

Manejo de eventos de GUI

- Modelo de delegación: fuentes y escuchas de eventos de GUI
- Interfaces EventListeners
- Tipos de eventos AWT: AWTEvent
- Estrategias de implementación de escuchas de eventos:
 - Implementación de interfaces *listeners*
 - Clases Adapters
 - Clases Anónimas
- Algunos ejemplos



Manejo de eventos

El manejo de eventos de la **GUI** está basado en el **modelo de delegación**. Dicho modelo se basa en **objetos que originan o disparan eventos** llamados **fuentes de eventos** y **objetos que escuchan y atienden eventos** llamados **escuchas de eventos o listeners**.



Las **componentes de GUI**, como botones, listas, campos de texto, etc. son los objetos que **disparan eventos** y **delegan el manejo** de los mismos a otros objetos, los **escuchas de eventos**.

La esencia de este modelo es simple: **objetos que disparan eventos atendidos** por objetos **escuchas o listeners**.

En Java los escuchas o listeners son clases que implementan interfaces particulares conocidas como `AWTListeners`.

Manejo de eventos

Eventos

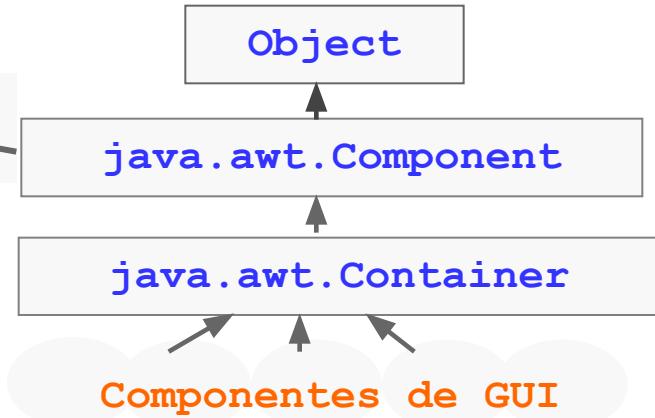
Un **evento** es generado por una componente de GUI como consecuencia de una acción iniciada por un usuario (presionar un botón, seleccionar un ítem de una lista, etc.)

Fuentes de Eventos

Las componentes de GUI son las que generan eventos. Estas componentes además responden a dos métodos para que los escuchas o *listeners* registren o quiten interés en los eventos que ellas generan:

```
public void addXXXListener(XXXListener)  
public void removeXXXListener(XXXListener)
```

Cuando una componente genera un evento, éste es pasado al manejador o a los manejadores que se registraron en ella.



