



[Unidad 7

Componentes gráficos (AWT | Swing)



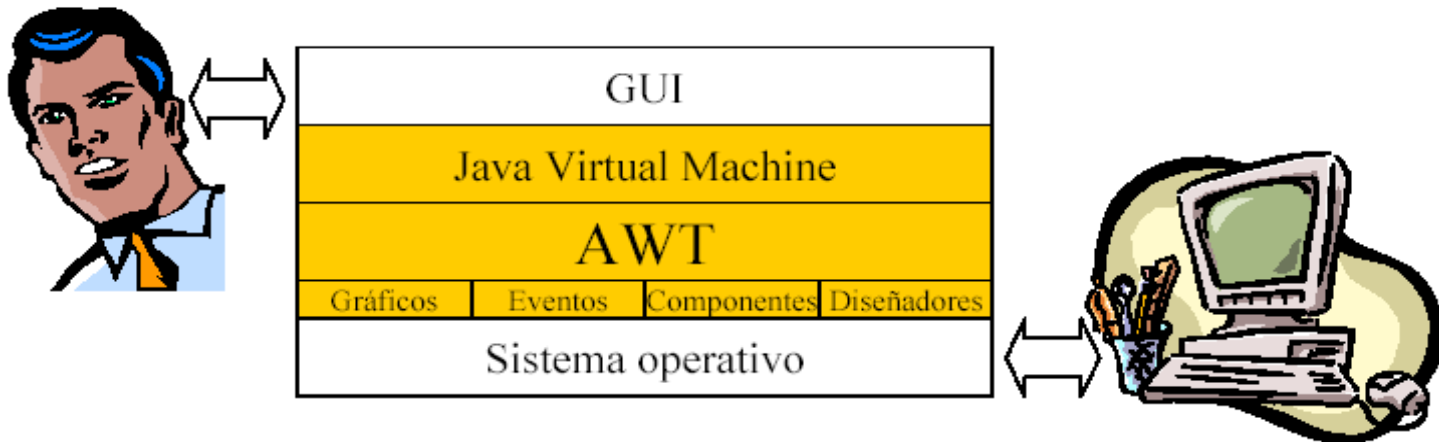
[Introducción

Java permite la creación de interfaces gráficas de interacción con el usuario (GUI, graphic user interface). Una GUI consiste en:

- Un conjunto de componentes de fácil diseño (botones, cajas de texto, etc.)
- Un modelo de control y manejo de eventos (teclado, ratón, etc.)
- Un conjunto de operaciones sobre gráficos, fuentes y formas
- Controladores de despliegue de componentes

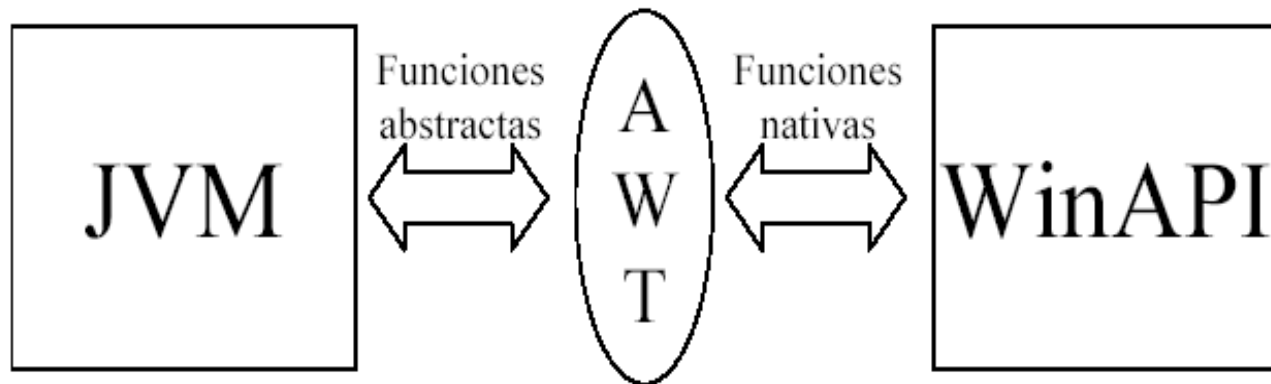
Introducción(...)

El conjunto de herramientas para aplicaciones GUI se conoce como Abstract Window Toolkit (AWT). Se dice que es *abstracto* ya que la máquina virtual depende de los recursos gráficos del sistema operativo.



[Introducción(...)

Particularmente, AWT interactúa con las librerías gráficas del sistema operativo en cuestión. Por ejemplo, Win32 Graphics Interface (WinAPI) en Windows y X11 en Linux. Por lo tanto, la AWT es un mapeo intermedio entre funciones de Java y funciones nativas del ambiente gráfico.





[Introducción(...)

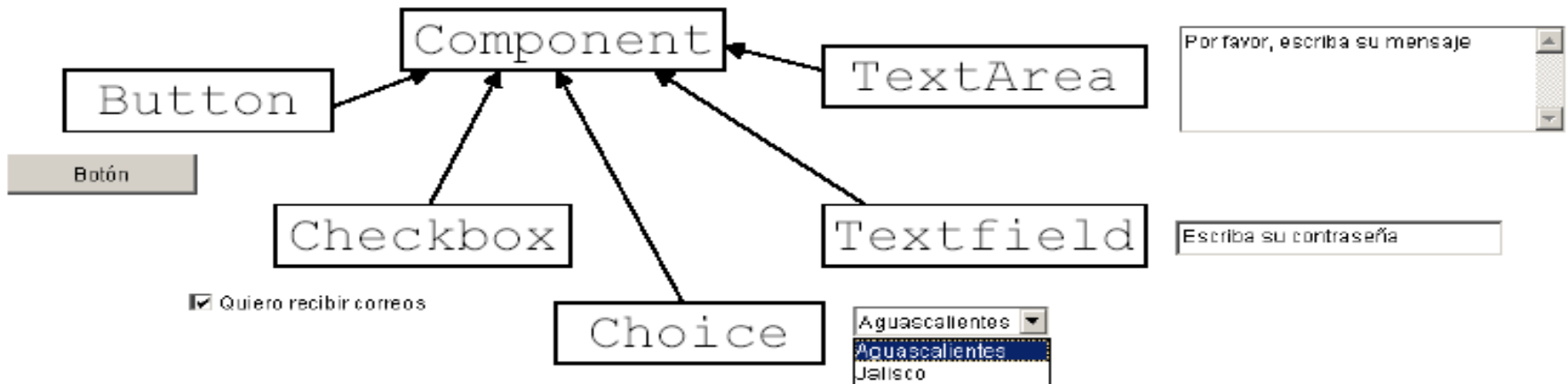
AWT es parte esencial del JDK y data desde la versión 1.0. Sin embargo, a raíz del crecimiento de Java, surgió la necesidad de desarrollar aplicaciones más sofisticadas con mayores capacidades. De esta idea nació **Swing**, que es un conjunto avanzado de componentes extendidos con mejor y mayor funcionalidad que las definidas en el estándar de AWT.

En este curso se verá mayormente componentes Swing.



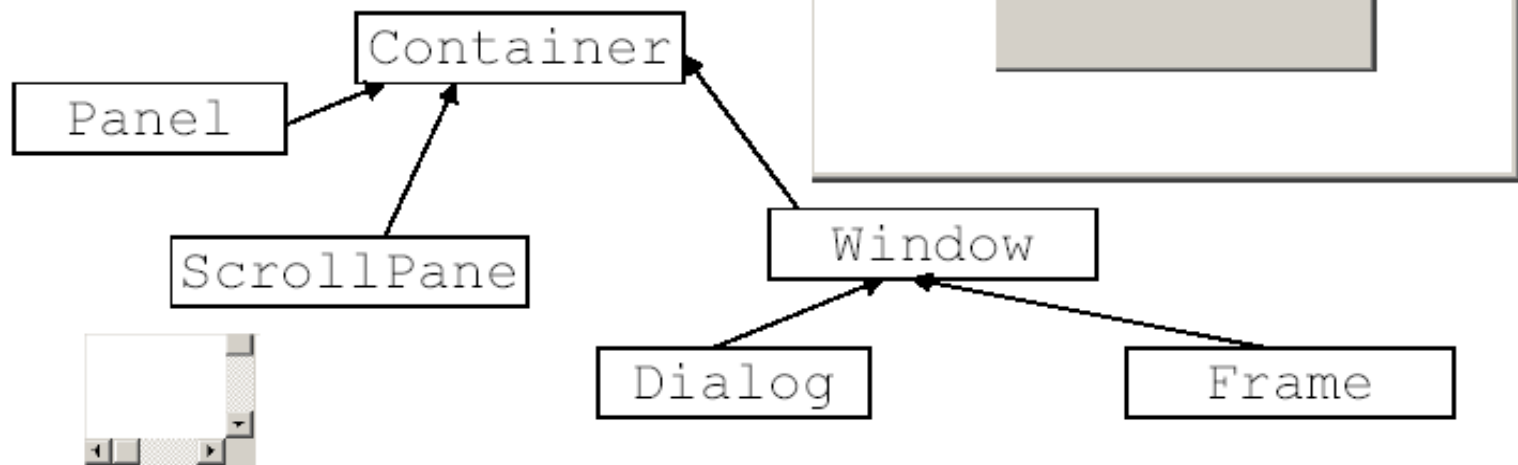
La clase Component

Esta es la “madre” de todos los componentes gráficos. Instancias de subclases de Component son objetos que tienen una representación gráfica, que pueden mostrarse en la pantalla y que pueden interactuar con el usuario. Dentro del paquete `java.awt` se encuentran componentes tales como:



La clase Container

Un contenedor es un componente que puede contener otros componentes. Su método principal es el de `add(Component)`. Entre las clases principales de Container se encuentran:





[La clase Container(...)

La clase Container es subclase de Component por lo que es posible anidar contenedores dentro de contenedores (excepto aquéllos contenedores que hereden de Window, p.e. Frame y Dialog).

```
import java.awt.*;

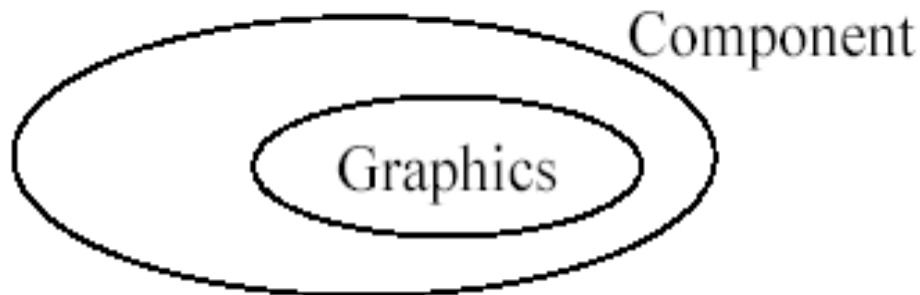
public Ventana extends Frame {
    public Ventana() {
        super("Ejemplito");
        add(new Panel());
        show();
    }
}
```




Espacios gráficos

Todo componente tiene asociado un espacio bidimensional para impresión de gráficos. En realidad, es un pedazo de memoria que es desplegable en pantalla. Tal espacio tiene que ver con operaciones directas sobre la pantalla tales como dibujo de figuras, líneas, textos, colores, etc.

Este espacio asociado es representado mediante un objeto de clase **Graphics**. Para efectuar operaciones sobre este objeto es necesario obtenerlo mediante el método `Graphics` `getGraphics()`, o sobrescribiendo el método `public void paint(Graphics)`.





[La clase Graphics

Este objeto asociado permite efectuar operaciones gráficas sobre un componente o un contenedor.

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

public class UsoGraficos extends JFrame {

    public UsoGraficos() {
        super("Uso de gráficos");
        setSize(200, 300);
        show();
    }
}
```

UsoGraficos.java



[La clase Graphics(...)

