<p><b>Etiquetas</b></p> 	<p>Las etiquetas son <b>JLabel</b>, paralelas a las <i>Label</i> de AWT pero con más características propias (iconos, etc).</p>
<p><b>Cuadros de texto</b></p> 	<p>Las clases <b>JTextField</b> y <b>JTextArea</b> representan los cuadros de texto en Swing, de forma parecida a los <i>TextField</i> y <i>TextArea</i> de AWT.</p>
<p><b>Listas</b></p> 	<p>Las clases <b>JComboBox</b> y <b>JList</b> se emplean para lo mismo que <i>Choice</i> y <i>List</i> en AWT, respectivamente.</p>
<p><b>Diálogos, ventanas y contenedores</b></p> 	<p>Las clases <b>JDialog</b> (y sus derivadas) y <b>JFrame</b> se emplean para definir diálogos y ventanas. Se tienen algunos cuadros de diálogo específicos, para elegir ficheros (<i>JFileChooser</i>), para elegir colores (<i>JColorChooser</i>), etc.</p> <p>Internamente a estos diálogos y ventanas, tenemos el contenedor <b>JPanel</b>, equivalente al <i>Panel</i> de AWT, para agrupar componentes.</p>

<p><b>Menús</b></p> 	<p>Con <b>JMenu</b>, <b>JMenuBar</b>, <b>JMenuItem</b>, se construyen los menús que se construían en AWT con <i>Menu</i>, <i>MenuBar</i> y <i>MenuItem</i>.</p>
---	---

## 2. Gestores de disposición

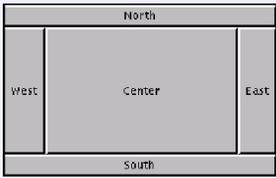
Para colocar los controles Java en los contenedores se hace uso de un determinado **gestor de disposición**. Dicho gestor indica cómo se colocarán los controles en el contenedor, siguiendo una determinada distribución. Para establecer qué gestor queremos, se emplea el método *setLayout(...)* del contenedor. Por ejemplo:

```
Panel panel = new Panel();
panel.setLayout(new BorderLayout());
```

Después, utilizamos el método *add(...)* del contenedor, para añadir controles. Dicho método tendrá unos parámetros u otros en función del tipo de gestor de disposición que se tenga, aunque, en cualquier caso, siempre se le pasa el control que queremos añadir al contenedor (más aparte otros parámetros adicionales, si se requieren).

### 2.1. Tipos de gestores

Veremos ahora los gestores más importantes:

<p><b>BorderLayout</b></p>  <p>(gestor por defecto para contenedores tipo <i>Window</i>)</p>	<p>Divide el área del contenedor en 5 zonas: Norte (<i>NORTH</i>), Sur (<i>SOUTH</i>), Este (<i>EAST</i>), Oeste (<i>WEST</i>) y Centro (<i>CENTER</i>), de forma que al colocar los componentes deberemos indicar en el método <i>add(...)</i> en qué zona colocarlo:</p> <pre>panel.setLayout(new BorderLayout()); Button btn = new Button("Pulsame"); panel.add(btn, BorderLayout.SOUTH);</pre> <p>Al colocar un componente en una zona, se colocará sobre el que existiera anteriormente en dicha zona (lo tapa).</p>
<p><b>FlowLayout</b></p> 	<p>Con este gestor, se colocan los componentes en fila, uno detrás de otro, con el tamaño preferido (<i>preferredSize</i>) que se les haya dado. Si no caben en una fila, se utilizan varias.</p> <pre>panel.setLayout(new FlowLayout());</pre>

(gestor por defecto para contenedores de tipo <i>Panel</i> )	<pre>panel.add(new Button("Pulsame"));</pre>
<b>GridLayout</b> 	<p>Este gestor sitúa los componentes en forma de tabla, dividiendo el espacio del contenedor en celdas del mismo tamaño, de forma que el componente ocupa todo el tamaño de la celda. Se indica en el constructor el número de filas y de columnas. Luego, al colocarlo, va por orden (rellenando filas de izquierda a derecha).</p> <pre>panel.setLayout(new GridLayout(2,2)); panel.add(new Button("Pulsame")); panel.add(new Label("Etiqueta"));</pre>
<b>Sin gestor</b>	<p>Si especificamos un gestor <i>null</i>, podremos colocar a mano los componentes en el contenedor, con métodos como <b>setBounds(...)</b>, o <b>setLocation(...)</b>:</p> <pre>panel.setLayout(null); Button btn = new Button("Pulsame"); btn.setBounds(0, 0, 100, 30); panel.add(btn);</pre> <p>Con <b>setBounds</b> indicamos las coordenadas X,Y de la esquina superior izquierda del control (2 primeros parámetros), y la anchura y altura del mismo (2 últimos parámetros). Observemos que de esta forma controlamos dónde va a ir exactamente el control, y con qué tamaño. El inconveniente es que si se redimensiona la ventana, el control sigue en el mismo lugar que indicamos al principio.</p>