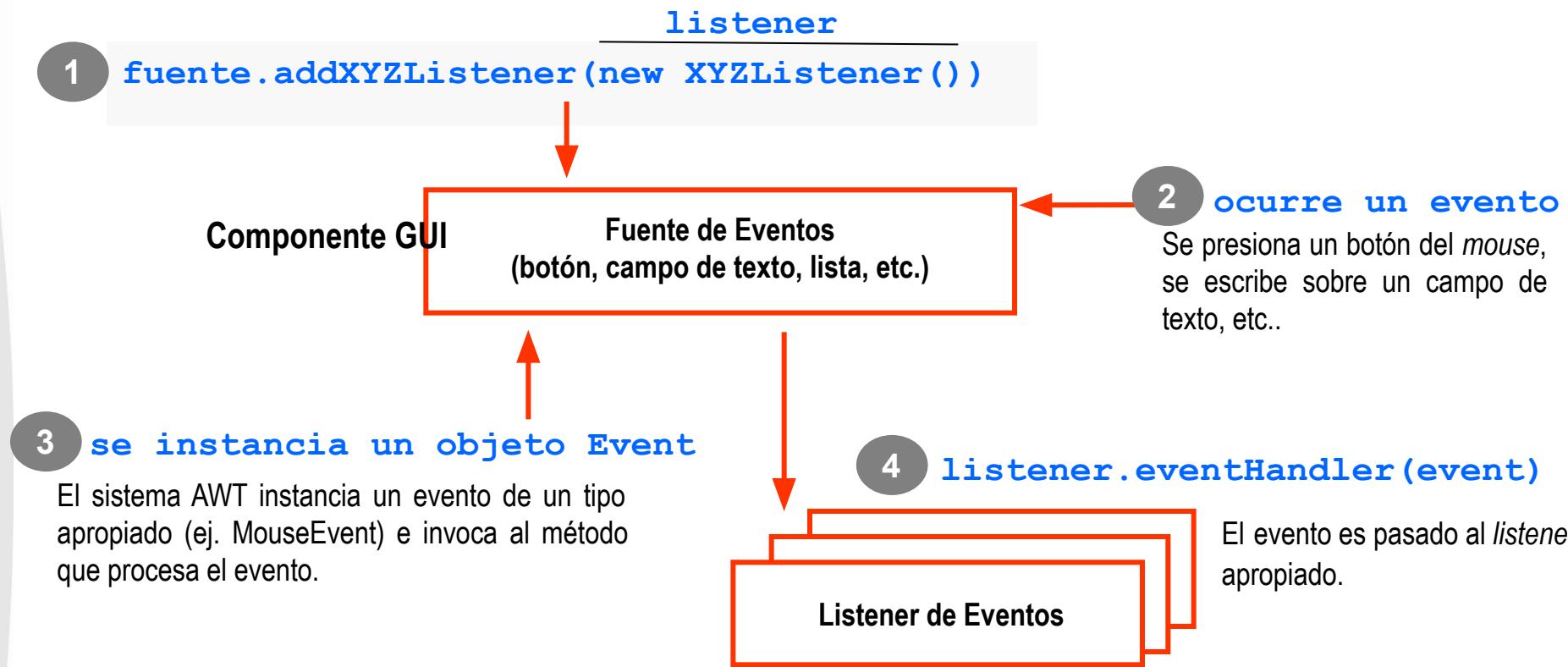
Manejadores de eventos: Listeners

Un *listener* es un objeto que implementa una determinada interface. Los métodos de estos objetos reciben como parámetro un objeto **AWTEvent** específico, que contiene información sobre el evento y sobre la componente AWT que lo disparó.

Manejo de eventos

Sobre una componente de interfaz de usuario se pueden registrar uno o más *listeners*. El orden en que los *listeners* son notificados del evento es indefinido.

¿Cómo funciona la delegación de eventos?



