



Java UI – Manejo de eventos

- En Java los eventos son representados por objetos
- Ejemplos:
 - ◆ clic en un botón
 - ◆ arrastrar el mouse
 - ◆ pulsar Enter
- Los componentes AWT y Swing generan (*fire*) eventos
- `java.awt.AWTEvent`



Los XXXEvent nos informan...

- Quién lo dispara?
- De qué tipo es?
- Cuándo ocurrió?
- Información propia del evento

Los detalles del evento pueden ser obtenidos usando métodos de acceso:

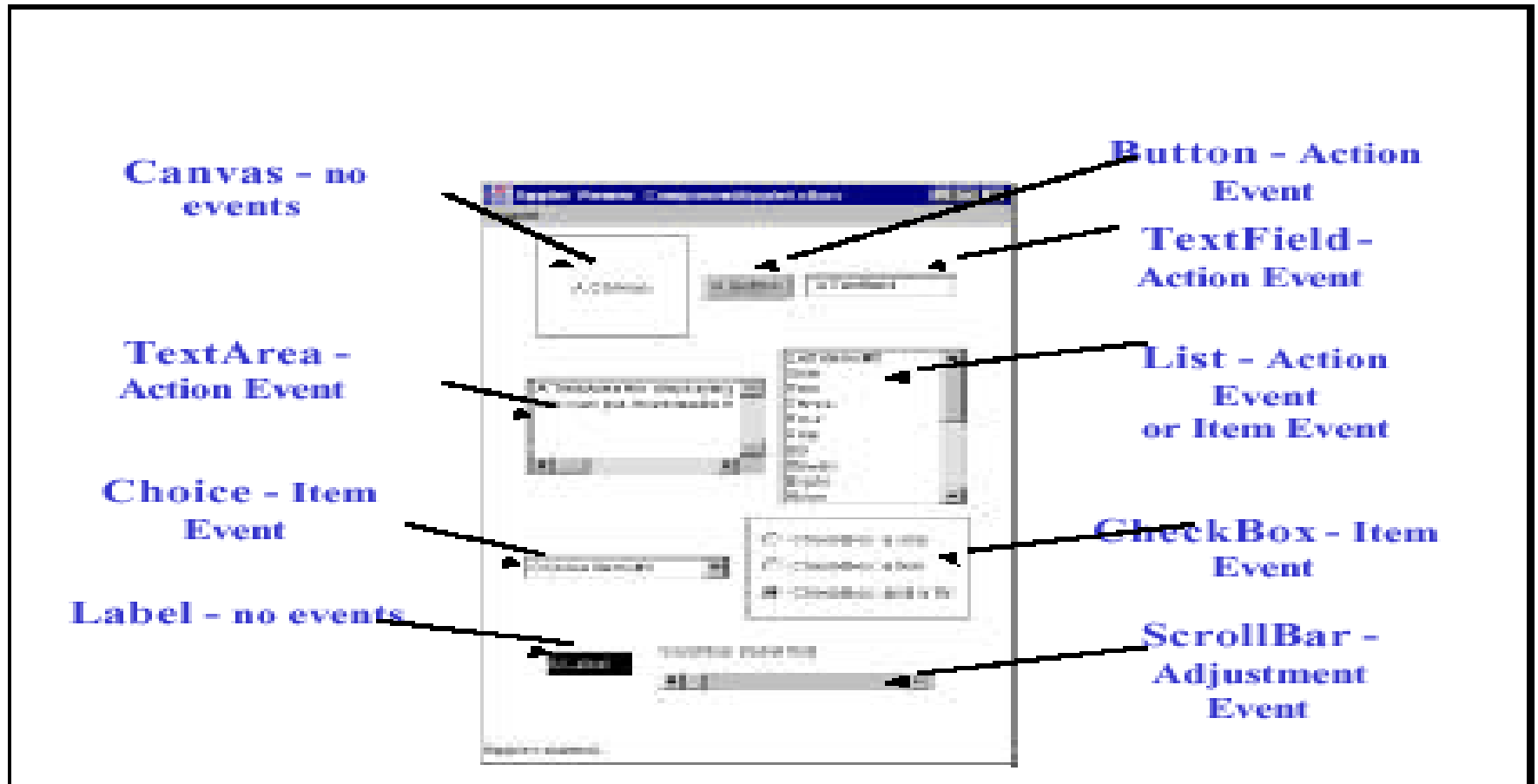
Ej.: `getActionCommand()`
`getModifiers()`



Eventos AWT

- De bajo nivel
 - ◆ Componentes – `ComponentEvent`, `FocusEvent`, `KeyEvent`, `MouseEvent`, `PaintEvent`
 - ◆ Contenedores – `ContainerEvent`
 - ◆ Ventanas – `WindowEvent`
- Semánticos – mayor nivel de abstracción
 - ◆ `ActionEvent`, `ItemEvent`, `TextEvent`, `AdjustmentEvent`

Eventos AWT





Eventos semánticos

- No son disparados por todos los componentes
- Ejemplo 1: ItemEvent indica que un ítem fue seleccionado o deseleccionado
 - ◆ Disparado por JComboBox
 - ◆ No disparado por JButton
- Ejemplo 2: ActionEvent
 - ◆ Disparado por JComboBox
 - ◆ Disparado por JButton



Eventos Swing

- Swing tiene además su propio paquete de manejo de eventos: `javax.swing.event`
- Casi todos estos nuevos eventos están relacionados con la arquitectura MVC
- Ejemplo:
 - ◆ `TreeModelEvent`
 - ◆ `ListDataEvent`

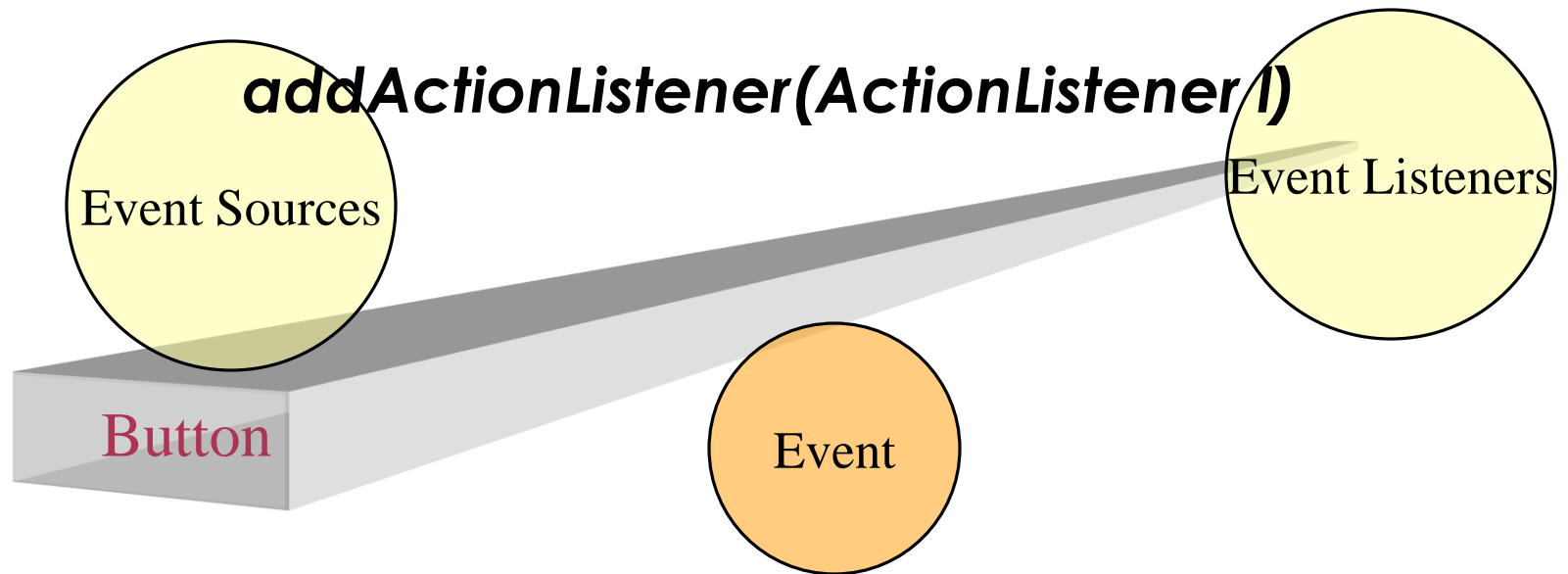


Manejo de eventos

- Modelo de Delegación de Eventos
 - ◆ Interfaces “listeners”
 - ◆ Registro para recibir eventos de una fuente
 - ◆ Patrón Observer