```
public void paint(Graphics g) {  
    g.drawString("Demo de gráficos", 10, 50);  
    //Dibujar carita  
    g.drawArc(50, 60, 50, 50, 0, 360);  
    g.drawArc(60, 70, 30, 30, 180, 180);  
    g.fillOval(65, 75, 5, 5);  
    g.fillOval(80, 75, 5, 5);  
    //Dibujar cuerpo  
    g.drawLine(75, 110, 75, 200);  
    //Brazos  
    g.drawLine(75, 120, 45, 160);  
    g.drawLine(75, 120, 105, 160);  
    //Piernas  
    g.drawLine(75, 200, 45, 240);  
    g.drawLine(75, 200, 105, 240);  
}
```



[La clase Graphics(...)

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





Los métodos paint y update

Cada vez que una ventana se convierte en la ventana activa actual (aquella que tiene la atención primaria del usuario), el método `paint(Graphics)` es ejecutado automáticamente. Cuando es necesario “refrescar” la información gráfica, basta con mandar llamar el método `repaint()` que implícitamente ejecuta `update(Graphics)`. Por lo tanto, si una aplicación requiere constantemente cambiar su contenido gráfico, deberá sobrescribir el método `update(Graphics)`.



[Creando un reloj análogo

Se trata de implementar un cuadro que muestre la hora en forma analógica. Sea Reloj nuestro cuadro deseado. La definición inicial se presenta enseguida:

```
import javax.swing.*;

public class Reloj extends JFrame {

    public Reloj() {
        super("Relojito");

        setResizable(false);
        setSize(200, 200);
        show();
    }
}
```



[Creando un reloj análogo(...)

Cada segundo el segundero actualiza su posición. El minuterio lo hace cada 60 segundos el minuterio y el horario lo hace cada 60 minutos. Por lo tanto, la actualización del segundero es más frecuente.

Es innecesario actualizar gráficamente la posición de todos los indicadores, bastaría solo hacerlo para el segundero. Esta optimización se logra con una técnica conocida como double buffereo.



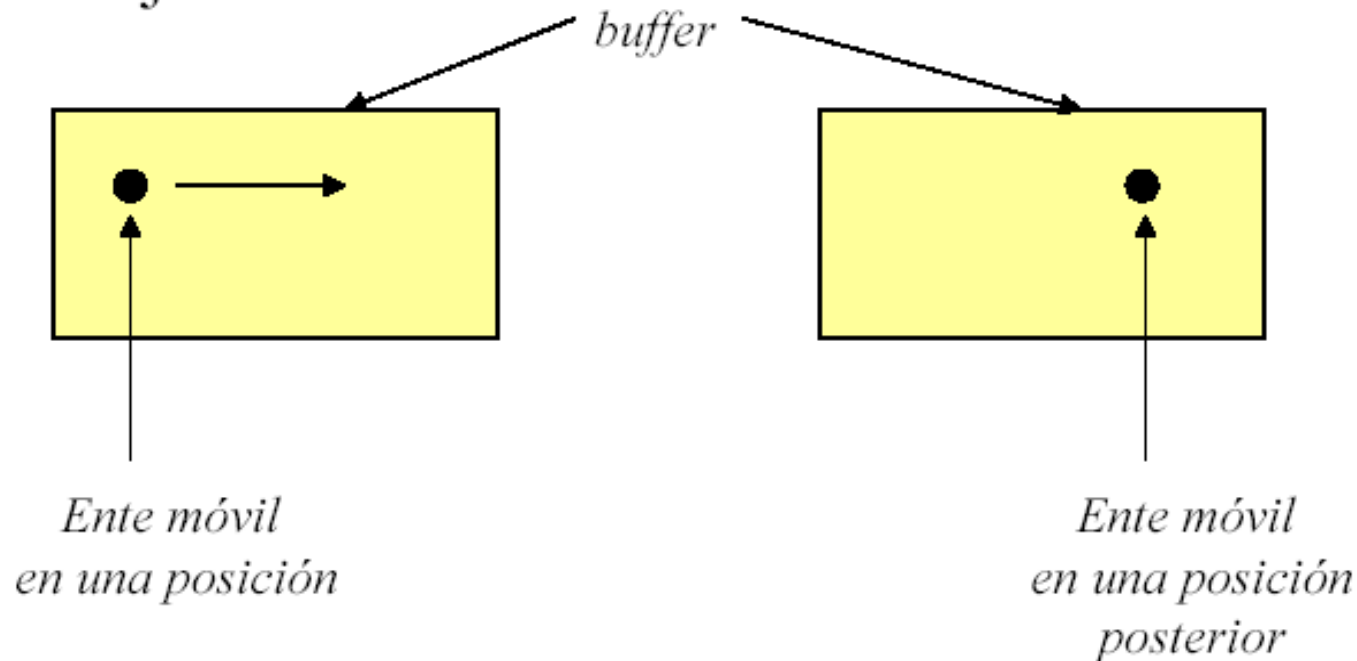
[Doble buffereo

Esta técnica de optimización gráfica permite actualizar el movimiento constante de un elemento dentro de una imagen común sin necesidad de volverla a pintar por completo. Por ejemplo, en un juego de ping-pong, no es conveniente volver a construir gráficamente la mesa debido a que es estática. Sin embargo, sí tiene sentido hacerlo para la pelota.

El **doble buffereo** consiste en tener en un espacio gráfico temporal (*buffer*) la imagen estática, es decir, la que no tiene movimiento y pintar sobre este buffer aquella porción gráfica dinámica.

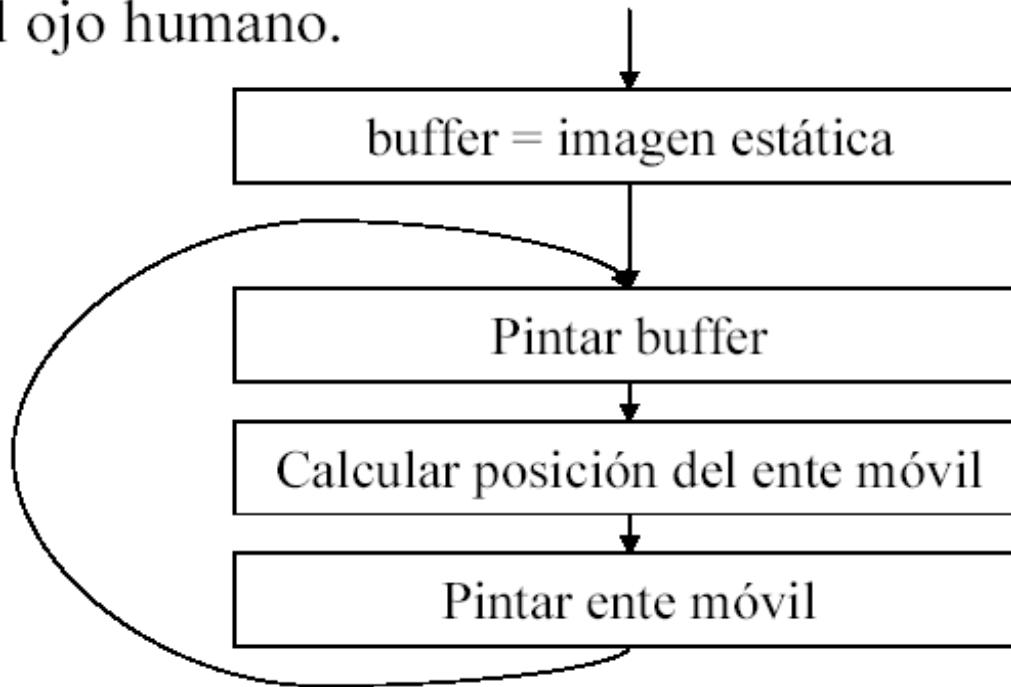
[Doble buffereo(...)

Esto hace que el repintado sea más rápido y la sensación de movimiento sea nítida e imperceptible al ojo humano.



[Doble buffereo(...)]

Esto hace que el repintado sea más rápido y la sensación de movimiento sea nítida e imperceptible al ojo humano.





[Reloj análogo(...)

La estrategia es:

- Crear la imagen estática que contenga el horario y el minuterero
- A cada segundo, se pinta este buffer y se actualiza la posición del segundero
- Cuando termine el minuto, se actualiza la imagen estática.



[Reloj análogo(...)

```
private Image fondo;
private Image buffer;
...
public void paint(Graphics g) {
    if (fondo == null) {
        fondo = createImage(getWidth(), getHeight());
        //Pintar el círculo del reloj
        Graphics gfondo = fondo.getGraphics();
        gfondo.setClip(0, 0, getWidth(), getHeight());
        gfondo.drawOval((getWidth()-100)/2, (getHeight()-
100)/2, 100, 100);
    }

    update(g);
}
```



[Reloj análogo(...)

```
public void update(Graphics g) {  
  
    g.setClip(0, 0, getWidth(), getHeight());  
    Calendar cal = Calendar.getInstance();  
  
    if (cal.get(Calendar.MINUTE) != min) {  
        //Regenerar la imagen de fondo  
        hora = cal.get(Calendar.HOUR);  
        min = cal.get(Calendar.MINUTE);  
        //Crear la imagen estática  
        buffer = createImage(getWidth(), getHeight());  
        Graphics gbuffer = buffer.getGraphics();  
        gbuffer.setClip(0, 0, getWidth(), getHeight());  
        gbuffer.drawImage(fondo, 0, 0, this);  
        gbuffer.fillArc((getWidth()-90)/2+5, (getHeight()-90)/2+5,  
80, 80, angulo12(hora), 3);  
        gbuffer.fillArc((getWidth()-100)/2+5, (getHeight()-100)/2+5,  
90, 90, angulo60(min), 3);  
    }  
}
```



[Reloj análogo(...)

```
//Pintar buffer  
g.drawImage(buffer, 0, 0, this);  
sec = cal.get(Calendar.SECOND);  
//Pintar ente móvil  
g.fillArc((getWidth()-100)/2+5, (getHeight()-100)/2+5,  
90, 90, angulo60(sec), 3);  
}
```



[Reloj análogo(...)

Ahora bien, es necesario agregarle movimiento a nuestro reloj. Como ya se vió, lo lograremos gracias a un hilo.

Sin embargo, nuestra clase Reloj hereda de JFrame.

¿Cómo lograremos que esta misma clase se comporte como un hilo?



Más sobre hilos

La segunda forma para definir hilos es haciendo que una clase implemente la interface `java.lang.Runnable` e internamente asociar nuestro código con un hilo ejecutable. Al igual que `Thread` se deberá implementar el método `run()`

```
public class NomClase implements Runnable {  
    public void run() {  
        //Código del hilo  
    }  
}
```




Más sobre hilos(...)

¿Cómo arrancar el hilo? Asociando lo a un objeto de clase Thread.