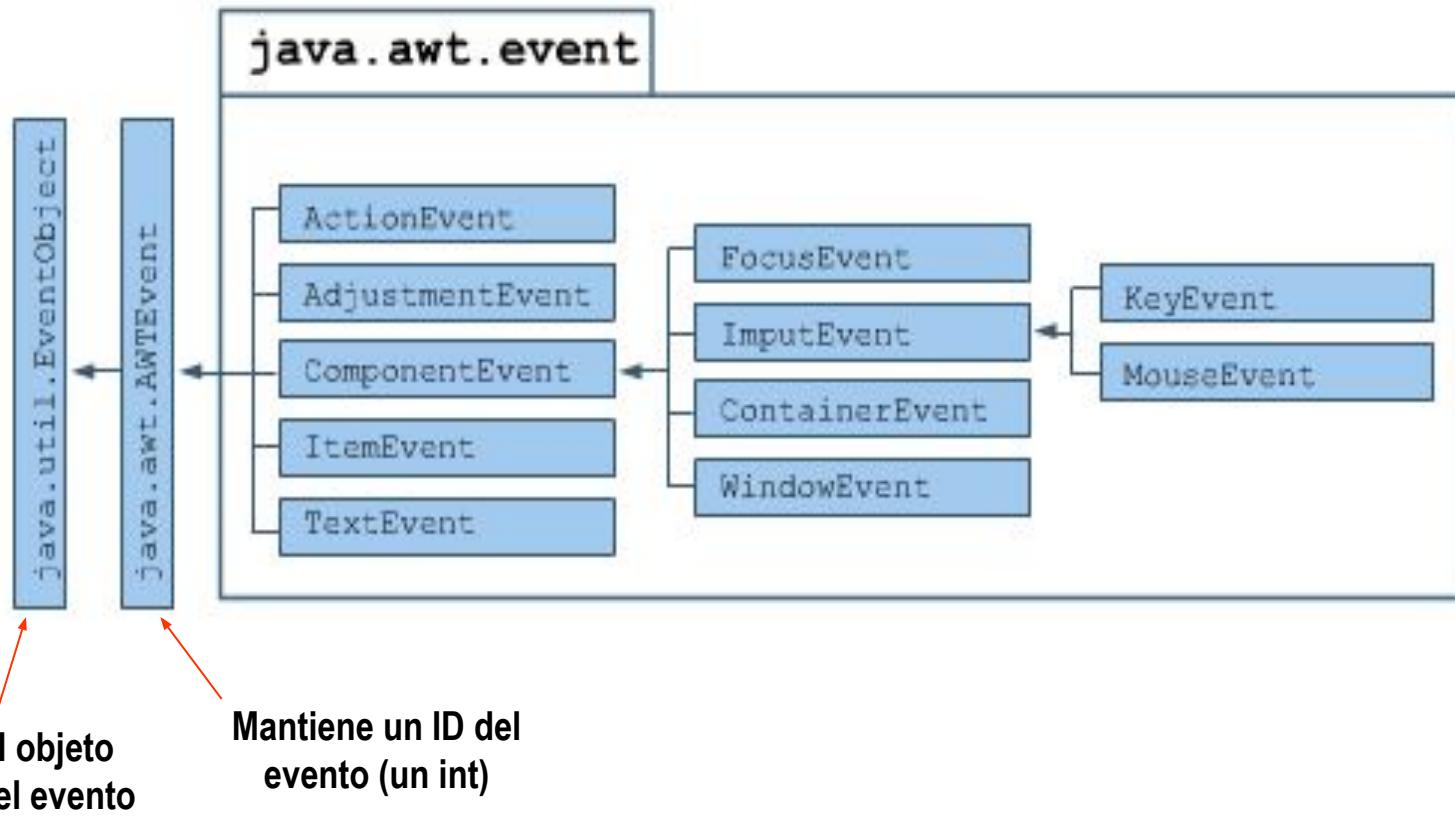
Interfaces EventListener en AWT

Para cada categoría de eventos hay una interface que debe ser implementada. Cada interface tiene uno o más métodos que deben ser implementados y serán invocados cuando ocurre un evento específico sobre la componente. Todas son subclases de `java.util.EventListener`

Interface o clase que la implementa	Métodos de la interface	Interface o clase que la implementa	Métodos de la interface
ActionListener	actionPerformed(ActionEvent)	MouseListener	mouseClicked(MouseEvent)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)	MouseAdapter	mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
ComponentListener ComponentAdapter	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)	MouseMotionListener MouseMotionAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
ContainerListener ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)	WindowListener WindowAdapter	windowOpened(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent)
FocusListener FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)	ItemListener	itemStateChanged(ItemEvent)
KeyListener KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)	MouseWheelListener	mouseWheelMoved(MouseWheelEvent e)

Tipos de eventos en AWT

Los métodos de las interfaces EventListeners contienen diferentes tipos de eventos. Estos eventos se organizan en una jerarquía, que es la siguiente:



La clase **java.util.EventObject** junto con las interfaces analizadas constituyen el fundamento del modelo de delegación de eventos.