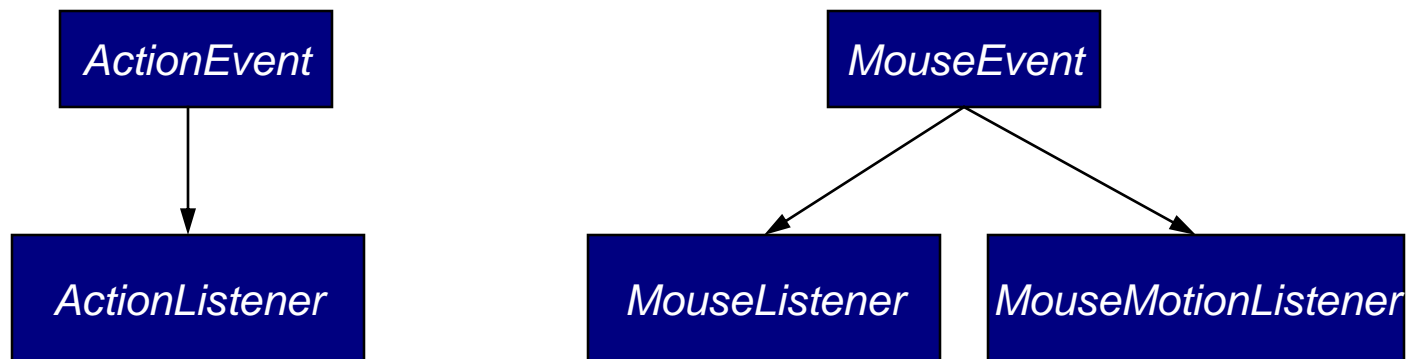
Manejo de eventos





Listeners

- Interfaces que manejan los eventos
(`java.util.EventListener`). Basadas en Observer
- Cada clase Event tiene su correspondiente interface Listener
- Varios Listeners para el mismo tipo de eventos





Listeners

Ejemplos de Listeners:

```
public interface ActionListener {  
    public void actionPerformed(ActionEvent e);  
}
```

```
public interface ItemListener {  
    public void itemStateChanged(ItemEvent e);  
}
```

```
public interface ComponentListener {  
    public void componentHidden(ComponentEvent e);  
    public void componentMoved(ComponentEvent e);  
    public void componentResized(ComponentEvent e);  
    public void componentShown(ComponentEvent e);  
}
```

```
public interface WindowListener extends
EventListener {
void windowActivated(WindowEvent e);
void windowClosed(WindowEvent e);
void windowClosing(WindowEvent e);
void windowDeactivated(WindowEvent e);
void windowDeiconified(WindowEvent e);
void windowIconified(WindowEvent e);
void windowOpened(WindowEvent e);}

public interface MouseListener extends
EventListener {
public void mouseClicked(MouseEvent e);
public void mousePressed(MouseEvent e);
public void mouseReleased(MouseEvent e);
public void mouseEntered(MouseEvent e);
public void mouseExited(MouseEvent e);}
```

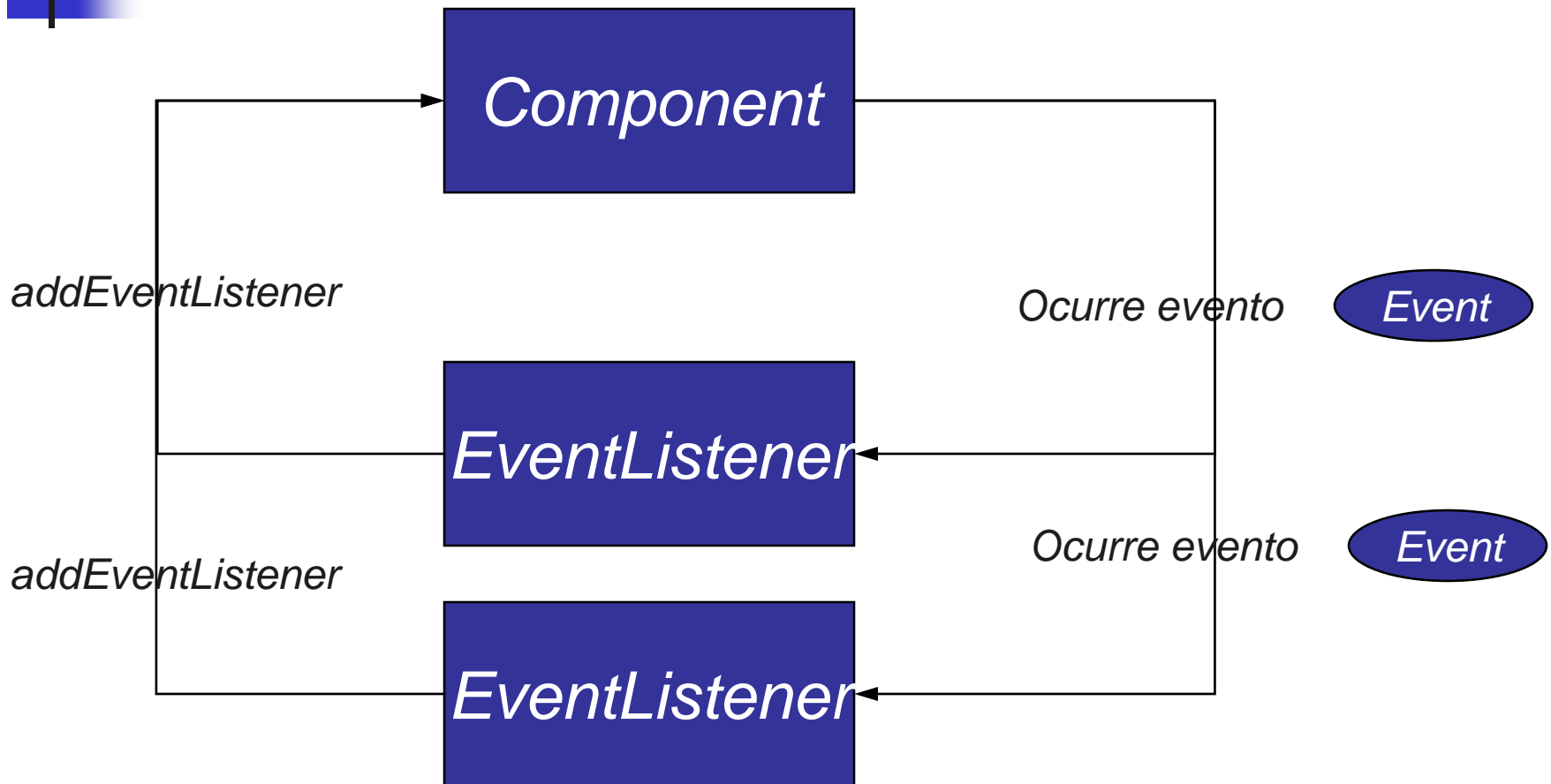


Registro de Listeners

- Un Listener debe en primer lugar registrarse con la(s) fuente(s) que pueda(n) generar los eventos de su interés:
 - ♦ `public void addXXXListener(XXXListener e)`
 - ♦ `addActionListener`, `addItemListener`, etc.
- Para el mismo evento en un componente, puede haber registrado varios Listeners
 - ♦ Un evento puede provocar numerosas respuestas
 - ♦ Los eventos son “difundidos” a todos sus Listeners



Múltiples Listeners





Conectar un Listener con una fuente de eventos

- Defina una clase que implemente la interface Listener(o que extienda una clase que la implemente)

```
public class AppFrame extends JFrame implements ActionListener {...
```

- Añada la implementación de la interface

```
...  
public void actionPerformed(ActionEvent e) {  
    // here's where I do stuff when the action happens  
...
```

- Registre el Listener con la fuente

```
...  
Jbutton okButton = new JButton("OK");  
okButton.addActionListener(this);  
...
```



Tips

- Debe implementar todos los métodos de la interface
Si el código usado para implementar el manejo de eventos tiene unas pocas líneas se suele usar una clase interna anónima.
- No hay garantía de cuál Listener es notificado primero.
No escribir código contando con un orden específico.
- Trate eventos semánticos antes que de bajo nivel
 - ◆ cambios en look and feel
 - ◆ componentes compuestos
- Utilice métodos descriptivos de los eventos
 - ◆ `ActionEvent` – `getActionCommand()`
- Threads