```
public class NomClase implements Runnable {  
    private Thread hilo;  
  
    public NomClase() {  
        hilo = new Thread(this);  
        hilo.start();  
    }  
    public void run() {  
        //Código del hilo  
    }  
}
```



[Reloj analógico(...)

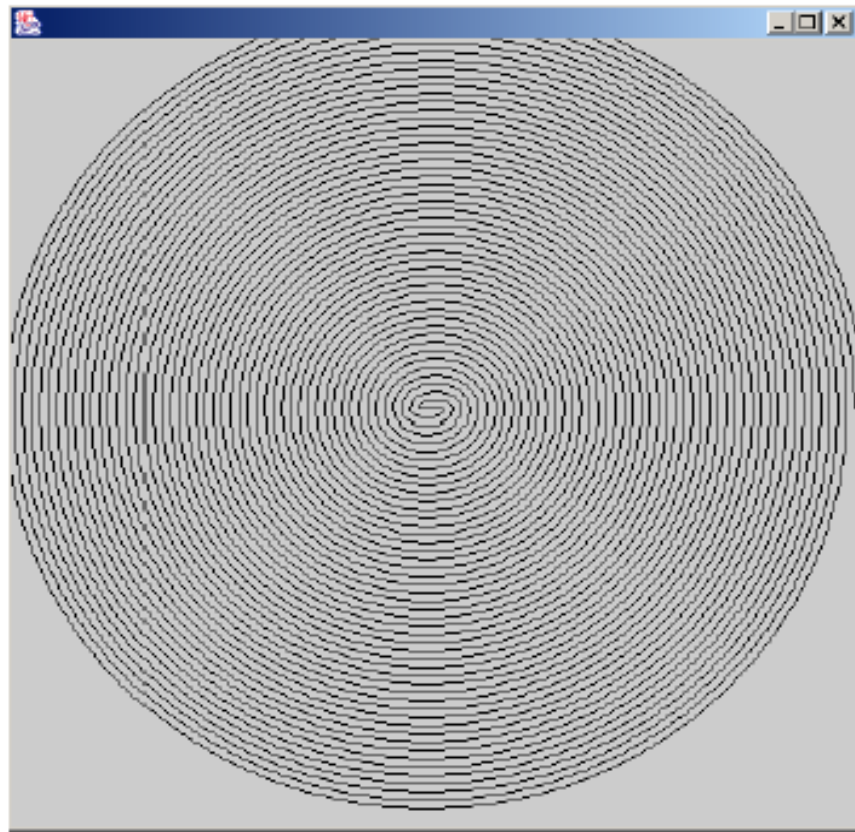
```
public class Reloj extends JFrame implements Runnable {
    //Hilo
    private Thread thr;

    public Reloj() {
        ...
        thr = new Thread(this);
        thr.start();
    }

    public void run() {
        while (true) {
            try {
                thr.sleep(1000);
            } catch (InterruptedException ex) {
            }
            repaint();
        }
    }
}
```

[Ejercicio 1

Implemente una ventana que imprima la espiral de Arquímedes a velocidad lenta.





[Ejercicio 2

Desarrolle un componente (subclase de Component) que permita la impresión de un diagrama de pastel. Esta clase se le asociará un objeto de clase Pregunta que será la clase base para calcular la estadística. Tendrá otro método que permitirá agregar preguntas contestadas de tal forma que el diagrama se va actualizando conforme la respuesta de la pregunta recibida.



[La clase Image

Java permite la impresión gráfica de imágenes (en formato GIF, PNG y JPG) mediante objetos de clase Image. Existen dos formas de crear imágenes:

- Obteniendo la información gráfica mediante el método `createImage(int, int)`,
- Leyendo la información gráfica desde un archivo
De la segunda forma, el archivo deberá tener el formato correspondiente.



[La clase Image(...)

- A través, del método `createImage(int, int)` de una instancia `Component`

```
Image img = createImage(getWidth(), getHeight());
```

- A través de un archivo

```
Image img =  
Toolkit.getDefaultToolkit().getImage("imagen.gif");
```

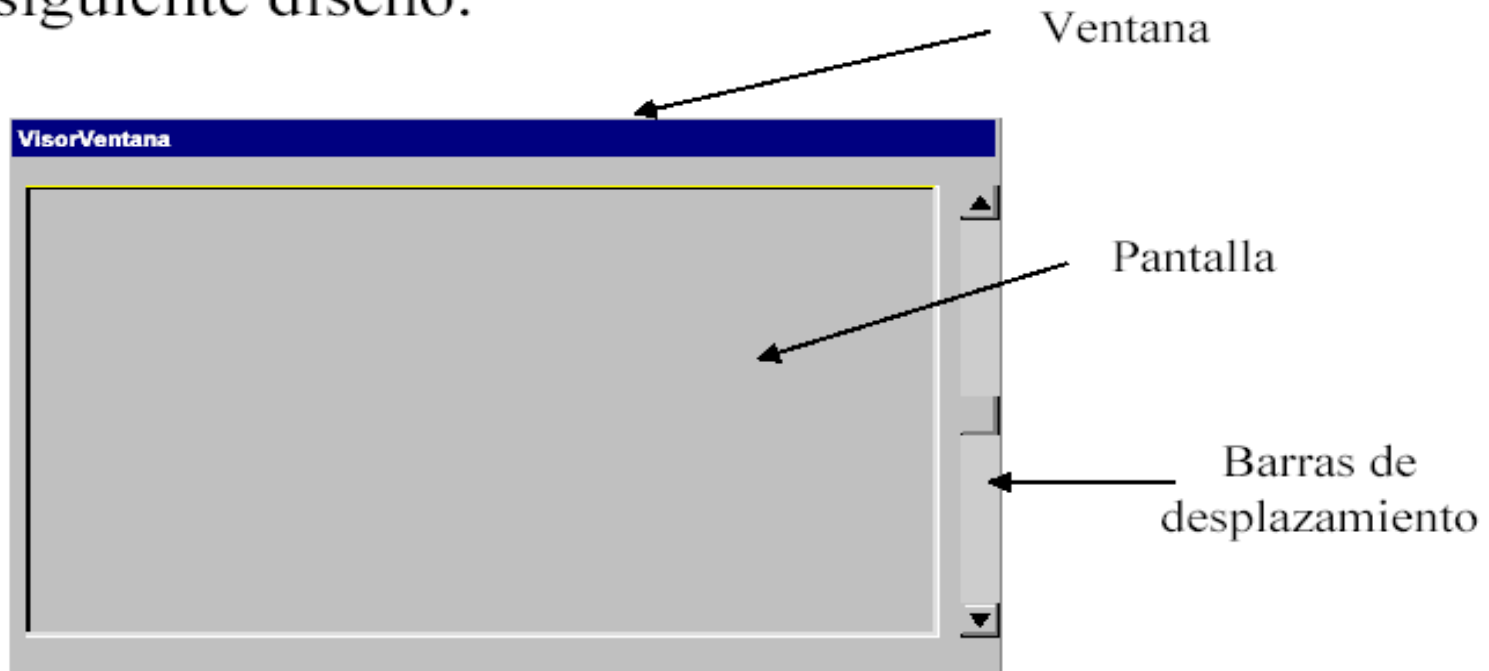
Para imprimir la imagen a través del método

`drawImage(Image, int, int, ImageObserver)`

```
public void paint(Graphics g) {  
    g.drawImage(img, 0, 0, this);  
}
```

Creando un visor de imágenes

Un visor de imágenes pudiera crearse con el siguiente diseño:





[Creando un visor (...)

La pantalla puede ser un simple panel sencillo (`JPanel`) contenido en un panel con barras de desplazamiento (`JScrollPane`). Esto con la finalidad de que si es muy grande, el usuario pueda deslizarse aún que la ventana (`JFrame`) no sea lo suficientemente grande para desplegar toda la imagen.



[Creando un visor (...)

La pantalla entonces se definiría así:

```
public class Pantalla extends JPanel {  
  
    private Image imagen;  
  
    public void paint(Graphics g) {  
        super.paint(g); //Desplegar para obtener dimensiones reales en pantalla  
        Dimension tam = new Dimension(imagen.getWidth(this),  
imagen.getHeight(this));  
        setPreferredSize(tam);  
        setMinimumSize(tam);  
        setMaximumSize(tam);  
        setSize(tam);  
        update(g);  
    }  
  
    public void update(Graphics g) {  
        g.drawImage(imagen, 0, 0, this);  
    }  
}
```



[Creando un visor (...)

El visor de imágenes queda definido como:

```
public class VisorImagen extends JFrame {  
  
    private JScrollPane panel;  
    private Pantalla pantalla;  
  
    public VisorImagen(String archivo) {  
  
        super("Visor imagen");  
  
        Image img = Toolkit.getDefaultToolkit().getImage(archivo);  
        pantalla = new Pantalla(img);  
        panel = new JScrollPane(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,  
                                JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);  
        getContentPane().add(panel);  
        panel.setViewportViewView(pantalla);  
  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setSize(400, 300);  
        show();  
    }  
}
```



[Componentes Swing

Los componentes Swing son subclases de Component que ofrecen un mejor comportamiento y resolución que los componentes simples AWT.

A continuación veremos los componentes más clásicos que se encuentran en el paquete Swing (javax.swing).



Menús

Objetos de la clase `JMenuBar` que define una barra de menús en la parte superior de la ventana (`JFrame`). Cada menú es un objeto de clase `JMenu`, y cada menú posee opciones representadas como objetos de clase `MenuItem`.

```
//Construcción menu
menu = new JMenu("Archivo");
menu.add(new JMenuItem("Salir", 'S'));
barramenu = new JMenuBar();
barramenu.add(menu);
setJMenuBar(barramenu); //Asignar al JFrame
```





[Botones

Objetos de la clase JButton. En su cara ofrecen un texto alusivo a la acción que ejecuta una vez presionado el botón



```
//Construcción botón  
boton = new JButton("Activar");
```



[Etiquetas

Objetos de la clase JLabel. Funcionan para impresión de texto.

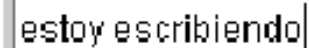
Nombre:

```
//Construcción etiqueta  
etiqueta = new JLabel("Nombre:", JLabel.RIGHT);
```



[Cajas de texto

Objetos de la clase JTextField. Permiten el ingreso de datos textuales (cadenas).

A screenshot of a Java Swing JTextField component. It is a rectangular box with a thin gray border. Inside the box, the text "estoy escribiendo" is written in a black, monospaced font. A vertical cursor line is positioned at the end of the text, indicating the current position for typing.

```
//Construcción caja texto  
cajatexto = new JTextField(20);
```



[Áreas de texto]

Objetos de la clase JTextArea. Permiten el ingreso de datos textuales multilineas (edición).

```
//Construcción área de texto  
areatexto = new JTextArea("Escriba aquí su  
comentario...", 5, 20);  
areatexto.setLineWrap(true);
```

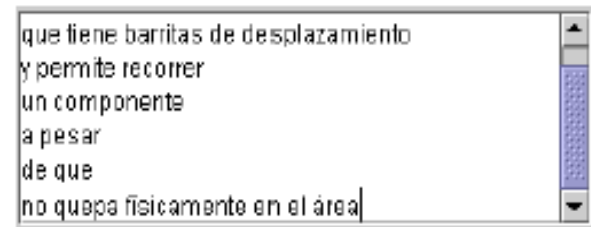
A screenshot of a Java Swing JTextArea component. It is a rectangular box with a thin gray border. The text "Escriba aquí su comentario..." is displayed at the top left of the box in a small, black, monospaced font. The rest of the box is empty, showing the white background of the text area.

Escriba aquí su comentario...



Paneles con barras de scroll

Objetos de la clase JScrollPane. Contenedores de componentes (como tablas y áreas de texto) que permiten desplazarse a lo largo y ancho de los mismos.



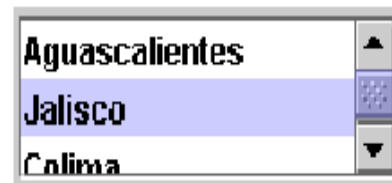
```
//Construcción panel con barra de scroll
panelscroll = new JScrollPane(
    JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
    JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
panelscroll.setViewportView(areatexto);
```



[Listas

Objetos de la clase JList. Permiten el despliegue de opciones condensadas en un recuadro deslizable. De acuerdo a su configuración ofrece selección sencilla o múltiple.

```
//Construcción lista  
Vector opciones = new Vector();  
opciones.addElement("Aguascalientes");  
opciones.addElement("Jalisco");  
opciones.addElement("Colima");  
opciones.addElement("Yucatan");  
opciones.addElement("Quintana Roo");  
opciones.addElement("Sinaloa");  
lista = new JList(opciones);
```

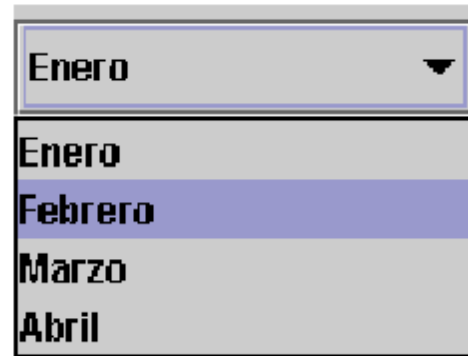




[Combos de opciones

Objetos de la clase JComboBox. Permiten el despliegue de opciones condensadas en un menú flotante.