Tipos de eventos en AWT

Algunos ejemplos

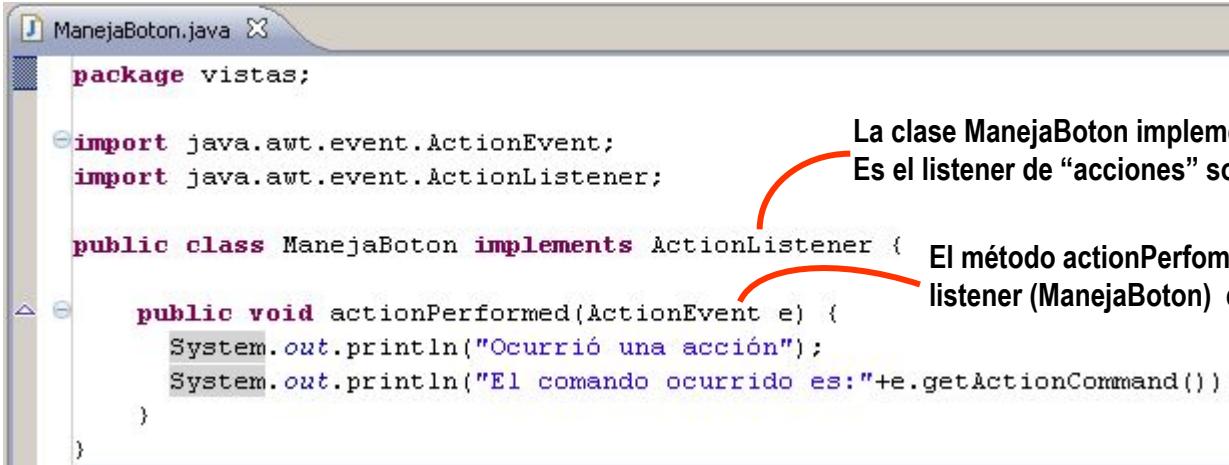
Tipo de Evento	Clase	¿Cuándo se dispara?	Listener	Características
Evento de Acción	ActionEvent	Clic en un Button (botón), Enter en un TextField	ActionListener	Capturan interacciones lógicas o acciones completas realizadas por el usuario.
Evento de Mouse	MouseEvent	Clic, doble clic, movimiento del mouse	MouseListener, MouseMotionListener	Más detallados, capturan la interacción física del mouse (ubicación, clic, arrastre, etc.)
Evento de Teclado	KeyEvent	Pulsar o soltar una tecla	KeyListener	Detectan entradas específicas del teclado por ejemplo, una tecla presionada.
Evento de Foco	FocusEvent	Ganar o perder el foco	FocusListener	Indican cambios en el foco de los componentes, como cuando un campo de texto se activa o desactiva.
Evento de Ítem	ItemEvent	Selección de un Checkbox o Choice	ItemListener	Capturan cambios en el estado de selección de elementos individuales.
Evento de Ventana	WindowEvent	Abrir, cerrar, minimizar una ventana	WindowListener, WindowAdapter	Relacionados con el estado y el ciclo de vida de las ventanas.

Manejo de eventos

Ejemplo: eventos “acción” y ActionListener

Para crear un objeto *listener* se debe implementar alguna de las interfaces *listener* provistas por la API.

En este ejemplo se implementa la interface **ActionListener**: maneja eventos genéricos de tipo **ActionEvent** y define un único método, el **actionPerformed (ActionEvent e)**.



```
ManejaBoton.java X
package vistas;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ManejaBoton implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Ocurrió una acción");
        System.out.println("El comando ocurrido es:" + e.getActionCommand());
    }
}
```

La clase ManejaBoton implementa la interface ActionListener.
Es el listener de “acciones” sobre un botón

El método actionPerformed() se ejecuta cuando el objeto listener (ManejaBoton) es notificado que ocurrió la “acción”.

¿Qué sucede si se presiona un botón que tiene registrado este manejador?