Clases Adapter

- Son clases que implementan una interface Listener con métodos vacíos (“dummy”), uso herencia.
Desventaja: Java no permite herencia múltiple
Solución: usar clases internas anónimas
- Útiles sólo para interfaces Listeners con más de un método
- Principalmente por razones de conveniencia
- Ejemplos:
 - ◆ MouseAdapter
 - ◆ WindowAdapter



Clases internas

- En Java una clase puede ser definida en cualquier lugar
 - ◆ Anidada dentro de otras clases
 - ◆ En la invocación de un método
- Tienen acceso a los miembros y métodos de todas las clases externas a ellas
- Pueden tener nombres o ser anónimas
- Pueden extender de otra clase o implementar interfaces
- Muy útiles para el manejo de eventos



Clases internas con nombre

- Se definen como las clases normales
- No pueden ser ***public***

```
public class ApplicationFrame {  
    ....  
    class ButtonHandler implements ActionListener {  
        Public void actionPerformed(ActionEvent e) {  
            doTheOKThing();  
        }  
    }  
  
    private void doTheOKThing() { // here's where I handle OK  
    }  
    ....  
    JButton okButton = new JButton("OK");  
    okButton.addActionListener(new ButtonHandler()); // create inner class listener  
    ....  
}
```



Clases internas con nombre

```
public class MyClass extends JPanel {  
    ...  
    anObject.addMouseListener(new MyAdapter());  
    ...  
    class myAdapter extends MouseAdapter {  
        public void mouseClicked(MouseEvent e) {  
            // blah  
        } // end mouseClicked  
    } // end inner class  
} // end MyClass
```



Clases internas anónimas

- Definida dentro de addXXXListener

```
(new className( ) { classBody });  
(new interfaceName() { classBody });
```

Dentro del cuerpo no puedo definir constructores.

```
Public class ApplicationFrame {
```

```
....
```

```
    JButton okButton = new JButton("OK");  
    okButton.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent event) {  
            doTheOKThing();  
        }  
    } );
```

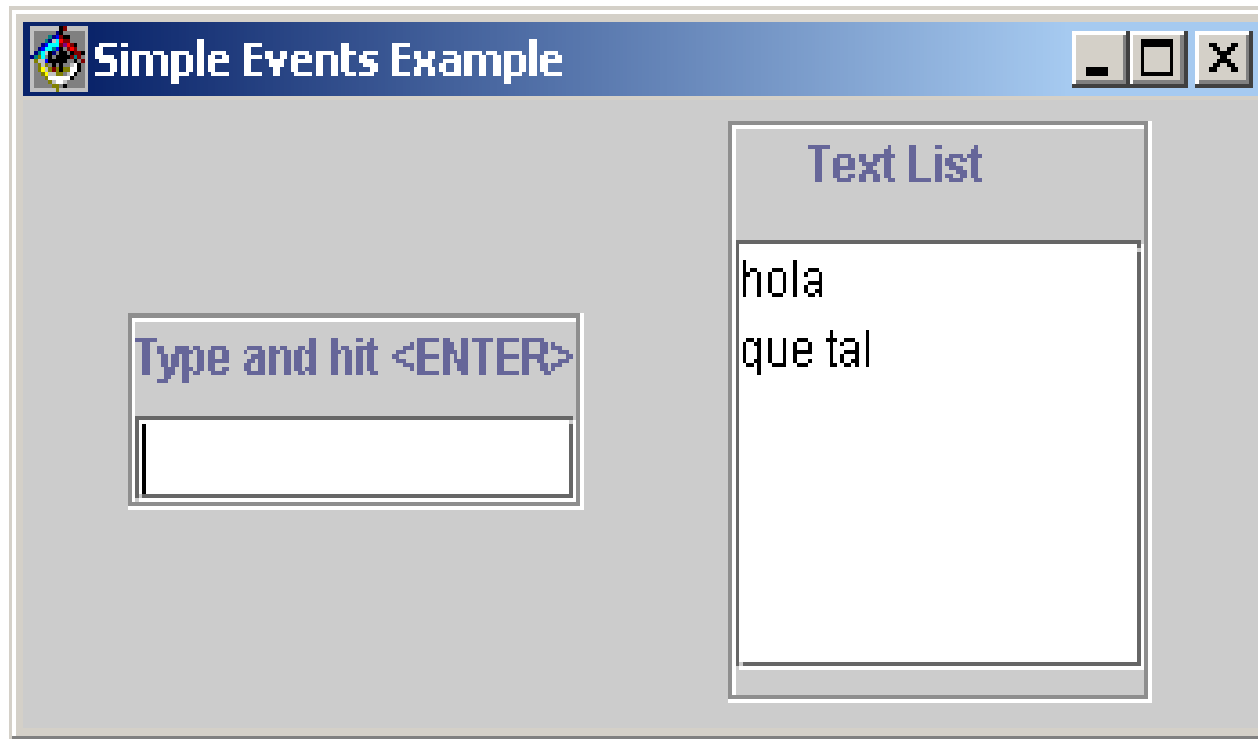
```
....
```

```
    private void doTheOKThing() { // here's where I handle the OK  
    }
```

```
....
```



Ejemplo





Código

```
// importa los símbolos de AWT and Swing
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SimpleEvents extends JFrame {
    static final int WIDTH=350; //ancho y alto del frame
    static final int HEIGHT=180;
    // Declara JTextField para entrar texto
    JTextField textField;
    // Declara JTextArea p/recibir líneas de textField
    JTextArea textList;
    // Declara JScrollPane para JTextArea
    JScrollPane pane;
    // Constructor:aquí se hace casi todo el trabajo
    public SimpleEvents(String lab) {
        // llama el constructor de JFrame:pone etiqueta
        super(lab);
```



Código

```
/****** Crea un contedor para textField *****/  
// Instancia un JPanel  
JPanel textPanel = new JPanel();  
// le pone un borde (por defecto no tiene)  
textPanel.setBorder(BorderFactory.createEtchedBorder());  
// Set el layout del textPanel a BorderLayout  
textPanel.setLayout(new BorderLayout());  
// Crea una etiqueta y la añade al panel  
JLabel textTitle = new JLabel("Type and hit <ENTER>");  
textPanel.add(textTitle, BorderLayout.NORTH);  
// Instancia un JTextField y añade a textPanel  
textField = new JTextField();  
textPanel.add(textField, BorderLayout.SOUTH);  
// Añade un strut al textPanel como margen inferior  
textPanel.add(Box.createVerticalStrut(6));  
// Crea un contenedor p/ textArea  
// Instancia un JPanel  
JPanel listPanel = new JPanel();
```



Código

```
// añade borde
listPanel.setBorder (BorderFactory.createEtchedBorder());
// Set el layout del textPanel
listPanel.setLayout(new BorderLayout(listPanel,BorderLayout.Y_AXIS));
// Crea una etiqueta y añade al panel
JLabel title = new JLabel("Text List");
listPanel.add(title);
// Añade un strut al BorderLayout
listPanel.add(Box.createVerticalStrut(10));
// Instancia una JTextArea sin texto inicial
// 6 filas, 10 columnas, y vertical scrollbars
textList=new JTextArea("", 6, 10);
// la hace read-only
textList.setEditable(false);
// Añade textList a listPanel
pane = new JScrollPane (textList);
listPanel.add(pane);
```




Código

```
// Añade un strut a listPanel como margen inferior
listPanel.add(Box.createVerticalStrut(6));
/* Añade un listener a textField cuando se pulse ENTER copia el texto de
textField al area de texto. Los componentes están interrelacionados*/
textField.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
// Añade el texto de textField a textList
textList.append(textField.getText());
textList.append("\n");
// Reset el textField
textField.setText("");
}});
// Añade los 2 paneles al frame, separados por strut
Container c = getContentPane();
c.setLayout (new FlowLayout());
c.add(textPanel);
c.add(Box.createHorizontalStrut(30));
c.add(listPanel);}
```