```
//Construcción combo  
combo = new JComboBox();  
combo.addItem("Enero");  
combo.addItem("Febrero");  
combo.addItem("Marzo");  
combo.addItem("Abril");
```

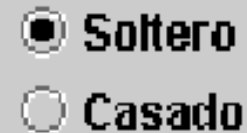




[Botones de opción

Objetos de la clase `JRadioButton`. Botones seleccionables.
Cada uno de ellos representa una opción.

```
//Construcción botones de opción  
opcion1 = new JRadioButton("Soltero", true);  
opcion2 = new JRadioButton("Casado");  
options = new ButtonGroup();  
options.add(opcion1);  
options.add(opcion2);
```





[Cajas de opción

Objetos de la clase JCheckBox. Pequeños cuadros booleanos seleccionables.

```
//Construcción cajas de opción  
checkbox = new JCheckBox("Deseo recibir chistes  
por email", true);
```

☒ Deseo recibir chistes por email



[Tablas

Objetos de la clase JTable. Permiten la presentación de información en una estructura matricial ordenado por columnas y renglones

```
//Construcción tabla  
Vector columnas = new Vector();  
columnas.addElement("Id.Encuesta");  
columnas.addElement("Tema");
```

| Id.Encuesta | Tema |
|-------------|-------------------|
| 1 | Chivas de Corazon |
| 2 | Elecciones 2003 |

[Tablas(...)]

```
Vector registros = new Vector();  
Vector registro1 = new Vector();  
registro1.addElement(new Integer(1));  
registro1.addElement("Chivas de Corazon");
```

```
Vector registro2 = new Vector();  
registro2.addElement(new Integer(2));  
registro2.addElement("Elecciones 2003");
```

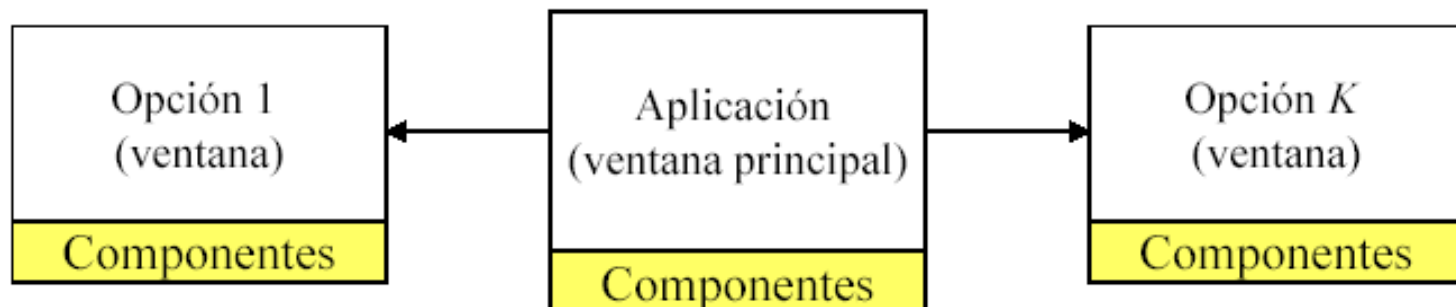
```
registros.addElement(registro1);  
registros.addElement(registro2);
```

```
tabla = new JTable(registros, columnas);
```

| Id.Encuesta | Tema |
|-------------|-------------------|
| 1 | Chivas de Corazon |
| 2 | Elecciones 2003 |

Un *componente* (Component) es parte de una aplicación que contiene otros componentes. Se dice entonces que la aplicación debe ser un *contenedor* (Container) de componentes.

Normalmente, una aplicación se constituye de una ventana principal la cual contendrá componentes y podrá ligarnos a otras ventanas.





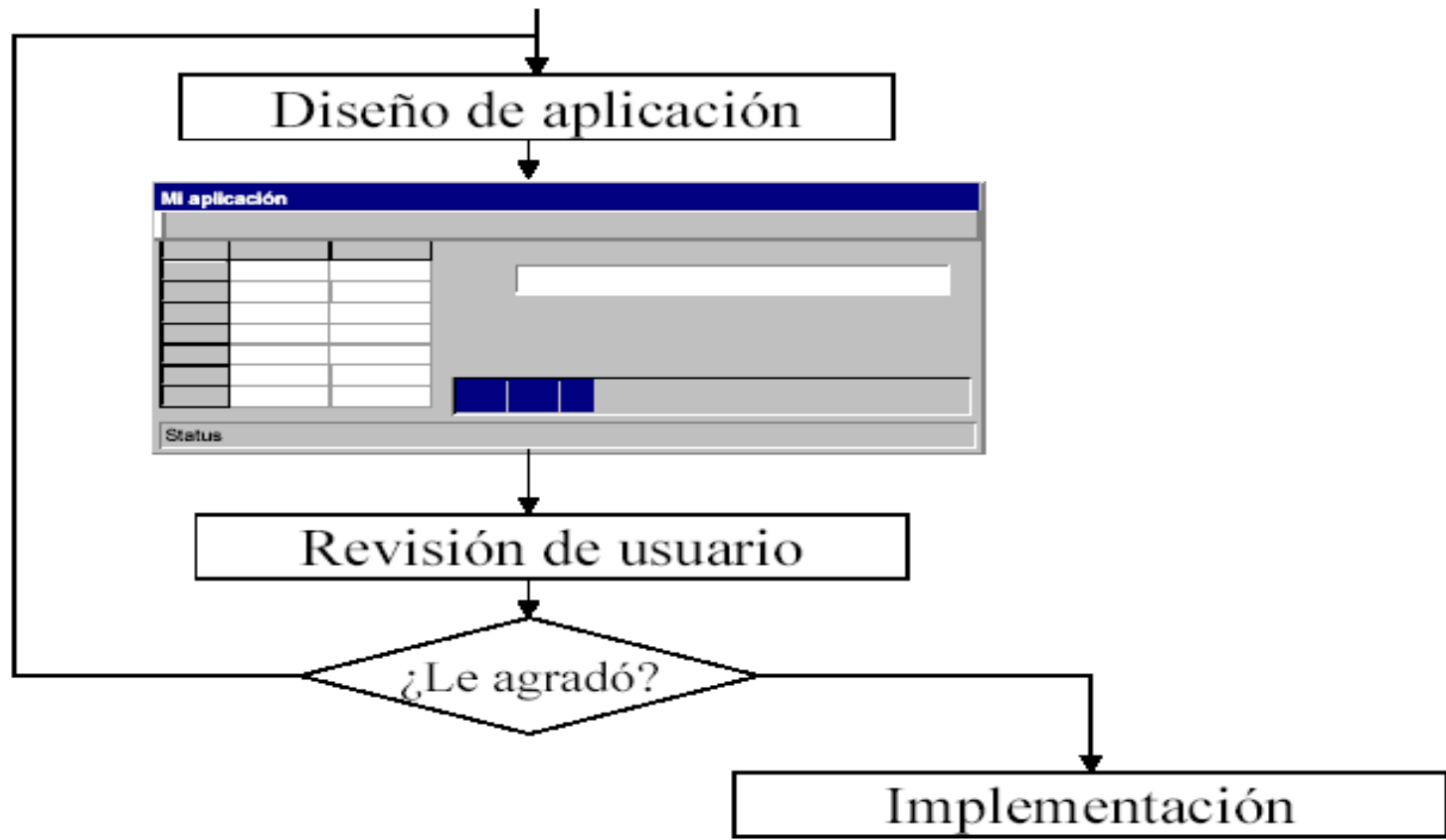
[Despliegue de componentes(...)

Cuando se esté desarrollando una GUI, es importante crear un diseño preliminar de la interface gráfica. En términos computacionales se les conoce como dummies, y son dibujos de la interface que visualizan la aplicación en su aspecto gráfico.

Si el usuario se siente satisfecho por el diseño de la aplicación, entonces se implementa el diseño sobre un lenguaje visual, en nuestro caso Java.



Despliegue de componentes(...)





[Despliegue de componentes(...)]

Java permite el diseño de aplicaciones mediante el control sobre la forma en que se acomodan los componentes en un contenedor. Tal control se le conoce como manejador de despliegue (layout manager) y es representado por medio de instancias de que implementan la interface `java.awt.LayoutManager`.

De esta forma, se le indica a un contenedor cómo debe pintar cada componente dentro de su área gráfica.



[Despliegue de componentes(...)

Para asignar un `LayoutManager` a una instancia `Container` se utilizará el método

```
ObjectoContainer.setLayout (ObjectoLayout) ;
```

Entre los diversos manejadores de despliegue se encuentran los siguientes:



[El manejador nulo

El manejador nulo (`null`) indica que el contenedor no determinará un acomodo para los componentes, por lo que el programador deberá indicar las posiciones para cada componente dentro del contenedor.

```
public class ManejadorNulo extends JFrame {  
  
    public ManejadorNulo() {  
        ...  
        getContentPane().setLayout(null);  
  
        JButton boton = new JButton("Cerrar");  
        boton.setBounds(80, 100, 300, 50);  
  