## 2.2. Uso de los gestores

Los gestores de disposición son iguales para AWT y para Swing, si bien hay diferencias a la hora de establecer el gestor de disposición, y a la hora de añadir controles, con una y otra librería:

- Para establecer el gestor de disposición en un contenedor AWT, se utiliza el método **setLayout(...)** del contenedor:

```
Panel panel = new Panel();
panel.setLayout(new BorderLayout());

Frame f = new Frame();
f.setLayout(new GridLayout(3, 2));
```

Para Swing, si el contenedor es un *JPanel* se procede igual que con AWT, pero si estamos trabajando con un *JFrame* o un *JDialog*, o algún subtipo de éstos, el método **setLayout(...)** de estas clases no se recomienda usarlo, y en su lugar se recomienda llamar

al método **getContentPane(...)** del contenedor, lo que nos da acceso al objeto *Container* que representa, y una vez hecho esto, llamar a su método *setLayout(...)*:

```
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());
```

```
JFrame f = new JFrame();
f.getContentPane().setLayout(new GridLayout(3, 2));
```

- Para **añadir controles** a un contenedor AWT, utilizamos el método **add(...)** correspondiente del contenedor (dependiendo del tipo de gestor que tengamos, emplearemos un método *add(...)* u otro):

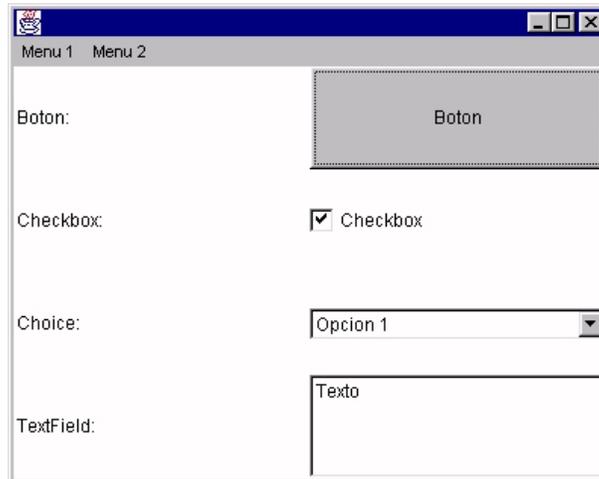
```
Button b = new Button("Hola");
Panel panel = new Panel();
panel.add(b);
Frame f = new Frame();
f.add(panel, BorderLayout.CENTER);
```

Para Swing, si el contenedor es un *JPanel* se procede igual que con AWT, pero si estamos trabajando con un *JFrame* o un *JDialog*, o algún subtipo de éstos, ocurre lo mismo que con *setLayout(...)*: se recomienda llamar al método **getContentPane(...)** del contenedor, lo que nos da acceso al objeto *Container* que representa, y una vez hecho esto, llamar a su método *add(...)*:

```
JButton b = new JButton("Hola");
JPanel panel = new JPanel();
panel.add(b);
JFrame f = new JFrame();
f.getContentPane().add(panel, BorderLayout.CENTER);
```

**IMPORTANTE:** Estas diferencias han sido resueltas **a partir de la versión 1.5 de Java**, y ya no es necesario utilizar el puente `getContentPane()` en los casos en los que era necesario utilizarlo antes. Sin embargo, conviene tener presente esta característica para versiones anteriores de Java.