Código

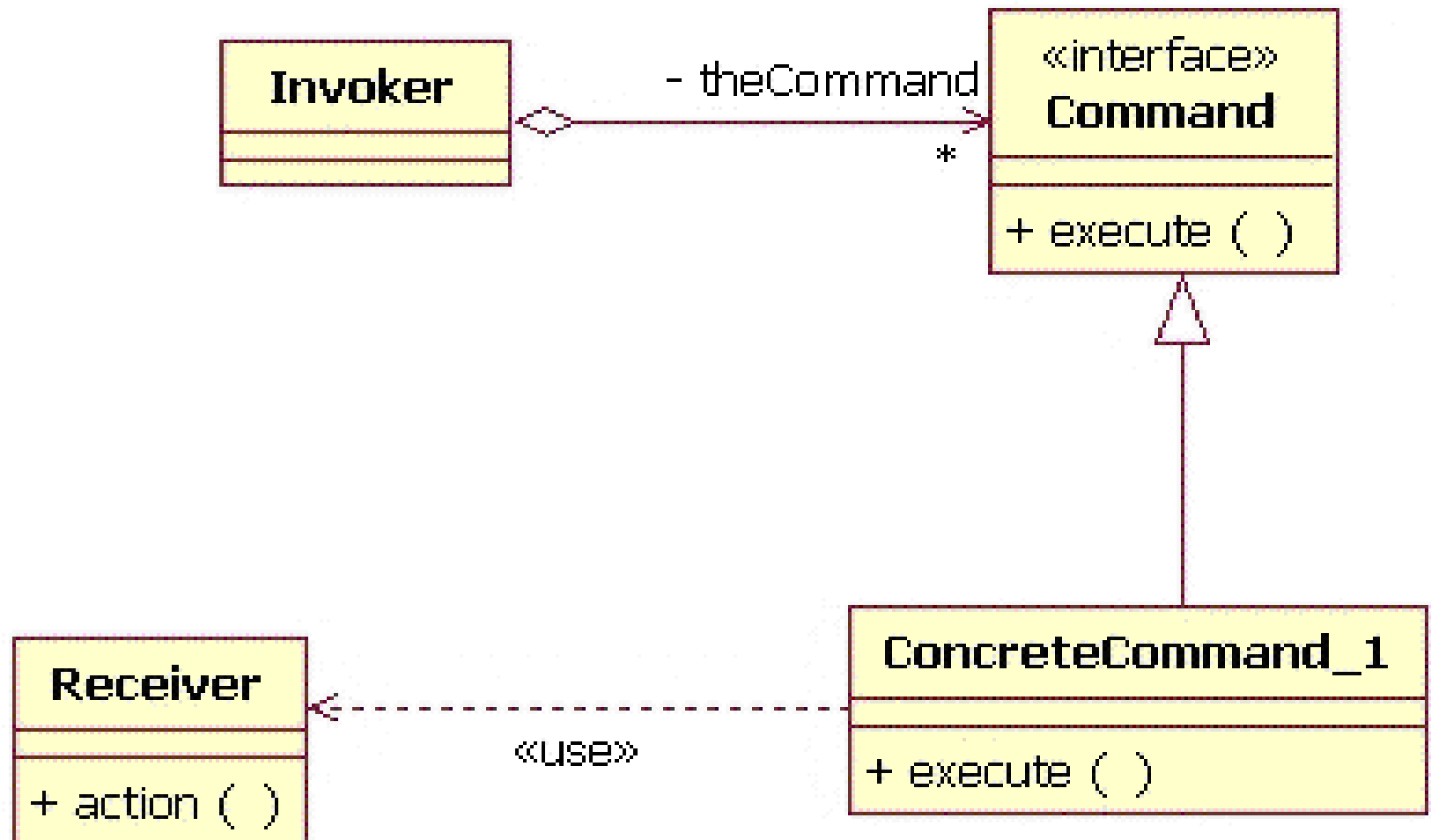
```
public static void main(String args[]) {  
    SimpleEvents frame =  
    new SimpleEvents("Simple Events Example");  
    // standard adapter usado en casi todas las  
    // aplicaciones para cerrar la ventana  
    frame.addWindowListener(new WindowAdapter() {  
        public void windowClosing(WindowEvent e) {  
            System.exit(0);  
        }  
    });  
    // set el tamaño de frame y lo muestra  
    frame.setSize(WIDTH, HEIGHT);  
    frame.setVisible(true);  
}
```



Command Pattern

- Command encapsula un requerimiento en un objeto, permitiendo parametrizar clientes con diferentes requerimientos, hacer una cola o registro de ellos y permitir por lo tanto hacer undo/redo.

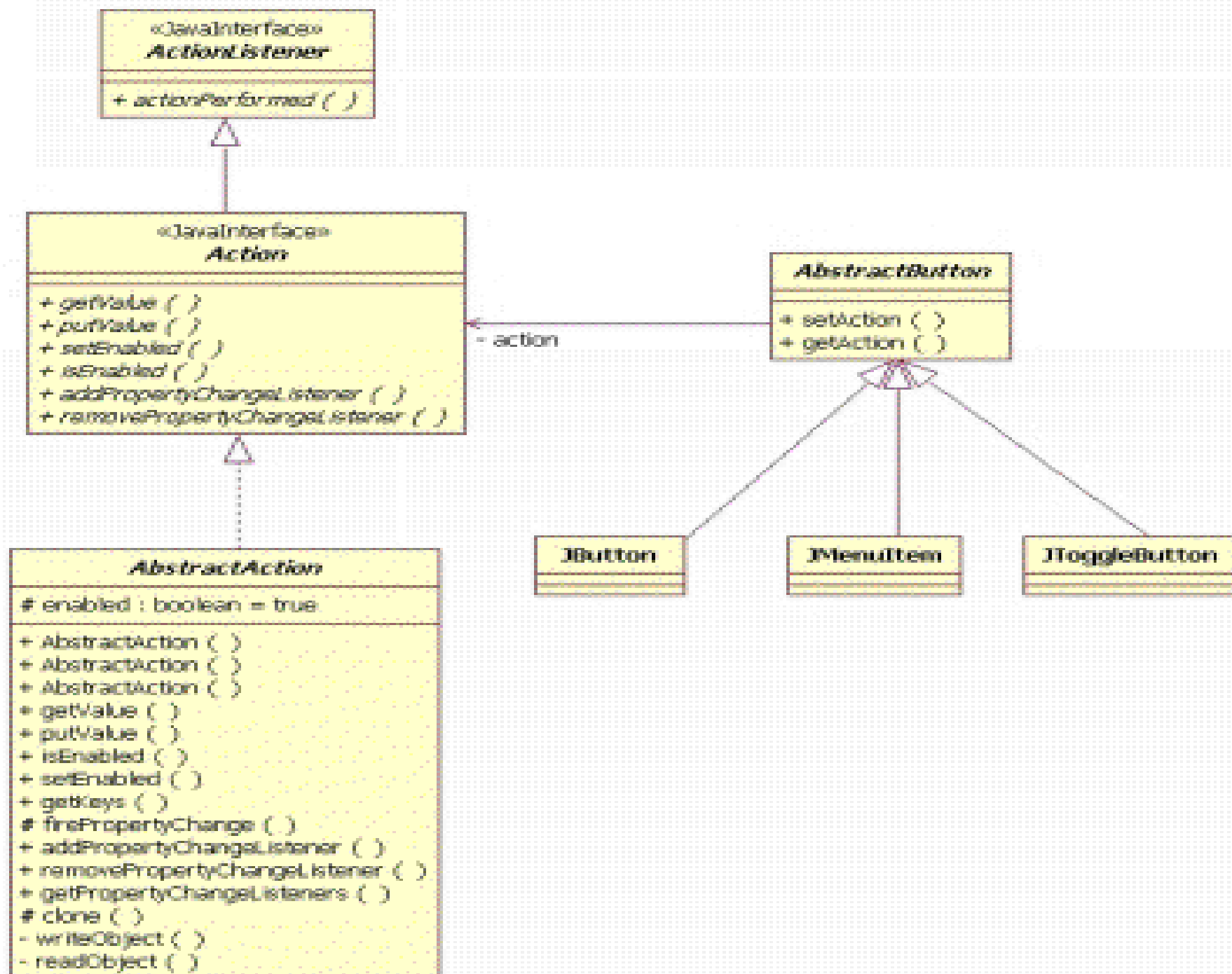
Command Pattern





Swing Action

- Command permite desacoplar el objeto invocante (Invoker) del receptor (Receiver, el que conoce cómo atender el requerimiento).
- Swing implementa el patrón Command con objetos de clases que implementen Action, por ej., subclasses de AbstractAction. Siempre que selecciona un ítem de un menú o pulsa un botón, estos objetos action emiten un requerimiento de una acción específica de la aplicación.