        getContentPane().add(boton);  
    }  
}
```

ManejadorNulo.java

java

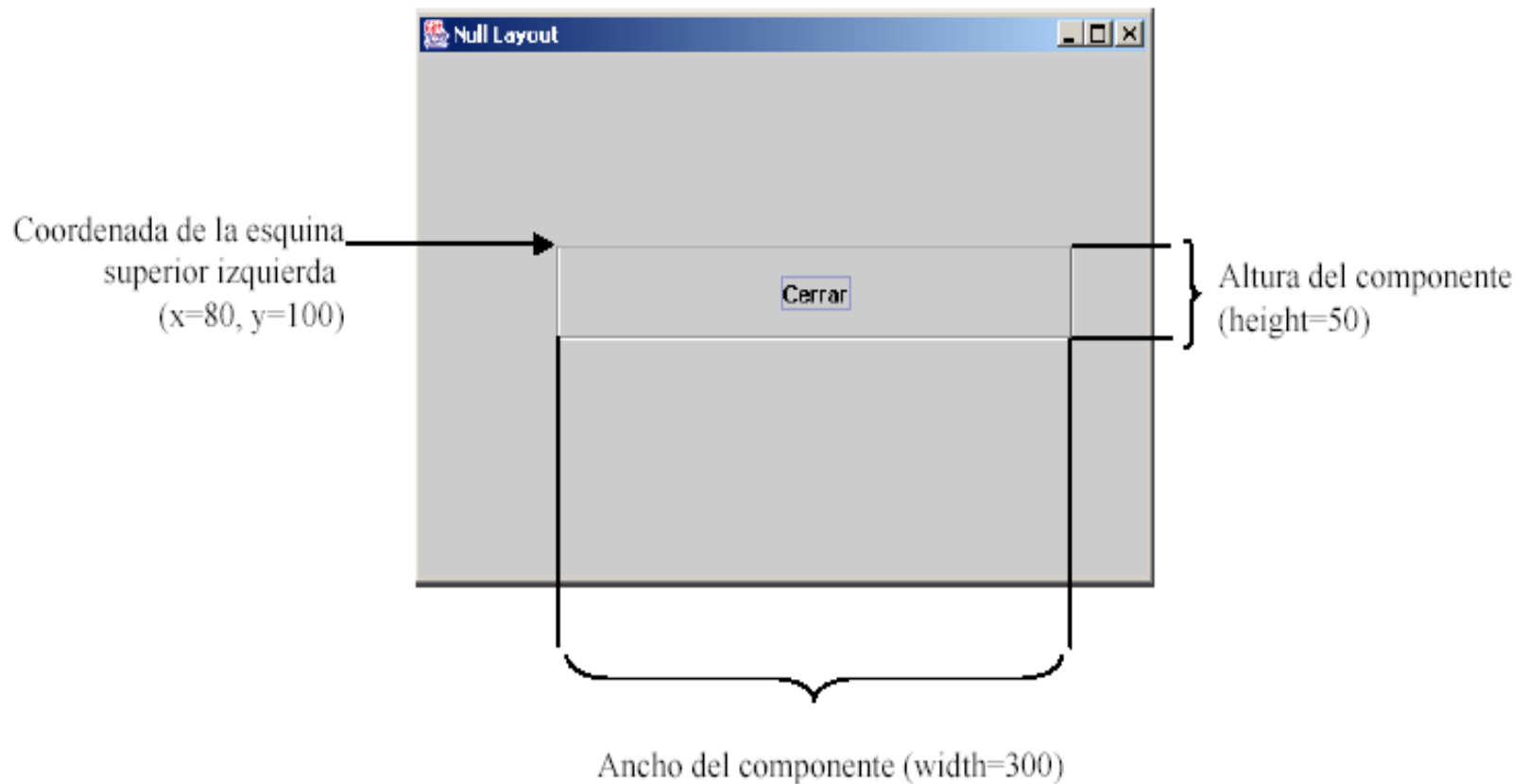
Jose Francisco Perez Reyes



[El manejador nulo(...)

1. Para agregar un componente en un JFrame deberá hacerse através de su panel interno obtenido con el método `getContentPane()`
2. Para configurar un componente deberá llamarse el método `setBounds(x, y, ancho, altura)`, donde `x` y `y` conforman la coordenada de la esquina superior izquierda del componente relativo al contenedor (en este caso el JFrame)

El manejador nulo(...)





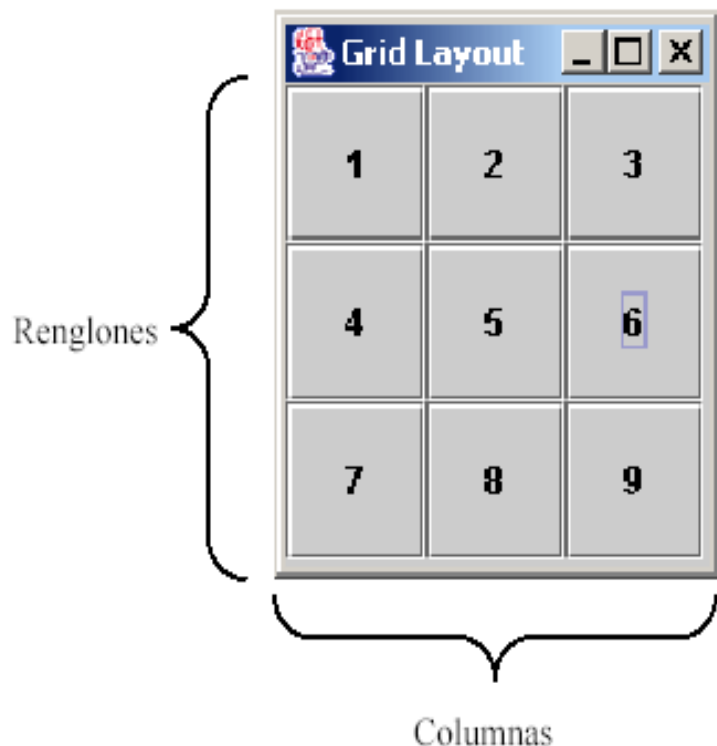
El manejador GridLayout

Este manejador, permite acomodar los componentes en una matriz (renglones \times columnas).

```
public class ManejadorGridLayout extends JFrame {  
  
    public ManejadorGridLayout() {  
        ...  
        GridLayout gl = new GridLayout(3, 3);  
        getContentPane().setLayout(gl);  
  
        for (int i=1; i<10; i++) {  
            JButton boton = new JButton("" + i);  
            getContentPane().add(boton);  
        }  
    }  
}
```

ManejadorGridLayout.java

El manejador GridLayout(...)



1. Conforme se van agregando los componentes, automáticamente se van acomodando en la estructura.
2. Si se intentara agregar un componente más, el manejador calcula la cantidad necesaria de renglones y columnas para desplegar el último componente.



El manejador FlowLayout

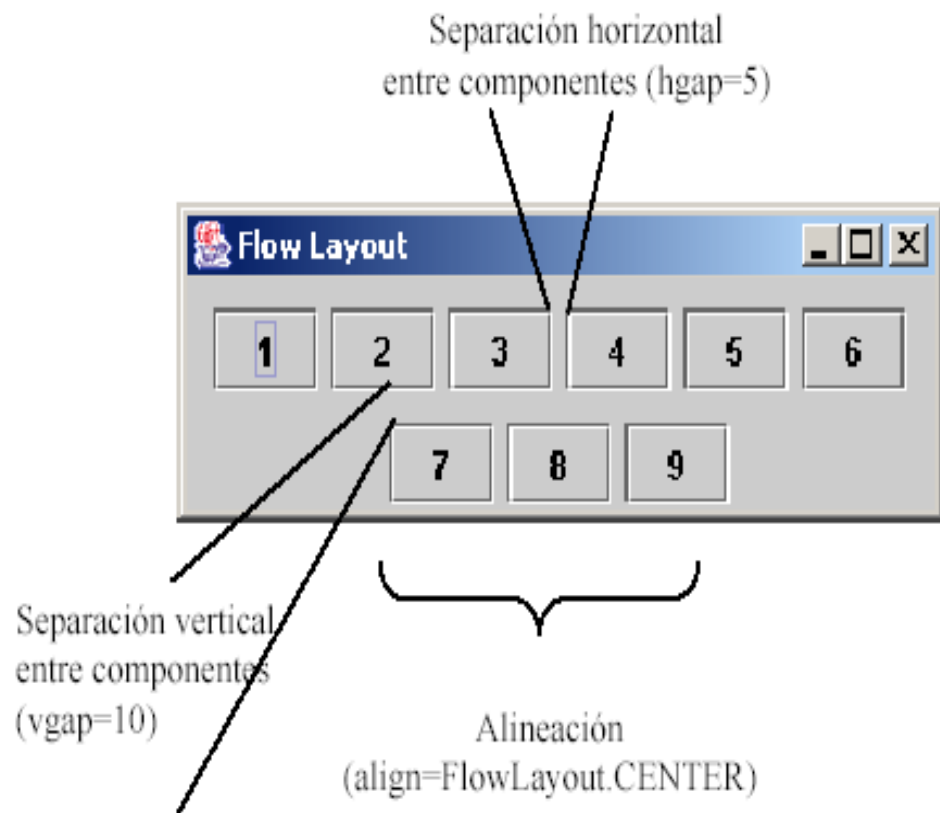
Como su nombre lo sugiere, acomoda los componentes en un flujo de izquierda a derecha, y hacia abajo, como el estilo normal de escritura en papel. Este tipo de diseño se le asocia generalmente a botones anidados en un panel.

```
...
FlowLayout fl = new FlowLayout(FlowLayout.CENTER,
5, 10);
getContentPane().setLayout(fl);

for (int i=1; i<10; i++) {
    JButton boton = new JButton("" + i);
    getContentPane().add(boton);
}
...
```

ManejadorFlowLayout.java

El manejador FlowLayout



1. Los componentes se acomodan en orden de inserción
2. Si el contenedor se redimensiona, el manejador reacomoda los componentes de acuerdo a las nuevas dimensiones.



El manejador BorderLayout

Este manejador distribuye los componentes en zonas (norte, sur, este, oeste, centro). Se asocia generalmente a aplicaciones que contengan un panel central de información.

```
...
getContentPane().setLayout(new BorderLayout());

JButton botonNorte = new JButton("Norte");
JButton botonSur = new JButton("Sur");
JButton botonEste = new JButton("Este");
JButton botonOeste = new JButton("Oeste");
JButton botonCentro = new JButton("Centro");

getContentPane().add(botonNorte, BorderLayout.NORTH);
getContentPane().add(botonSur, BorderLayout.SOUTH);
getContentPane().add(botonEste, BorderLayout.EAST);
getContentPane().add(botonOeste, BorderLayout.WEST);
getContentPane().add(botonCentro, BorderLayout.CENTER);
```

BorderLayout.java

El manejador BorderLayout

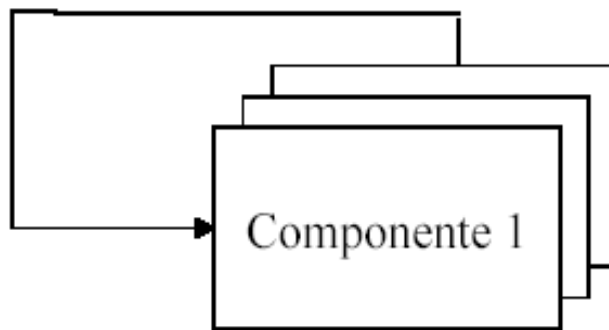


1. Cuando la ventana se redimensiona, el manejador redimensiona automáticamente los componentes en cada zona.
2. Si un componente no se agrega en una zona, la zona más próxima se apropia



El manejador CardLayout

Como su nombre lo indica, cada componente se trata como si fuera una “carta”: al quitar la “carta” actual, es decir, la visible, se observa entonces la siguiente, y así consecutivamente.



Este diseño se suele asociar con aplicaciones que tienen que mostrar formas una tras otra (ejemplo: las llegadas y salidas de un aeropuerto)



[El manejador CardLayout(...)

La ventaja de este manejador que el intercambio entre un componente a otro es rápida.

```
private CardLayout cl;  
...  
cl = new CardLayout();  
getContentPane().setLayout(cl);  
  
for (int i=1; i<=50; i++) {  
    JButton boton = new JButton("" + i);  
    getContentPane().add(boton, "" + i);  
}
```

ManejadorCardLayout.java



[El manejador CardLayout(...)

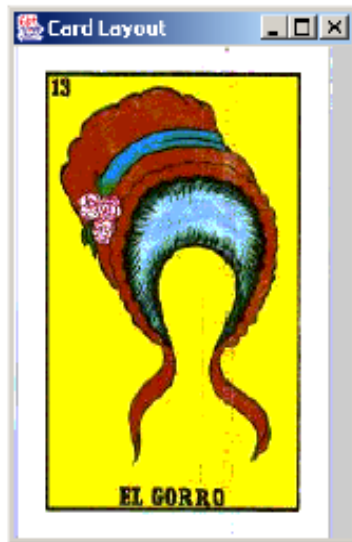
La rotación o el intercambio de un componente a otro se hace por medio de los métodos `first`, `last`, `next`, `show` de la clase `CardLayout`

```
cl.next(getContentPane());
```

Una vez indicado cuál componente mostrar, es necesario desplegarlo en el contenedor con el método `doLayout()`

```
getContentPane().doLayout();
```


El manejador CardLayout(...)