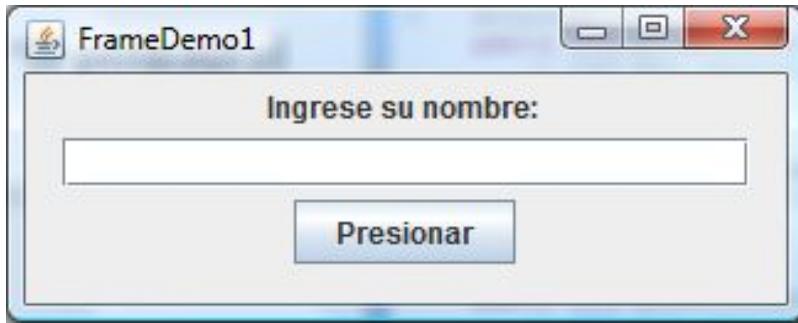
Se imprime en la consola lo siguiente:

Ocurrió una acción
El comando del botón es: ButtonPressed

Manejo de eventos

Ejemplo: eventos del Mouse

Supongamos que tenemos la siguiente clase, llamada **MiListener**, que implementa la interface **MouseListener** y está interesada sólo en saber cuando se hace “click” del mouse sobre el botón. Supongamos que se registra este listener en el botón “Presionar” ¿Qué sucede al presionarlo?

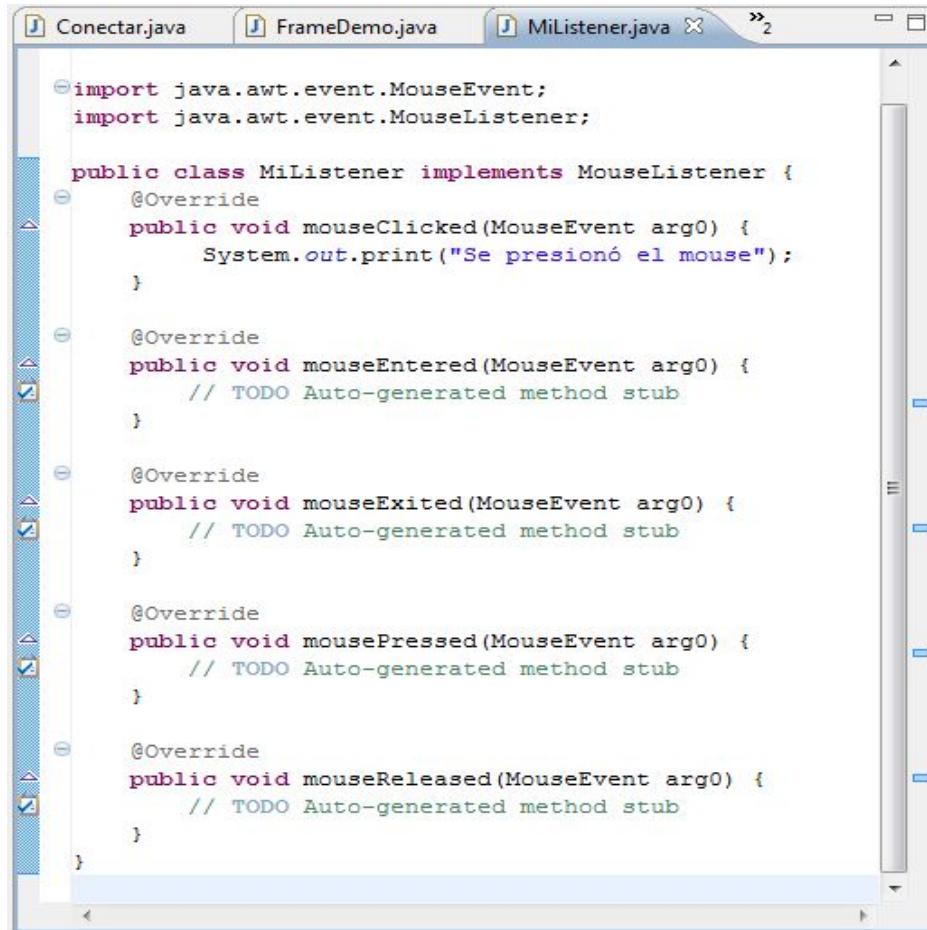


Se imprime en la consola “**Se presionó el mouse**”.
¿Qué sucede cuando me posiciono sobre el botón?