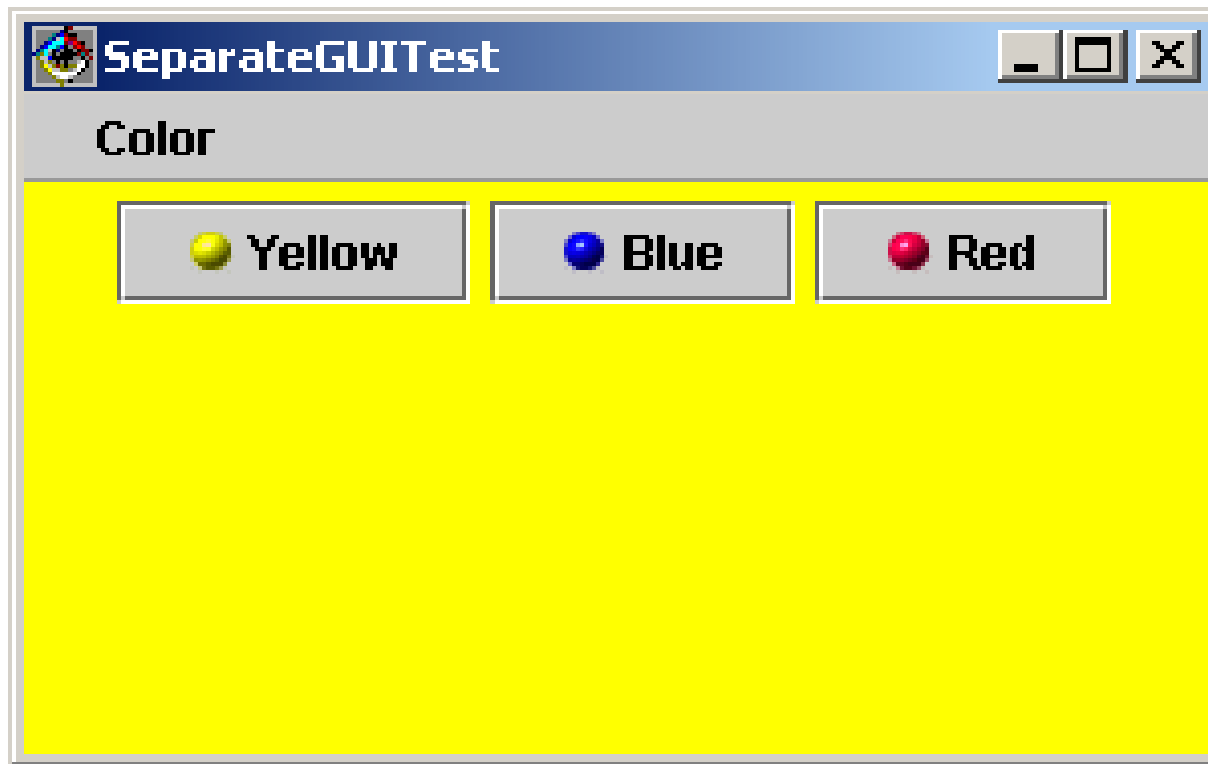




Swing Action

- Puede configurar un objeto JButton mediante una implementación concreta de Action que delega el requerimiento al Receiver.
- JButton actúa como Invoker llamando al método actionPerformed de un objeto concreto Action y no le importa qué objeto ejecuta realmente el comando requerido.
- El objeto Action delega el llamado al objeto Receiver que sabe cómo procesar el requerimiento, pero no quién lo hizo.
- El Receiver puede ser modificado o ser otro objeto y no afectar el código en JButton y del mismo modo se puede añadir el pedido de nuevos objetos, sin afectar al Receiver.

Ejemplo Swing Action





Código

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
class ColorAction extends AbstractAction{  
    private Component target;  
    public ColorAction(String name, Icon icon, Color c,  
        Component comp)  
    {putValue(Action.NAME, name);  
     putValue(Action.SMALL_ICON, icon);  
     putValue("Color", c);  
     target = comp;}
```



Código

```
public void actionPerformed(ActionEvent evt)
{ Color c = (Color)getValue("Color");
target.setBackground(c);
target.repaint();}}

class ActionButton extends JButton{
public ActionButton(Action a){
    setText((String)a.getValue(Action.NAME));
    Icon icon = (Icon)a.getValue(Action.SMALL_ICON);
    if (icon != null) setIcon(icon);
    addActionListener(a);}}
```



Código

```
class SeparateGUIFrame extends JFrame{  
public SeparateGUIFrame(){  
setTitle("SeparateGUITest");  
setSize(300, 200);  
addWindowListener(new WindowAdapter()  
{ public void windowClosing(WindowEvent e)  
{ System.exit(0);}});  
JPanel panel = new JPanel();  
Action blueAction = new ColorAction("Blue",new ImageIcon(  
"blue-ball.gif"),Color.blue, panel);
```



Código

```
Action yellowAction = new ColorAction("Yellow",  
new ImageIcon("yellow-ball.gif"),Color.yellow, panel);  
Action redAction = new ColorAction("Red",  
new ImageIcon("red-ball.gif"),Color.red, panel);  
panel.add(new JButton(yellowAction));  
panel.add(new JButton(blueAction));  
panel.add(new JButton(redAction));  
panel.registerKeyboardAction(yellowAction,  
KeyStroke.getKeyStroke(KeyEvent.VK_Y, 0),  
JComponent.WHEN_IN_FOCUSED_WINDOW);
```



Código

```
panel.registerKeyboardAction(blueAction,  
KeyStroke.getKeyStroke(KeyEvent.VK_B, 0),  
JComponent.WHEN_IN_FOCUSED_WINDOW);  
panel.registerKeyboardAction(redAction,  
KeyStroke.getKeyStroke(KeyEvent.VK_R, 0),  
JComponent.WHEN_IN_FOCUSED_WINDOW);  
Container contentPane = getContentPane();  
contentPane.add(panel);  
JMenu m = new JMenu("Color");  
m.add(yellowAction);
```



Código

```
m.add(blueAction);  
m.add(redAction);  
JMenuBar mbar = new JMenuBar();  
mbar.add(m);  
setJMenuBar(mbar);}}  
  
public class SeparateGUITest{  
    public static void main(String[] args){  
        JFrame frame = new SeparateGUIFrame();  
        frame.show();}}
```



Threads y Swing

- Si su programa crea y se refiere a la GUI en forma correcta, probablemente no tenga que preocuparse de este tema.
- Si su programa crea threads para realizar tareas que afecten la GUI, o si manipula la GUI ya visible en respuesta a algo que no sea un evento standard, sea cuidadoso!



Regla

- Regla del thread único:

“Una vez que un componente se “realizó”, todo código que pueda afectarlo o dependa del estado del mismo, debería ejecutarse en el “*event-dispatching thread*”.”

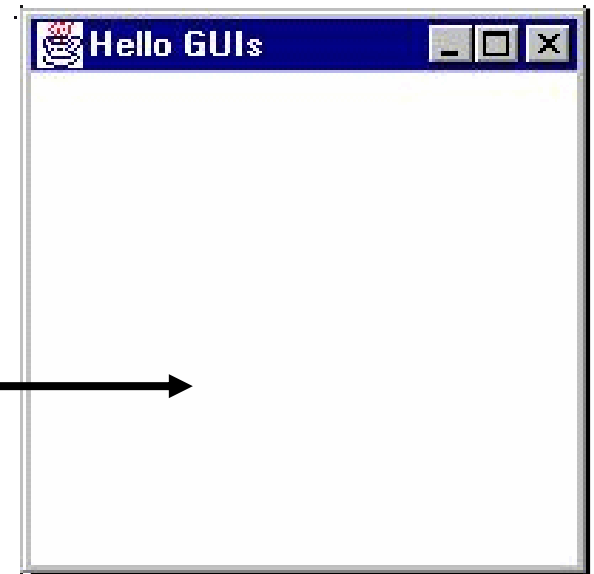
- Hay unas pocas excepciones:

<http://java.sun.com/docs/books/tutorial/uiswing/overview/threads.html>



La historia real