```
cl.next(getContentPane());  
getContentPane().doLayout();
```



1. Los componentes ya están cargados en memoria por lo que facilita su transición
2. Cada componente puede tener un identificador lo cual podría agilizar el cambio

```
cl.show(getContentPane(), "Muerte");
```



El manejador GridBagLayout

Sin duda, el manejador más flexible. Permite colocar los componentes en una cuadrícula gráfica organizada por celdas. Cada componente ocupará 1 o varias celdas y tendrá sus propias características (expansión, holgura, relleno, posición).

A este espacio gráfico se le conoce como *área de despliegue*.

Al utilizar este diseño, los componentes deberán agregarse asociándolos a una instancia `GridBagConstraints` que especifican al manejador los campos de configuración para el componente dado y su forma de dibujarse dentro del contenedor.



[GridBagLayout (...)

Asignación de despliegue

```
GridBagLayout gbl = new GridBagLayout();  
getContentPane().setLayout(gbl);
```

Una vez configurado el diseño de despliegue, se agregan los componentes asociándolos a una instancia `GridBagConstraints` cuyos campos indicarán la forma gráfica dentro del contenedor

`add(Component, GridBagConstraints)`



[GridBagLayout (...)

Sea `gbc` una instancia de `GridBagConstraints`.

Los campos para el objeto son los siguientes:

Posición

```
gbc.gridx = 0; //Primera columna
```

```
gbc.gridy = 2; //Tercer renglón
```

Expansión

```
//Última columna, ocupará "lo que sobre"
```

```
gbc.gridwidth = GridBagConstraints.REMAINDER;
```

```
gbc.gridheight = 2; //Dos renglones
```



[GridBagLayout (...)

Proporción relativa al contenedor

```
gbc.weightx = 1.0; //100% de ancho
```

```
gbc.weighty = 0.1; //10% de altura
```

Orientación horizontal

```
//Cargado a la izquierda
```

```
gbc.anchor = GridBagConstraints.LEFT;
```

Ocupación relativa a la celda

```
//Rellenar vertical y horizontalmente
```

```
gbc.anchor = GridBagConstraints.BOTH;
```



[GridBagLayout (...)

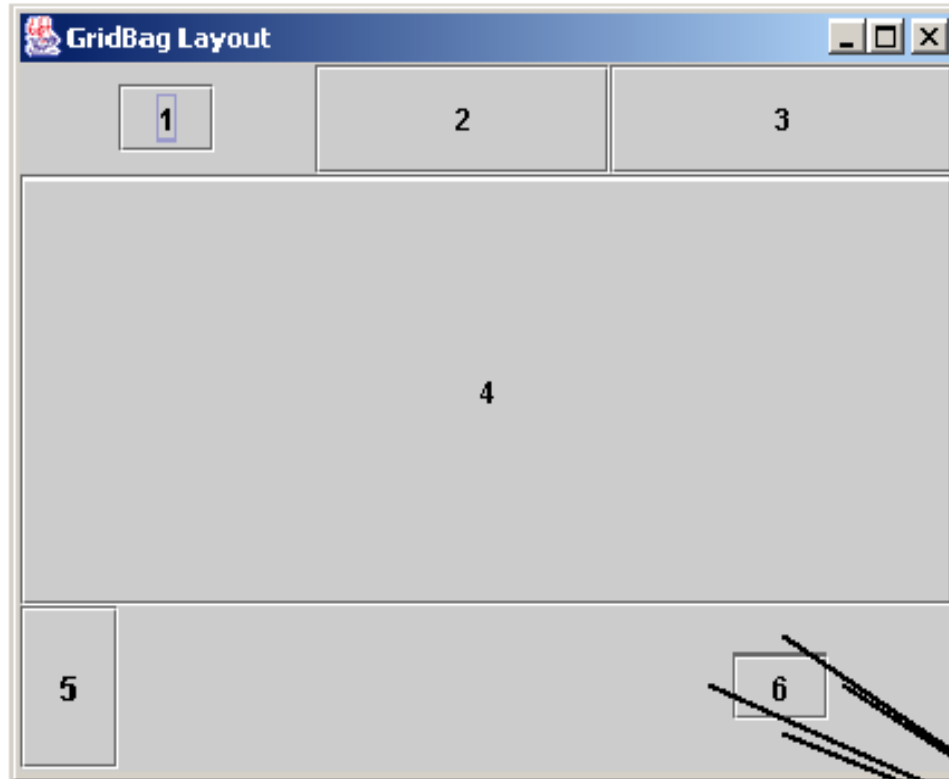
Relleno interno

```
gbc.ipadx = 1.0; //100% de ancho  
gbc.ipady = 0.1; //10% de altura
```

Relleno externo