No hace NADA!!

Sólo interesa el método **mouseClicked(MouseEvent arg0)**, sin embargo se debieron implementar TODOS los métodos y dejar sus cuerpos vacíos.

¿Hay otra opción? Si ↗ Clases Adapters



```
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class MiListener implements MouseListener {
    @Override
    public void mouseClicked(MouseEvent arg0) {
        System.out.print("Se presionó el mouse");
    }

    @Override
    public void mouseEntered(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mouseExited(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mousePressed(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mouseReleased(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }
}
```

Clases Adapters

Para solo implementar los métodos necesarios de la interface listener, AWT ofrece un conjunto de clases que implementan las interfaces listeners cuyos métodos contienen cuerpos vacíos. Estas clases se denominan **Adapters** o **Adaptadoras**.

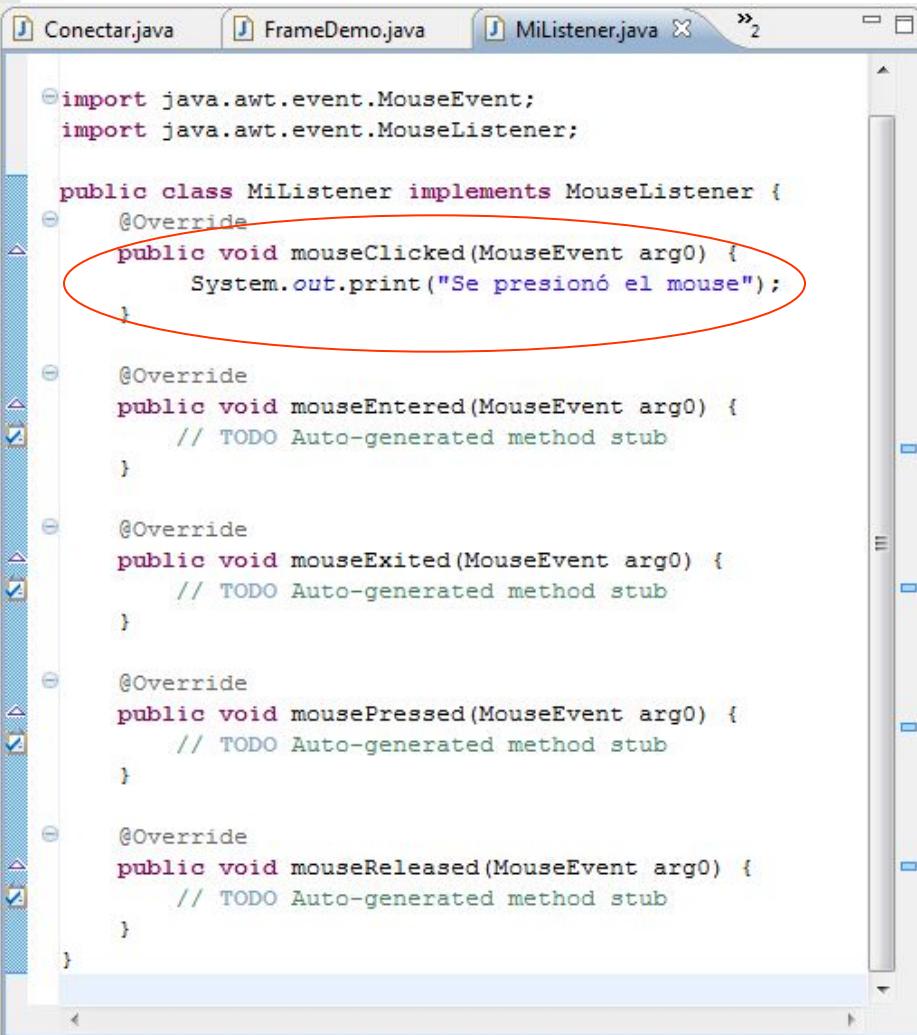
Por lo tanto, para **crear un Listener** se puede usar **herencia** es decir **extender una clase adaptadora y sobrescribir solamente los métodos que interesan**.