```
import java.awt.*;
public class HelloGUI {
    public static void main (String[ ] arg) {
        System.out.println
            ("About to make GUI");
        Frame f = new Frame ("Hello GUIs");
        f.setSize( 200, 200 );
        f.show();
        System.out.println
            ("Finished making GUI");
    } // main
} // class HelloGUI
```



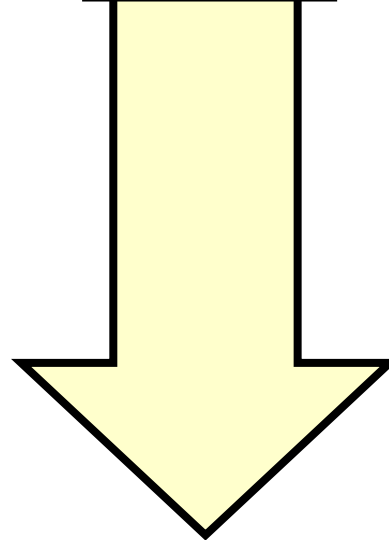
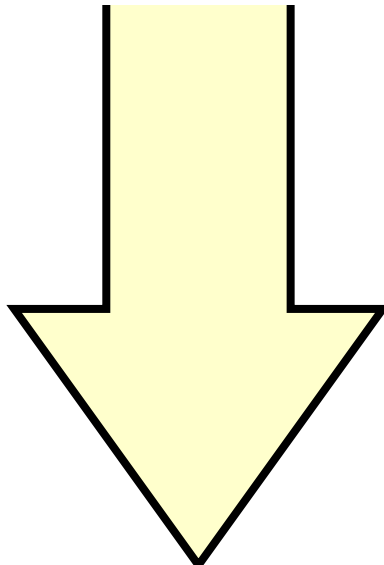
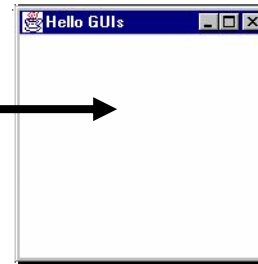
Usualmente se piensa en un programa como un único conjunto lineal de pasos a ejecutar, aunque ocurre algo especial cuando se crean objetos gráficos.



Event-dispatching thread

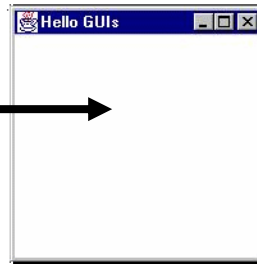


```
import java.awt.*;
public class HelloGUI {
    public static void main (String[] arg) {
        System.out.println
            ("About to make GUI");
        Frame f = new Frame ("Hello GUIs");
        f.setSize( 200, 200 );
        f.show();
        System.out.println
            ("Finished making GUI");
    } // main
} // class HelloGUI
```



Las dos áreas de un programa con GUI

```
import java.awt.*;
public class HelloGUI {
    public static void main (String[] arg) {
        System.out.println
            ("About to make GUI");
        Frame f = new Frame ("Hello GUIs");
        f.setSize( 200, 200 );
        f.show();
        System.out.println
            ("Finished making GUI");
    } // main
} // class HelloGUI
```



1

***El código que
escribe en main()
Y otros métodos***

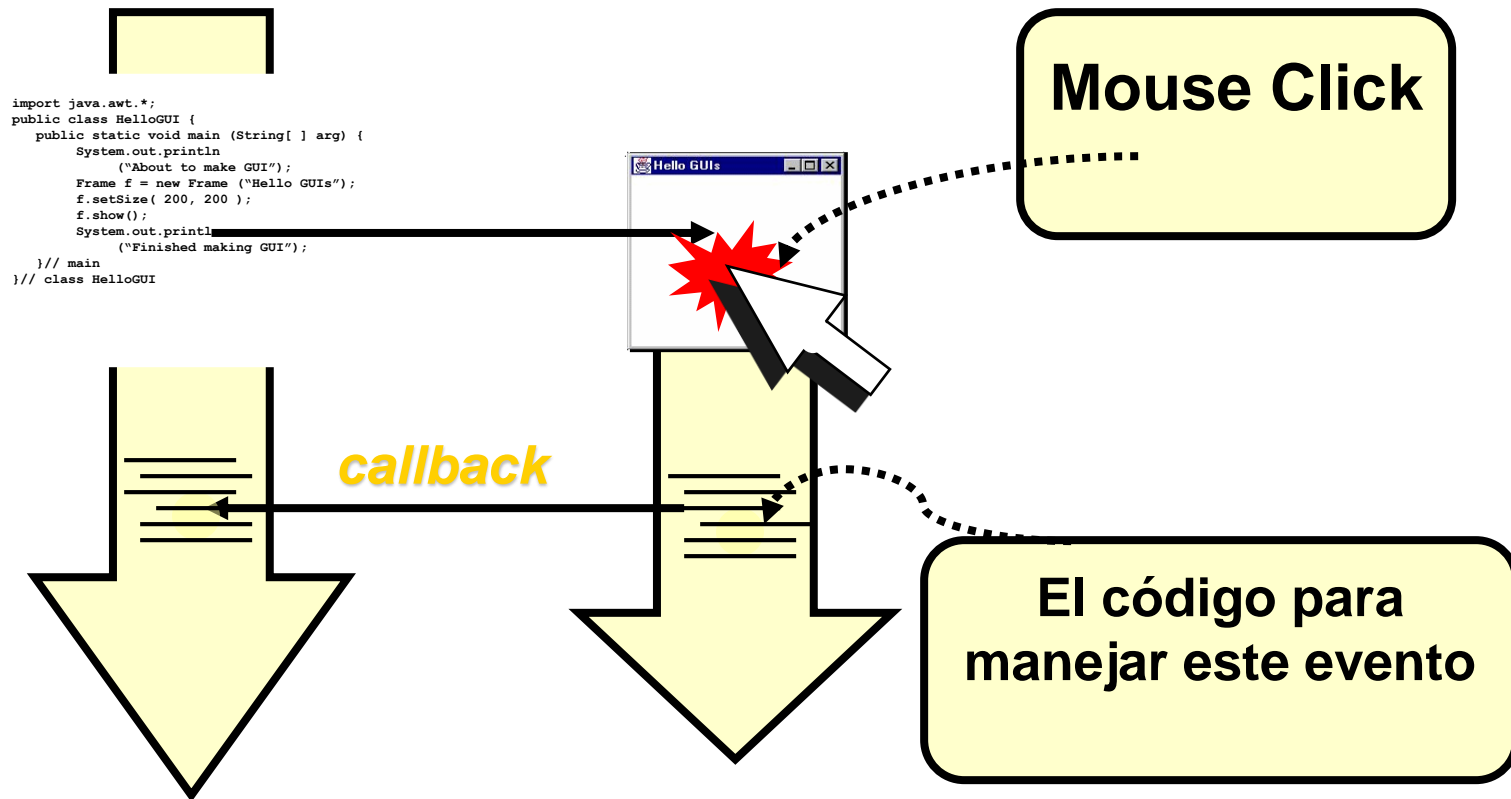
2

***El código que Java
provee para manejar
la parte gráfica de la
aplicación.***

Su código

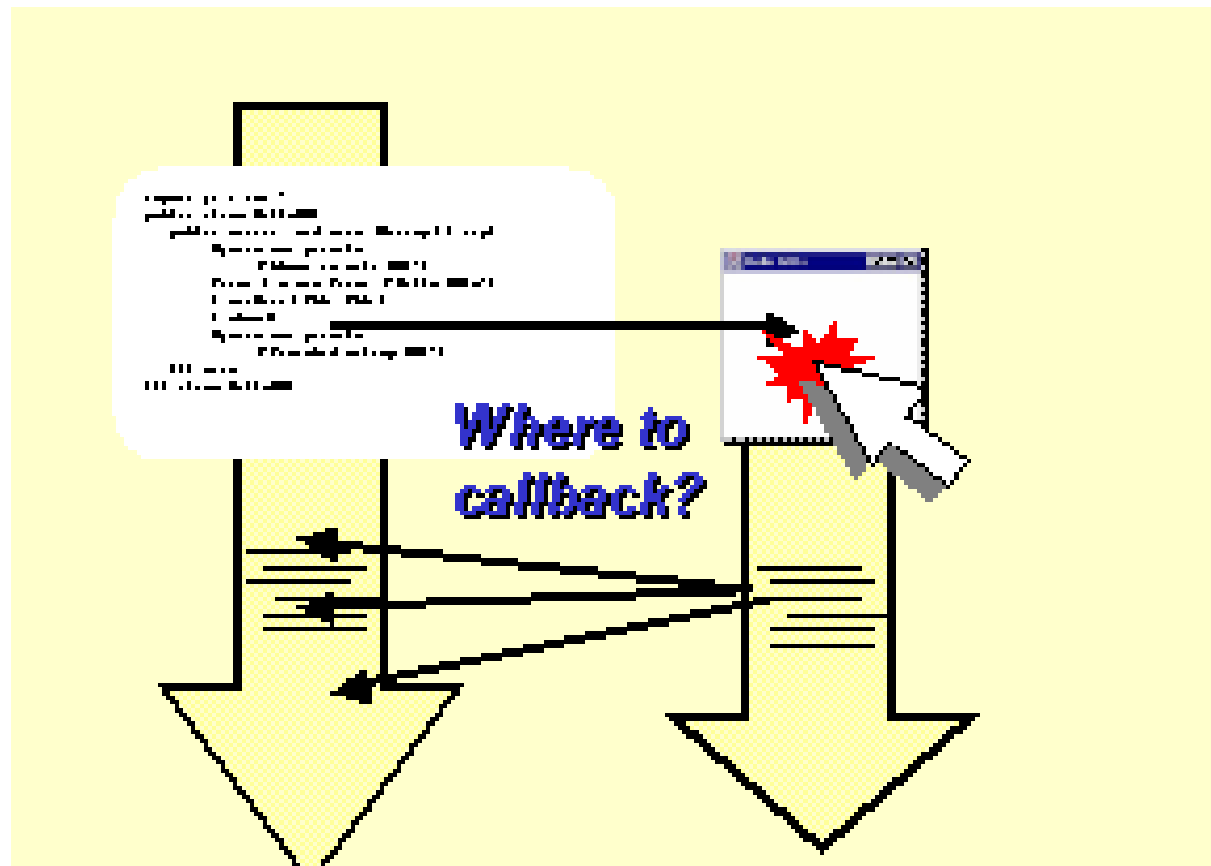
Graphics

Event-dispatching thread



Event-dispatching thread

Registro de eventos antes que ocurren



Event-dispatching thread

EventListener