```
//Cargado a la izquierda  
gbc.insets = GridBagConstraints.LEFT;
```

GridBagLayout (...)

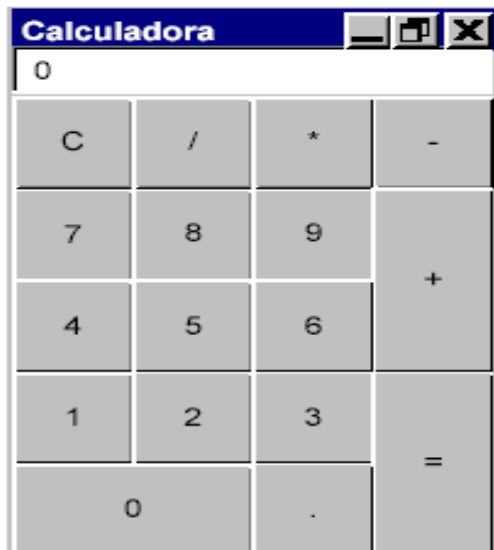


1. Los componentes se reajustan respetando su proporción, dirección y posición dentro de la pantalla cuando el contenedor cambia.

insets

Ejercicio

Dado el siguiente dummy, implemente una ventana en Java que tenga exactamente los mismos componentes respetando su diseño.



Notas:

- La ventana no es redimensionable
- El cuadro de texto tendrá el texto orientado a la derecha y será no editable
- Si se oprime el botón de salir [X], salir completamente de Java



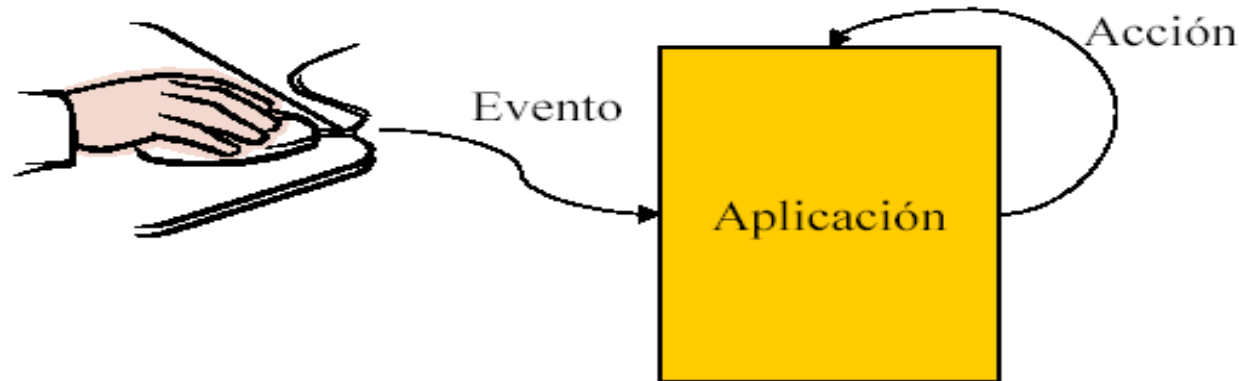
Eventos

La interacción entre usuario-aplicación se da a través de eventos. Un evento es una acción que el usuario desea efectuar sobre la aplicación. Por ejemplo, para un botón, un evento típico sería presionarlo.

Normalmente, cuando el usuario selecciona la opción de un menú, espera ver la reacción a la acción encomendada.

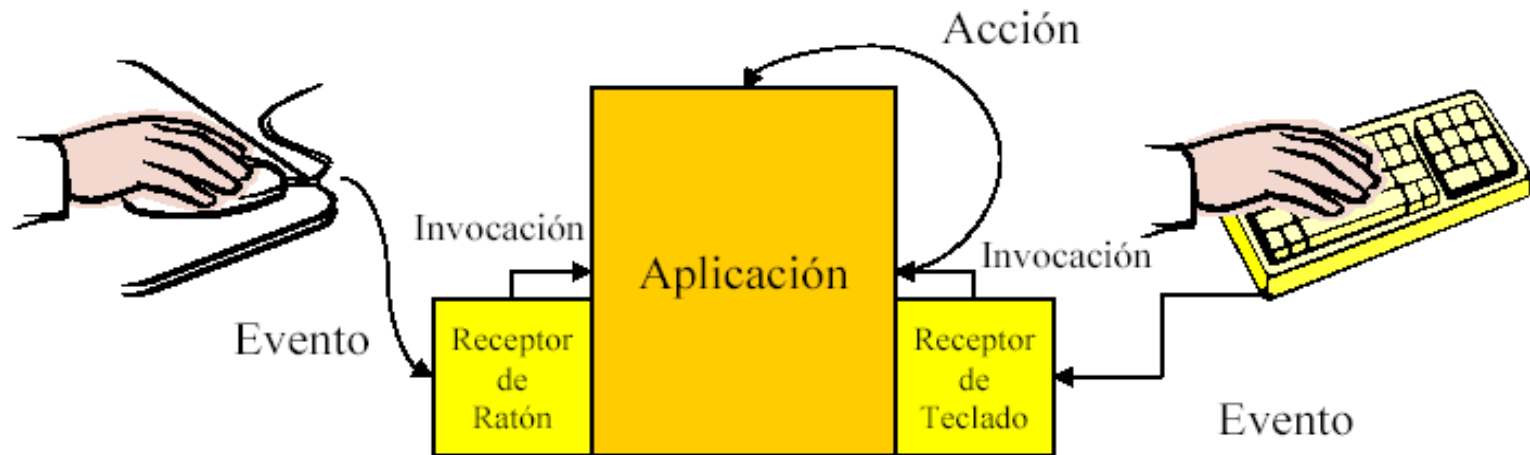
Por lo tanto, el usuario es responsable de generar los eventos sobre la aplicación. Cada componente debe atender ese evento ejecutando una acción asociada al mismo.

[Eventos(...)]



Java ofrece un mecanismo para la atención de eventos basado en *receptores*. Un **receptor** es un agente monitor que se encarga de interceptar un evento (ocasionado por teclado, ratón, etc.), y ejecutar la acción correspondiente. Se entiende por acción como el código reactivo a ese evento.

[Eventos(...)]

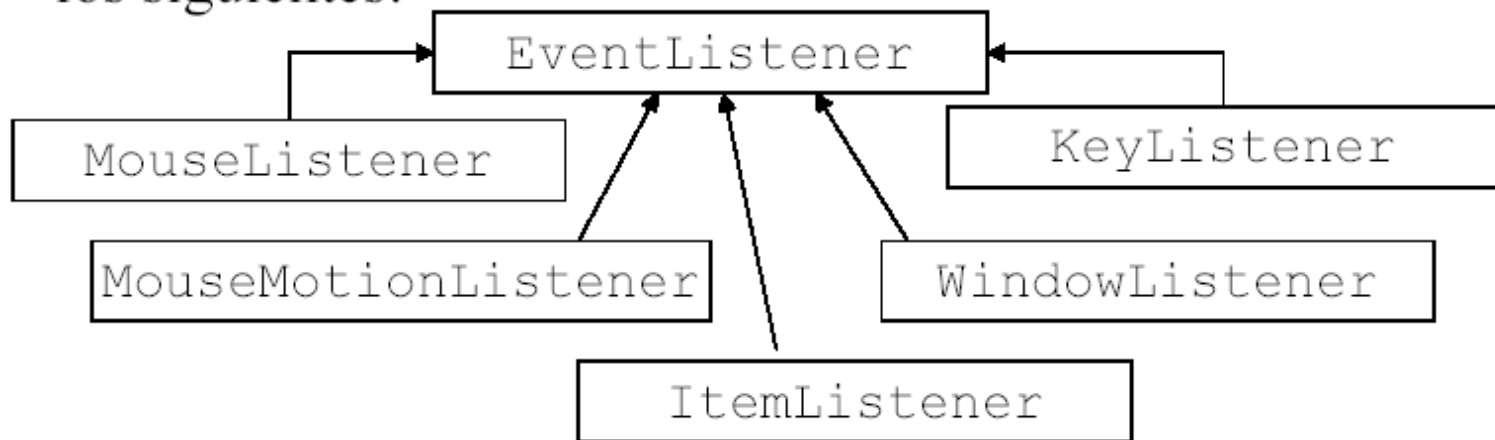


Una aplicación puede tener tantos receptores sean necesarios. Por ejemplo, si la aplicación requiere atender eventos de teclado y ratón entonces deberían asignarse los respectivos receptores a la aplicación.



[Eventos(...)]

Un receptor en Java es un objeto de una clase que implementa una subinterface de `java.util.EventListener`. Esta interface no tiene métodos, sin embargo, las subinterfaces añaden métodos particulares de los eventos que controlan. Entre los receptores más populares están los siguientes:



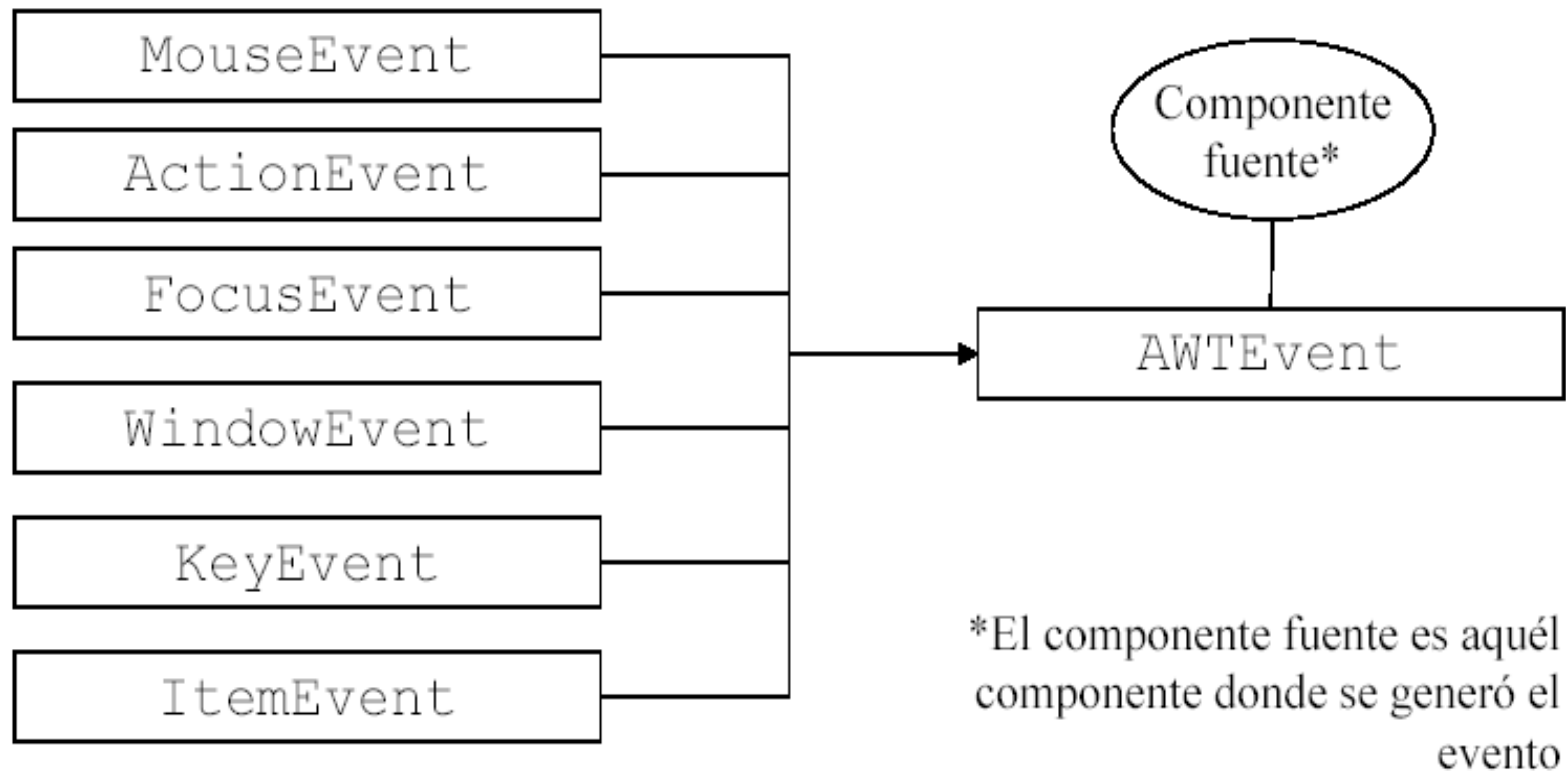


[Eventos(...)]

Un evento es un objeto de una subclase de `java.util.EventObject`. Cuando un evento ocurre, Java crea una instancia de alguna subclase de `EventObject` según sea el tipo de evento (ratón, teclado, etc.) y se la envía al receptor de eventos. El receptor de eventos ejecuta entonces la acción pertinente a ese evento.

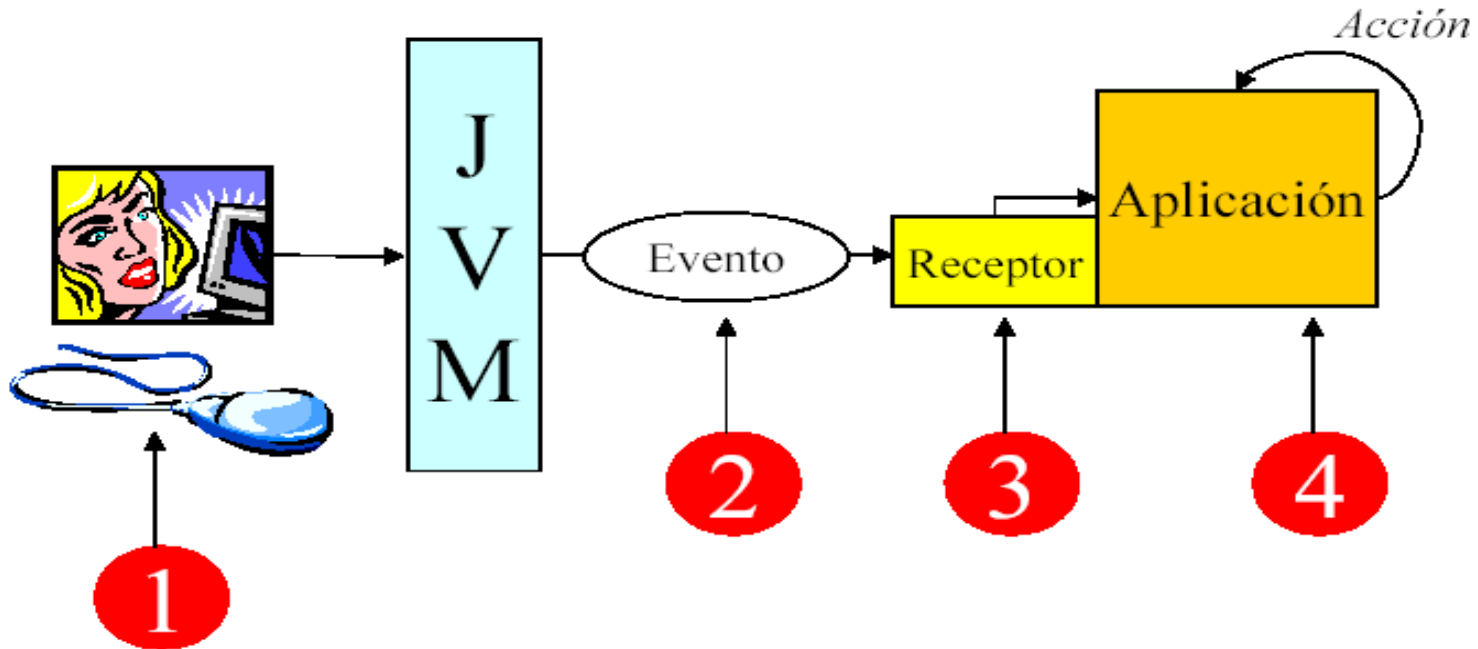
Particularmente, para AWT y Swing, se usarán subclases de `java.awt.AWTEvent` que es subclase de `EventObject`.

[Eventos(...)]



[Eventos(...)]

Existe entonces una relación entre el receptor, la máquina virtual y el evento.





[ActionListener

La interface `ActionListener` permite cualquier tipo de acción sobre un componente. Los eventos generados son instancias de `ActionEvent` que pueden tener asociados una cadena identificadora del tipo de acción.

| <i>Método</i> | <i>Acción</i> |
|---|--------------------------|
| <code>actionPerformed(ActionEvent)</code> | Cualquier tipo de acción |



[ActionListener(...)

Un receptor de acciones (ActionListener) es una clase que implementa la interface ActionListener.

```
import java.awt.event.*;
class ReceptorAccion implements
ActionListener {
    public void
        actionPerformed(ActionEvent e) {
        // Código reactivo
    }
}
```



[ActionListener(...)

Una vez definida la clase receptora, deberá crearse una instancia de la misma y asociarse al componente del cual se quieren atender los eventos. En la mayoría de los componentes se extiende el método `addActionListener(ActionListener)`

```
Componente comp;
```

```
...
```

```
comp.addActionListener(new ReceptorAccion());
```



ActionListener(...)

De esta forma, cuando ocurra una acción sobre el componente, la instancia receptora ejecutará el código reactivo definido.

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

public class DondeQuedoLaBolita extends JFrame implements
    ActionListener {

    //Botones
    private JButton boton1;
    private JButton boton2;
    private JButton boton3;
    //Elementos del juego
    private Random azar;
    private int pos;
    private int juegosGanados;
```

DondeQuedoLaBolita.java



ActionListener(...)

```
public DondeQuedoLaBolita() {  
    super();  
    ...  
    //Crear las opciones  
    boton1 = new JButton("1");  
    boton1.setActionCommand("1");  
    boton1.addActionListener(this); //Asociar el receptor con el botón  
    boton2 = new JButton("2");  
    boton2.setActionCommand("2");  
    boton2.addActionListener(this);  
    boton3 = new JButton("3");  
    boton3.setActionCommand("3");  
    boton3.addActionListener(this);  
    ...  
    azar = new Random();  
    generarNum();  
    juegosGanados = 0;  
}
```



ActionListener(...)

```
public void actionPerformed(ActionEvent e) {  
  
    String comando = e.getActionCommand();  
  
    int seleccion = Integer.parseInt(comando);  
    if (seleccion == pos) {  
        JOptionPane.showMessageDialog(this, "Bien!! Atinaste!", "Felicidades!",  
                                     JOptionPane.INFORMATION_MESSAGE);  
  
        juegosGanados++;  
        setTitle(juegosGanados + " juegos ganados");  
    } else  
        JOptionPane.showMessageDialog(this, "Sigue intentando", "Lo sentimos",  
                                     JOptionPane.ERROR_MESSAGE);  
  
    generarNum();  
}  
  
private void generarNum() {  
    pos = azar.nextInt(3) + 1;  
}
```

Eventos de ratón

La interface `MouseListener` permite manejar cualquier tipo relacionado con el ratón. Los eventos generados son instancias de `MouseEvent` que tiene asociado la coordenada (x, y) dentro del componente donde ocurrió el evento.

| <i>Método</i> | <i>Acción</i> |
|--|---|
| <code>mouseClicked(MouseEvent)</code> | El usuario hizo click en el ratón (presionar y soltar inmediatamente) |
| <code>mouseEntered(MouseEvent)</code> | El cursor del ratón se posa por el área gráfica del componente |
| <code>mouseExited(MouseEvent)</code> | El cursor del ratón sale del área gráfica del componente |
| <code>mousePressed(MouseEvent)</code> | El usuario presiona el botón del ratón |
| <code>mouseReleased(MouseEvent)</code> | El usuario suelta el botón del ratón |



[Eventos de ratón(...)