**Ejemplo:** Vemos el aspecto de algunos componentes de AWT, y el uso de gestores de disposición en este ejemplo, donde se añaden una serie de controles (etiquetas, botones, checkboxes, etc) en una ventana. La aplicación no hace nada, simplemente muestra los controles:



```

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

/**
 * Este es un ejemplo donde se muestran algunos de los controles
 * de AWT
 */
public class EjemploAWT extends Frame
{
    /**
     * Constructor
     */
    public EjemploAWT()
    {
        setSize(400, 300);
        setLayout(new GridLayout(4, 2));

        // Menus

        MenuBar mb = new MenuBar();
        Menu m1 = new Menu ("Menu 1");
        Menu m11 = new Menu ("Menu 1.1");
        Menu m2 = new Menu ("Menu 2");
        MenuItem mi1 = new MenuItem ("Item 1.1");
        MenuItem mi11 = new MenuItem ("Item 1.1.1");
        CheckboxMenuItem mi2 = new CheckboxMenuItem("Item 2.1");
        mb.add(m1);
        mb.add(m2);
        m1.add(mi1);
        m1.add(m11);
        m11.add(mi11);
        m2.add(mi2);
        setMenuBar(mb);

        // Componentes

```

```
Label lblBoton = new Label("Boton:");
Label lblCheck = new Label("Checkbox:");
Label lblChoice = new Label("Choice:");
Label lblText = new Label("TextField:");

Button btn = new Button ("Boton");
Checkbox chk = new Checkbox ("Checkbox", true);
Choice ch = new Choice();
ch.addItem("Opcion 1");
ch.addItem("Opcion 2");
TextField txt = new TextField("Texto");

add(lblBoton);
add(btn);
add(lblCheck);
add(chk);
add(lblChoice);
add(ch);
add(lblText);
add(txt);

// Evento para cerrar la ventana
addWindowListener(new WindowAdapter()
{
    public void windowClosing (WindowEvent e)
    {
        System.exit(0);
    }
});
}

/**
 * Main
 */
public static void main (String[] args)
{
    EjemploAWT e = new EjemploAWT();
    e.show();
}
}
```

El aspecto de la aplicación utilizando componentes Swing es el que aparece a continuación:



```

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

/**
 * Este es un ejemplo donde se muestran algunos de los controles
 * de Swing
 */
public class EjemploSwing extends JFrame
{
    /**
     * Constructor
     */
    public EjemploSwing()
    {
        setSize(400, 300);
        setLayout(new GridLayout(4, 2));

        // Menus

        JMenuBar mb = new JMenuBar();
        JMenu m1 = new JMenu ("Menu 1");
        JMenu m11 = new JMenu ("Menu 1.1");
        JMenu m2 = new JMenu ("Menu 2");
        JMenuItem mil = new JMenuItem ("Item 1.1");
        JMenuItem mill = new JMenuItem ("Item 1.1.1");
        JCheckBoxMenuItem mi2 = new JCheckBoxMenuItem("Item 2.1");
        mb.add(m1);
        mb.add(m2);
        m1.add(mil);
        m1.add(m11);
        m11.add(mill);
        m2.add(mi2);
        setJMenuBar(mb);

        // Componentes

```

```
JLabel lblBoton = new JLabel("Boton:");
JLabel lblCheck = new JLabel("Checkbox:");
JLabel lblChoice = new JLabel("Choice:");
JLabel lblText = new JLabel("TextField:");

JButton btn = new JButton ("Boton");
JCheckBox chk = new JCheckBox ("Checkbox", true);
JComboBox ch = new JComboBox();
ch.addItem("Opcion 1");
ch.addItem("Opcion 2");
JTextField txt = new JTextField("Texto");

add(lblBoton);
add(btn);
add(lblCheck);
add(chk);
add(lblChoice);
add(ch);
add(lblText);
add(txt);

// Evento para cerrar la ventana
addWindowListener(new WindowAdapter()
{
    public void windowClosing (WindowEvent e)
    {
        System.exit(0);
    }
});

}

/**
 * Main
 */
public static void main (String[] args)
{
    EjemploSwing e = new EjemploSwing();
    e.show();
}
}
```

### 3. Modelo de eventos en Java

Entendemos por **evento** una acción o cambio en una aplicación que permite que dicha aplicación produzca una respuesta. El **modelo de eventos** de Java se descompone en dos grupos de elementos: las fuentes y los oyentes de eventos. Las **fuentes** son los elementos que generan los eventos (un botón, un cuadro de texto, etc), mientras que los **oyentes** son elementos que están a la espera de que se produzca(n) determinado(s) tipo(s) de evento(s) para emitir determinada(s) respuesta(s).