```
import java.awt.*;  
public class HelloGUI {  
    public static void main (String[] arg) {  
        System.out.println  
            ("About to make GUI");  
        Frame f = new Frame ("Hello GUIs");  
        f.setSize( 200, 200 );  
        f.show();  
        System.out.println  
            ("Finished making GUI");  
    }  
} // main  
} // class HelloGUI
```



```
public void actionPerformed  
    (ActionEvent e) {  
    // code doing something  
}
```

callback

Este método **DEBE**
estar aquí, tal que Java lo
sabe y puede “callback”
al mismo.



Cuando conviene usar threads?

- Para mover, fuera del thread principal, una tarea de inicialización que requiera mucho tiempo y para que la GUI sea más rápida (ej.: cálculos intensivos, I/O en disco tal como carga de imágenes).
- Para mover fuera del event-dispatching thread, tareas que requieran mucho tiempo, de forma tal que la GUI permanezca **sensible**.
(*Ejecutar el código de manejo de eventos en un único thread garantiza que el manejo de cada uno de ellos terminará de ejecutarse antes del manejo del siguiente*)
- Para ejecutar una operación repetidamente, usualmente a iguales intervalos de tiempo.
- Para esperar mensajes de otros programas.



Programas Multi-threads

- Debería analizar el código de su programa y documentar desde cuál thread se invoca cada método.
- Si no está seguro si su código se está ejecutando en el thread de eventos use `SwingUtilities.isEventDispatchThread()` que retorna true si se está ejecutando en el EDT.
- Invoque `SwingUtilities.invokeLater` (preferido, retorna inmediatamente sin esperar que el EDT ejecute el código) o `SwingUtilities.invokeAndWait` desde cualquier thread para requerir que el EDT ejecute cierto código que involucre un acceso a la GUI.



Programas Multi-threads

Ejemplo: un thread que necesita acceder a un par de text fields:

```
void printTextField() throws Exception {  
    final String[] myStrings = new String[2];  
    Runnable getTextFieldText = new Runnable() {  
        public void run() {  
            myStrings[0] = textField0.getText();  
            myStrings[1] = textField1.getText(); } };  
    SwingUtilities.invokeLaterAndWait(getTextFieldText);  
    System.out.println(myStrings[0] + " " +  
myStrings[1]); }
```

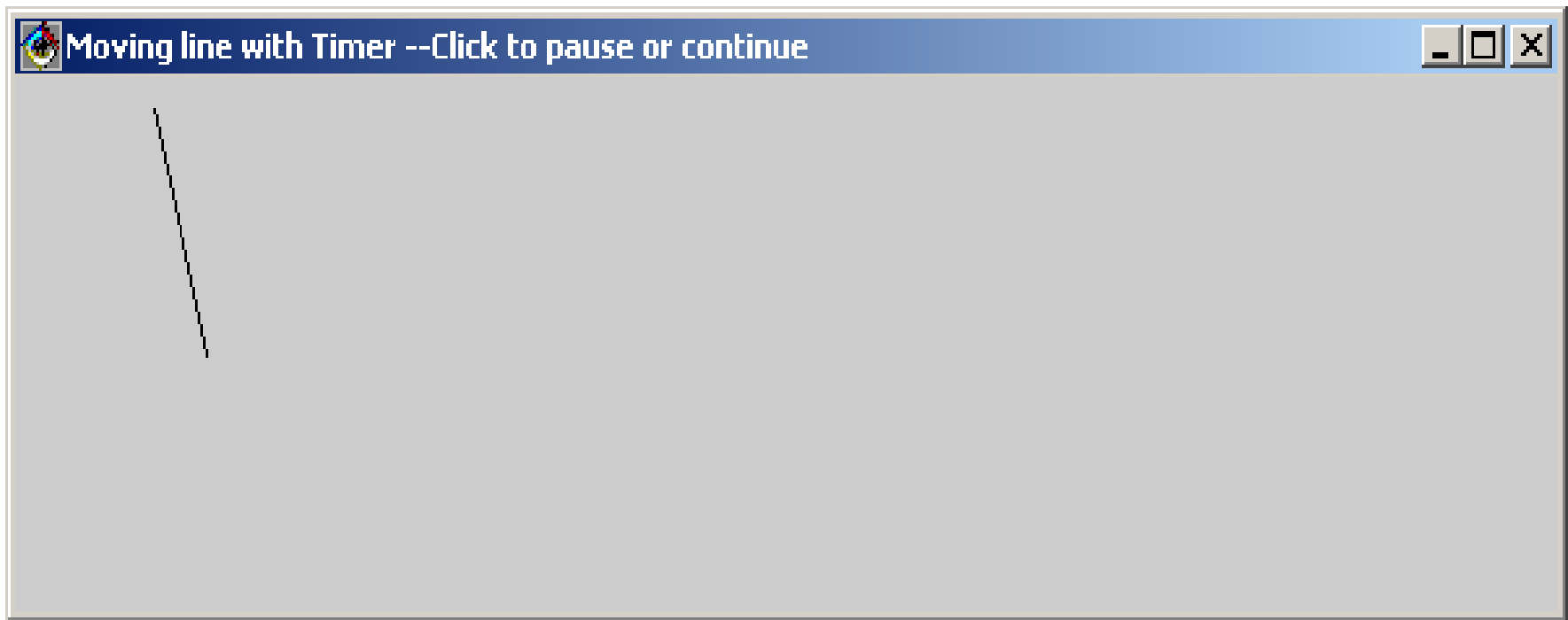


Programas Multi-threads

- **SwingWorker**
 - ◆ Crea un thread para ejecutar operaciones que requieran mucho tiempo. Después que las mismas finalizan, da la opción de ejecutar código adicional en el EDT.
 - ◆ no está en Swing release.
 - ◆ métodos `construct()` and `finish()`
- **javax.swing.Timer**
 - ◆ Se suele usar si se necesita actualizar un componente después de un delay o a intervalos regulares.



Ejemplo animación





Código

```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;  
public class Components extends JPanel {  
    int position = 10;  
    public void paintComponent(Graphics g) {  
        super.paintComponent(g);  
        g.drawLine(position,10,position + 20,90);  
        position ++;  
        if(position > 300) position = 10;}}
```



Código

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
  
public class MovingLine extends JFrame implements  
ActionListener {Timer timer;  
  
    boolean frozen = false;  
  
    Component moving;  
  
    MovingLine(int fps, String windowTitle) {  
        super(windowTitle);  
  
        int delay = (fps > 0) ? (1000 / fps) : 100;  
  
        timer = new Timer(delay, this);
```