La interface `MouseMotionListener` permite manejar eventos del ratón relacionados con “drag&drop”, esto es, los eventos que tienen que ver con movimiento dentro de un componente.

| <i>Método</i> | <i>Acción</i> |
|---|---|
| <code>mouseMoved (MouseEvent)</code> | El cursor del ratón se movió internamente por el componente sin tener presionado ningún botón |
| <code>mouseDragged (MouseEvent)</code> | El usuario presionó el ratón y sin soltarlo lo mueve internamente por el componente |





[Un pequeño editor de figuras]

El editor gráfico es un buen ejemplo para clarificar el uso de los eventos del ratón. Básicamente el usuario selecciona un tipo de figura (puntos, línea, rectángulo, círculo). Cuando el usuario presiona el ratón (`mousePressed`) comienza la operación de dibujar la figura seleccionada. Mientras el usuario mantenga presionado el ratón (`mouseDragged`) debemos ejecutar el mecanismo de doble buffereo para mantener un respaldo de la última pantalla. Hasta que el usuario suelte el ratón (`mouseReleased`) actualizamos el buffer para la siguiente operación.



[Un pequeño editor(...)

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

public class MiniPaint extends JFrame implements ActionListener,
                                                MouseListener,
                                                MouseMotionListener {

    private ButtonGroup modos;
    private JPanel area;
    private JLabel status;
    private Image buffer;
    private Image temporal;

    private final int PUNTOS      = 1;
    private final int LINEAS      = 2;
    private final int RECTANGULOS = 3;
    private final int CIRCULOS    = 4;
    private int modo;
    private int x, y;
```

MiniPaint.java



[Un pequeño editor(...)

```
public MiniPaint() {
    super("MiniPaint 1.0");

    JMenuBar menuBar = new JMenuBar();
    //Menu Archivo
    JMenu menuArchivo = new JMenu("Archivo");

    //Opcion nuevo
    JMenuItem opcionNuevo = new JMenuItem("Nuevo", 'N');
    opcionNuevo.addActionListener(this);
    opcionNuevo.setActionCommand("Nuevo");
    menuArchivo.add(opcionNuevo);

    menuArchivo.addSeparator();
    //Opcion Salir
    JMenuItem opcionSalir = new JMenuItem("Salir", 'S');
    opcionSalir.addActionListener(this);
    opcionSalir.setActionCommand("Salir");
    menuArchivo.add(opcionSalir);

    menuBar.add(menuArchivo);
}
```



[Un pequeño editor(...)

```
modos = new ButtonGroup();
//Menu Modo
JMenu menuModo = new JMenu("Modo");
    //Opcion Puntos
JRadioButtonMenuItem opcionPuntos = new
    JRadioButtonMenuItem("Puntos", true);
opcionPuntos.addActionListener(this);
opcionPuntos.setActionCommand("Puntos");
menuModo.add(opcionPuntos);
modos.add(opcionPuntos);
    //Opcion Lineas
JRadioButtonMenuItem opcionLineas = new
    JRadioButtonMenuItem("Líneas");
opcionLineas.addActionListener(this);
opcionLineas.setActionCommand("Lineas");
menuModo.add(opcionLineas);
modos.add(opcionLineas);
```



Un pequeño editor(...)

```
//Opcion Rectangulos
JRadioButtonMenuItem opcionRectangulos = new
JRadioButtonMenuItem("Rectángulos");
opcionRectangulos.addActionListener(this);
opcionRectangulos.setActionCommand("Rectangulos");
menuModo.add(opcionRectangulos);
modos.add(opcionRectangulos);

//Opcion Círculos
JRadioButtonMenuItem opcionCirculos = new
JRadioButtonMenuItem("Círculos");
opcionCirculos.addActionListener(this);
opcionCirculos.setActionCommand("Circulos");
menuModo.add(opcionCirculos);
modos.add(opcionCirculos);
menuBar.add(menuModo);
```



[Un pequeño editor(...)

```
area = new JPanel();
area.addMouseListener(this);
area.addMouseMotionListener(this);
status = new JLabel("Status", JLabel.LEFT);
//Asignar barra menues
setJMenuBar(menuBar);
//Agregar zona grafica
getContentPane().add(area, BorderLayout.CENTER);
//Agregar barra de estado
getContentPane().add(status, BorderLayout.SOUTH);
modo = PUNTOS;
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setSize(400, 300);
show();
buffer = area.createImage(area.getWidth(), area.getHeight());
}
```



Un pequeño editor(...)

```
public void actionPerformed(ActionEvent e) {
    String comando = e.getActionCommand();
    if (comando.equals("Nuevo")) {
        area.getGraphics().clearRect(0, 0, area.getWidth(),
        area.getHeight());
    } else
    if (comando.equals("Salir")) {
        if (JOptionPane.showConfirmDialog(this, "¿En verdad desea salir?",
        "Confirmación", JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
        {
            dispose();
            System.exit(0);
        }
    } else
    if (comando.equals("Puntos")) {
        modo = PUNTOS;
    } else
    if (comando.equals("Lineas")) {
        modo = LINEAS;
    }
}
```



[Un pequeño editor(...)

```
public void mouseClicked(MouseEvent e) {
}
public void mousePressed(MouseEvent e) {
    x = e.getX();
    y = e.getY();
    temporal = area.createImage(area.getWidth(), area.getHeight());
    temporal.getGraphics().drawImage(buffer, 0, 0, this);
}
public void mouseReleased(MouseEvent e) {
    buffer.getGraphics().drawImage(temporal, 0, 0, this);
}
public void mouseEntered(MouseEvent e) {
    setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
}
public void mouseExited(MouseEvent e) {
    setCursor(Cursor.getDefaultCursor());
}
```



Un pequeño editor(...)

```
public void mouseDragged(MouseEvent e) {  
    Graphics g = temporal.getGraphics();  
    switch (modo) {  
        case PUNTOS:  
            g.fillOval(e.getX(), e.getY(), 1, 1);  
            area.getGraphics().drawImage(temporal, 0, 0, this);  
            break;  
        case LINEAS:  
            g.drawImage(buffer, 0, 0, area);  
            g.drawLine(x, y, e.getX(), e.getY());  
            area.getGraphics().drawImage(temporal, 0, 0, this);  
            break;  
        case RECTANGULOS:  
            g.drawImage(buffer, 0, 0, area);  
            g.drawRect(x, y, e.getX()-x, e.getY()-y);  
            area.getGraphics().drawImage(temporal, 0, 0, this);  
            break;  
        case CIRCULOS:  
            g.drawImage(buffer, 0, 0, area);  
            g.drawOval(x, y, e.getX()-x, e.getY()-y);  
            area.getGraphics().drawImage(temporal, 0, 0, this);  
            break;  
    }  
}
```




[Un pequeño editor(...)

```
public void mouseMoved(MouseEvent e) {  
    status.setText("x=" + e.getX() + ",y=" + e.getY());  
}  
public static void main(String[] args) {  
    new MiniPaint();  
}  
}
```



Eventos de teclado

La interface `KeyListener` permite manejar cualquier tipo relacionado con el teclado. Los eventos generados son instancias de `KeyEvent` que tiene asociado el código del carácter capturado.

| <i>Método</i> | <i>Acción</i> |
|------------------------------------|---------------------------------------|
| <code>keyPressed(KeyEvent)</code> | Una tecla fue presionada |
| <code>keyReleased(KeyEvent)</code> | La tecla presionada fue liberada |
| <code>KeyTyped(KeyEvent)</code> | Una tecla alfanumérica fue presionada |



[Eventos de teclado(...)]

Asociar un componente con un receptor de eventos de teclado se consigue mediante el método `addKeyListener(KeyListener)`

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

class ReceptorTeclado implements KeyListener {
    public void keyTyped(KeyEvent e) {
        //Código reactivo
    }
    public void keyPressed(KeyEvent e) {
        //Código reactivo
    }
    public void keyReleased(KeyEvent e) {
        //Código reactivo
    }
}
```



[Eventos de teclado(...)]

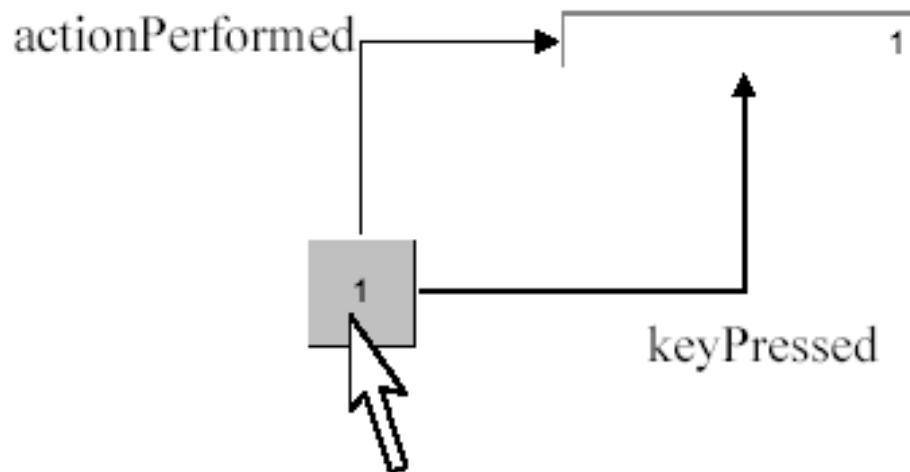
Los objetos `KeyEvent` tienen asociado el código de la tecla que fue presionada.

...

```
public void keyPressed(KeyEvent e) {  
    switch (e.getKeyCode()) {  
        case KeyEvent.VK_0:  
            System.out.println("0");  
            break;  
        case KeyEvent.VK_ESCAPE:  
            System.out.println("Salir");  
            break;  
    }  
}
```

[Ejercicio

De la calculadora del anterior ejercicio, defina las acciones para todos los componentes. Los botones al presionarse con el ratón o al escribir el dígito del botón respectivo.





Eventos de ventana

La interface `WindowListener` permite manejar los eventos relacionados como cerrar, activar, mostrar, maximizar y minimizar. Los eventos generados son instancias de `WindowEvent`.

| <i>Método</i> | <i>Acción</i> |
|---|--|
| <code>windowActivated(WindowEvent)</code> | La ventana ha sido activada, esto es, recibe el foco de atención del usuario |
| <code>windowClosed(WindowEvent)</code> | La ventana ha sido cerrada por completo |
| <code>windowClosing(WindowEvent)</code> | El usuario está intentando cerrar la ventana |
| <code>windowDeactivated(WindowEvent)</code> | La ventana ha sido desactivada, esto es, perdió el foco de atención |
| <code>windowDeiconified(WindowEvent)</code> | La ventana ha sido maximizada |
| <code>windowIconified(WindowEvent)</code> | La ventana ha sido minimizada |
| <code>windowOpened(WindowEvent)</code> | La ventana se ha desplegado en pantalla |



[Adaptadores

Hay situaciones donde el desarrollador no requiere atender algunos eventos en especial. Por ejemplo, de la interface `KeyListener` solo se desea el método `keyPressed`. Hasta ahorita, crear un receptor implica implementar todos los métodos (por regla de interfaces). Existe una alternativa: usando adaptadores.

Un adaptador es una clase que implementa una interface receptora. Si se desea crear un adaptador personalizado solo deberá crearse una subclase de ese adaptador.



[Adaptadores(...)

De tal forma que para nuestro receptor de teclado tendríamos:

```
public class MiAdaptador extends KeyAdapter {  
    public void keyPressed(KeyEvent e) {  
        //Código reactivo  
    }  
}  
...  
addKeyListener(new MiAdaptador());
```




[Adaptadores(...)

Ventajas:

- Es más fácil la delegación de la responsabilidad de atención de eventos
- La carga se reparte aligerando el diseño

Desventajas

- La comunicación entre receptor y componente por lo general se hace haciendo el receptor como una clase interna del componente.
- Una ventana por ejemplo, no podría responsabilizarse de sus eventos debido a la herencia singular.



[El MiniEncuestador

Se trata de desarrollar un pequeño encuestador para una sola pregunta. La aplicación ofrecerá la consulta estadística de los resultados hasta entonces capturados. Consulte el paquete “me” dentro del proyecto Sesión8.

¿Cómo implementa automáticamente JBuilder los receptores de eventos?