### 3.1. Oyentes para manejar eventos

Para poder gestionar eventos, necesitamos definir el **manejador de eventos** correspondiente, un elemento que actúe de oyente sobre las fuentes de eventos que necesitemos considerar. Cada tipo de evento tiene asignada una **interfaz**, de modo que para poder gestionar dicho evento, el manejador deberá implementar la interfaz asociada. Los oyentes más comunes son:

<b>ActionListener</b>	Para eventos de acción (pulsar un <i>Button</i> , por ejemplo)
<b>ItemListener</b>	Cuando un elemento ( <i>Checkbox</i> , <i>Choice</i> , etc), cambia su estado
<b>KeyListener</b>	Indican una acción sobre el teclado: pulsar una tecla, soltarla, etc.
<b>MouseListener</b>	Indican una acción con el ratón que no implique movimiento del mismo: hacer click, presionar un botón, soltarlo, entrar / salir...
<b>MouseMotionListener</b>	Indican una acción con el ratón relacionada con su movimiento: moverlo por una zona determinada, o arrastrar el ratón.
<b>WindowListener</b>	Indican el estado de una ventana

Cada uno de estos tipos de evento puede ser producido por diferentes fuentes.

### 3.2. Modos de definir un oyente

Supongamos que queremos realizar una acción determinada al pulsar un botón. En este caso, tenemos que asociar un *ActionListener* a un objeto *Button*. Veremos cómo podemos hacerlo:

#### 1. Que la propia clase que usa el control implemente el oyente

```
class MiClase implements ActionListener
{
    public MiClase()
    {
        ...
        Button btn = new Button("Boton");
        btn.addActionListener(this);
        ...
    }

    public void actionPerformed(ActionEvent e)
    {
        // Aqui va el codigo de la accion
    }
}
```

```
}  
}
```

## 2. Definir otra clase aparte que implemente el oyente

```
class MiClase  
{  
    public MiClase()  
    {  
        ...  
        Button btn = new Button("Boton");  
        btn.addActionListener(new MiOyente());  
        ...  
    }  
}  
  
class MiOyente implements ActionListener  
{  
    public void actionPerformed(ActionEvent e)  
    {  
        // Aqui va el codigo de la accion  
    }  
}
```

## 3. Definir una instancia interna del oyente

```
class MiClase  
{  
    public MiClase()  
    {  
        ...  
        Button btn = new Button("Boton");  
        btn.addActionListener(new ActionListener()  
        {  
            public void actionPerformed(ActionEvent e)  
            {  
                // Aqui va el codigo de la accion  
            }  
        });  
        ...  
    }  
}
```

### 3.3. Uso de los "adapters"

Algunos de los oyentes disponibles (como por ejemplo *MouseListener*) tienen varios métodos que hay que implementar si queremos definir el oyente. Este trabajo puede ser bastante pesado e innecesario si sólo queremos usar algunos métodos.

Una solución a esto es el uso de los *adapters*. Asociado a cada oyente con más de un método hay una clase *...Adapter* (para *MouseListener* está *MouseAdapter*, para *WindowListener* está *WindowAdapter*, etc). Estas clases implementan las interfaces con las que se asocian, de forma que se tienen los métodos implementados por defecto, y sólo tendremos que

sobreescribir los que queramos modificar.

Veamos la diferencia con el caso de *MouseListener*, suponiendo que queremos asociar un evento de ratón a un *Panel* para que haga algo al hacer click sobre él.

### 1. Mediante Listener:

```
class MiClase
{
    public MiClase()
    {
        ...
        Panel panel = new Panel();
        panel.addMouseListener(new MouseListener()
        {
            public void mouseClicked(MouseEvent e)
            {
                // Aqui va el codigo de la accion
            }

            public void mouseEntered(MouseEvent e)
            {
                // ... No se necesita
            }

            public void mouseExited(MouseEvent e)
            {
                // ... No se necesita
            }

            public void mousePressed(MouseEvent e)
            {
                // ... No se necesita
            }

            public void mouseReleased(MouseEvent e)
            {
                // ... No se necesita
            }
        });
        ...
    }
}
```

Vemos que hay que definir todos los métodos, aunque muchos queden vacíos porque no se necesitan.