Código

```
timer.setInitialDelay(0);  
timer.setCoalesce(true);  
addWindowListener(new WindowAdapter() {  
    public void windowIconified(WindowEvent e) {  
        stopAnimation();}  
    public void windowDeiconified(WindowEvent e) {  
        startAnimation();}  
    public void windowClosing(WindowEvent e) {  
        System.exit(0);} } );  
Component contents = new Components();
```



Código

```
contents.addMouseListener(new MouseAdapter() {  
    public void mousePressed(MouseEvent e) {  
        if (frozen) {  
            frozen = false;  
            startAnimation(); }  
        else {  
            frozen = true;  
            stopAnimation(); }} }));  
getContentPane().add(contents, BorderLayout.CENTER);}  
//Can be invoked by any thread (since timer is thread-safe).
```



Código

```
public void startAnimation() {
```

```
    if (frozen) {
```

```
        //Do nothing. The user has requested that we
```

```
        //stop changing the image.
```

```
    } else { //Start animating!
```

```
        if (!timer.isRunning()) {
```

```
            timer.start(); } } }
```

```
//Can be invoked by any thread (since timer is thread-safe).
```

```
public void stopAnimation() {
```

```
    //Stop the animating thread.
```

```
    if (timer.isRunning()) { timer.stop(); } }
```




Código

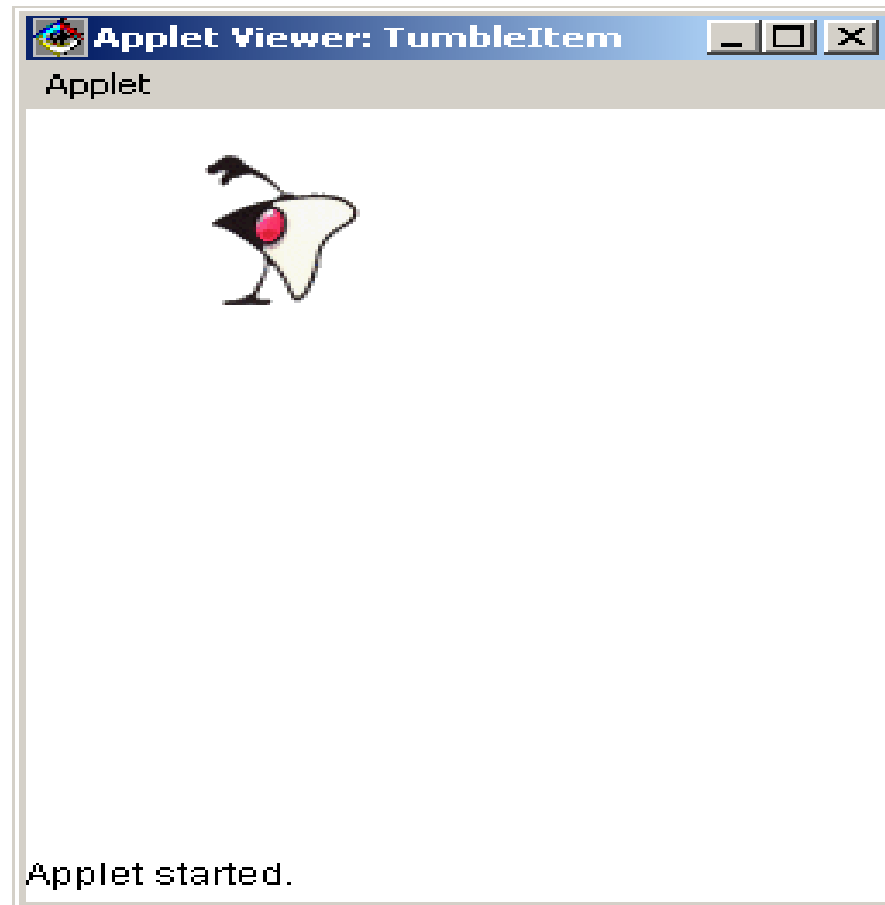
```
public void actionPerformed(ActionEvent e) {  
    //Advance the animation frame.  
    repaint(); }  
  
public static void main(String args[]) {  
    int fps = 10;  
    //Get frames per second from the command line argument.  
    if (args.length > 0) {try {  
        fps = Integer.parseInt(args[0]);  
    } catch (Exception e) {} }  
  
    MovingLine moving = new MovingLine(fps, "Moving line with  
    Timer --" + "Click to pause or continue");
```



Código

```
moving.pack();  
moving.setSize(600,200);  
moving.setVisible(true);  
//It's OK to start the animation here because  
//startAnimation can be invoked by any thread.  
moving.startAnimation(); }}
```

Applet, Timer y SwingWorker





Código

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

public class TumbleItem extends JApplet implements ActionListener {
    int loopslot = -1; //nro.de frame actual

    String dir;        //direct.relatoivo desde el cual cargo las 17 imágenes
    Timer timer;

    int pause;         //long. de pausas
    int offset;        //offset entre loops
    int off;           //offset actual
```



Código

```
int speed;           //veloc.de animación
int nimgs;           //nro.de imágenes a animar
int width;           //ancho del content pane del applet
JComponent contentPane;
ImageIcon imgs[];    //imágenes
int maxWidth;        //ancho máx.de las imágenes
boolean finishedLoading = false;
JLabel statusLabel;

static Color[] labelColor = { Color.black, Color.black,
Color.black, Color.black, Color.black, Color.white,
Color.white, Color.white,Color.white, Color.white };
```



Código

```
public void init() {  
    //parámetros del applet.....  
    //Anima de derecha a izquierda si offset es negativo.  
    width = getSize().width;  
    if (offset < 0) {off = width - maxWidth;}  
    //dibuja la imagen actual a un particular offset.  
    contentPane = new JPanel() {  
public void paintComponent(Graphics g) {super.paintComponent(g);  
    if (finishedLoading &&(loopslot > -1) && (loopslot < nimgs)) {  
        imgs[loopslot].paintIcon(this, g, off, 0);}}}  
    contentPane.setBackground(Color.white);
```



Código

```
setContentPane(contentPane);  
    // "Loading Images..." label  
    statusLabel = new JLabel("Loading Images...", JLabel.CENTER);  
    statusLabel.setForeground(labelColor[0]);  
    contentPane.setLayout(new BorderLayout());  
    contentPane.add(statusLabel, BorderLayout.CENTER);  
    // Set up timer, no comienza h/cargar todas las imágenes  
    timer = new Timer(speed, this);  
    timer.setInitialDelay(pause);  
    imgs = new ImageIcon[nimgs];  
    timer.start(); // comienza la animación
```



Código

//Carga de imágenes puede tomar mucho tiempo, las carga en un
SwingWorker thread

```
final SwingWorker worker = new SwingWorker() {  
    public Object construct() {  
        //Imágenes numeradas de 1 a nimgs,  
        for (int i = 0; i < nimgs; i++) {imgs[i] = createFrame(i+1);}  
        finishedLoading = true; return imgs;}  
    public void finished() {//saca la etiqueta "Loading images"  
        contentPane.removeAll();  
        contentPane.repaint();  
        loopslot = -1;}};
```




Código

```
worker.start();}
```

```
protected ImageIcon createFrame(int imageNum) {
```

```
    String path = dir + "/T" + imageNum + ".gif";
```

```
    URL imgURL = this.getClass().getResource(path);
```

```
    if (imgURL != null) {
```

```
        return new ImageIcon(imgURL);}
```

```
    else {System.err.println("Couldn't find file: " + path);
```

```
        return null;}}
```

```
public void start() { if (finishedLoading && (nimgs > 1)) {
```

```
    timer.restart();}}
```

```
public void stop() {timer.stop();}
```



Código

```
//actualiza nro.de frame y offset.
```

```
//si es el último frame, hacer una pausa larga entre loops
```

```
public void actionPerformed(ActionEvent e) {
```

```
    loopslot++;
```

```
    if (!finishedLoading) {
```

```
        int colorIndex = loopslot % labelColor.length;
```

```
        statusLabel.setForeground(labelColor[colorIndex]);
```

```
    return;}  
}
```



Código

```
if (loopslot >= nimgs) {  
    loopslot = 0;  
    off += offset;  
    if (off < 0) {  
        off = width - maxWidth;  
    } else if (off + maxWidth > width) {  
        off = 0;}}  
contentPane.repaint();  
if (loopslot == nimgs - 1) {  
    timer.restart();}}
```



Webgrafía

[www.dsi.fceia.unr.edu.ar/downloads/informatica/info_iii/**eventos**.ppt](http://www.dsi.fceia.unr.edu.ar/downloads/informatica/info_iii/eventos.ppt)