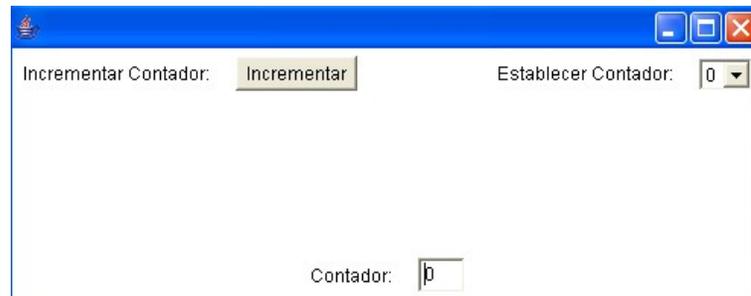
### 2. Mediante Adapter:

```
class MiClase
{
    public MiClase()
    {
        ...
    }
}
```

```
Panel panel = new Panel();
panel.addMouseListener(new MouseAdapter()
{
    public void mouseClicked(MouseEvent e)
    {
        // Aquí va el código de la acción
    }
});
...
}
```

Vemos que aquí sólo se añaden los métodos necesarios, el resto ya están implementados en *MouseAdapter* (o en el *adapter* que corresponda), y no hace falta ponerlos.

**Ejemplo:** Vemos el uso de oyentes en este ejemplo, donde se modifica el valor de un contador en un cuadro de texto a través de 2 eventos diferentes: pulsando un botón que lo incrementa en 1 cada vez, o eligiendo un valor predeterminado de un desplegable. Además, se tienen otros eventos implementados, como cambiar el texto de una etiqueta cuando se pasa el ratón por encima, o el de cerrar la ventana:



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

/**
 * Este es un ejemplo donde se muestran algunos de los controles
 * de AWT, y se modifica un contador bien pulsando un boton,
 * bien modificando el valor a mano, bien mediante un desplegable
 */
public class EjemploAWT2 extends Frame
{
    // Contador a modificar
    TextField txtCont;

    /**
     * Constructor
     */
    public EjemploAWT2()
    {
        setSize(500, 200);
    }
}
```

```

// ***** Panel con el contador *****

Panel panelCont = new Panel();
final Label lblCont = new Label("Contador:");
// Cambiar el texto de la etiqueta al pasar por encima
lblCont.addMouseListener(new MouseAdapter()
{
    public void mouseEntered(MouseEvent e)
    {
        lblCont.setText("En etiqueta!");
    }

    public void mouseExited(MouseEvent e)
    {
        lblCont.setText("Contador:");
    }
});
txtCont = new TextField("0");
panelCont.add(lblCont);
panelCont.add(txtCont);

// ***** Panel para el boton *****

Panel panelBoton = new Panel();
Label lblBoton = new Label("Incrementar Contador:");
Button btn = new Button("Incrementar");
// Incrementar el valor del contador al pulsarlo
btn.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        txtCont.setText("" +
(Integer.parseInt(txtCont.getText()) + 1));
    }
});
panelBoton.add(lblBoton);
panelBoton.add(btn);

// ***** Panel para el desplegable *****

Panel panelChoice = new Panel();
Label lblChoice = new Label("Establecer Contador:");
final Choice ch = new Choice();
for (int i = 0; i < 10; i++)
    ch.addItem("" + i);
// Establecer el contador al elemento seleccionado
ch.addItemListener(new ItemListener()
{
    public void itemStateChanged(ItemEvent e)
    {
        txtCont.setText(ch.getSelectedItem());
    }
});
panelChoice.add(lblChoice);

```

```
        panelChoice.add(ch);

        // Colocamos los paneles
        add(panelCont, "South");
        add(panelBoton, "West");
        add(panelChoice, "East");

        // Evento para cerrar la ventana
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing (WindowEvent e)
            {
                System.exit(0);
            }
        });
    }

    /**
     * Main
     */
    public static void main (String[] args)
    {
        EjemploAWT2 e = new EjemploAWT2();
        e.show();
    }
}
```

### 3.4. Diferencias entre AWT y Swing

---

El modelo de eventos explicado funciona exactamente igual para AWT que para Swing. Las diferencias en el modelo de eventos están en el hecho de que *Swing*, aparte de los oyentes vistos aquí, define otros subtipos de oyentes más específicos para controles suyos propios.