Interface listener ó Clase que la implementa	Métodos de la interface	Interface listener ó Clase que la implementa	Métodos de la interface
ActionListener	actionPerformed(ActionEvent)	MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)	MouseAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
ComponentListener ComponentAdapter	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)	WindowListener	windowOpened(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent)
ContainerListener ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)	ItemListener	itemStateChanged(ItemEvent)
FocusListener FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)	MouseWheelListener	mouseWheelMoved(MouseWheelEvent e)
KeyListener KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)		

Clases Adapter

Ejemplo: eventos del Mouse

Si se modifica el ejemplo anterior para usar la clase **Mousedeapter**, el código quedaría así:



```
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

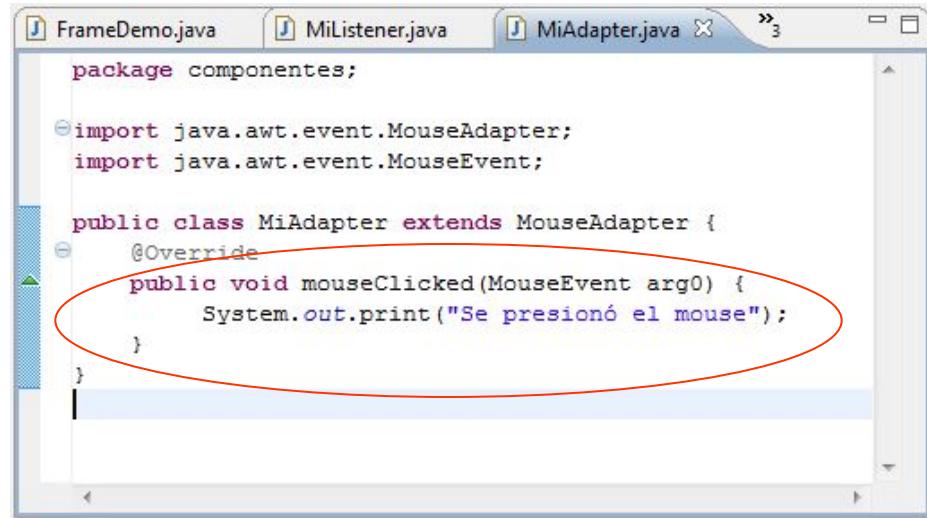
public class MiListener implements MouseListener {
    @Override
    public void mouseClicked(MouseEvent arg0) {
        System.out.print("Se presionó el mouse");
    }

    @Override
    public void mouseEntered(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mouseExited(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mousePressed(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }

    @Override
    public void mouseReleased(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }
}
```



```
package componentes;

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class MiAdapter extends MouseAdapter {
    @Override
    public void mouseClicked(MouseEvent arg0) {
        System.out.print("Se presionó el mouse");
    }
}
```

Usando clases adaptadoras, la clase se simplifica notablemente. Se obtiene un código más limpio y claro.

Registrando Listeners



Creamos componentes, analizamos
listeners y eventos, ahora ...
**¿Cómo registro listeners en
componentes?**

Para registrar un *listener* en una componente se utiliza el método **addxxxListener (unListener)**, donde xxx es la categoría de evento (mouse, window, key, etc.)

Continuando con el ejemplo anterior, **¿cómo registrar un listener que atienda los eventos específicos producidos por las interacciones físicas del mouse sobre el botón?**

unBoton.addMouseListener (unListener)

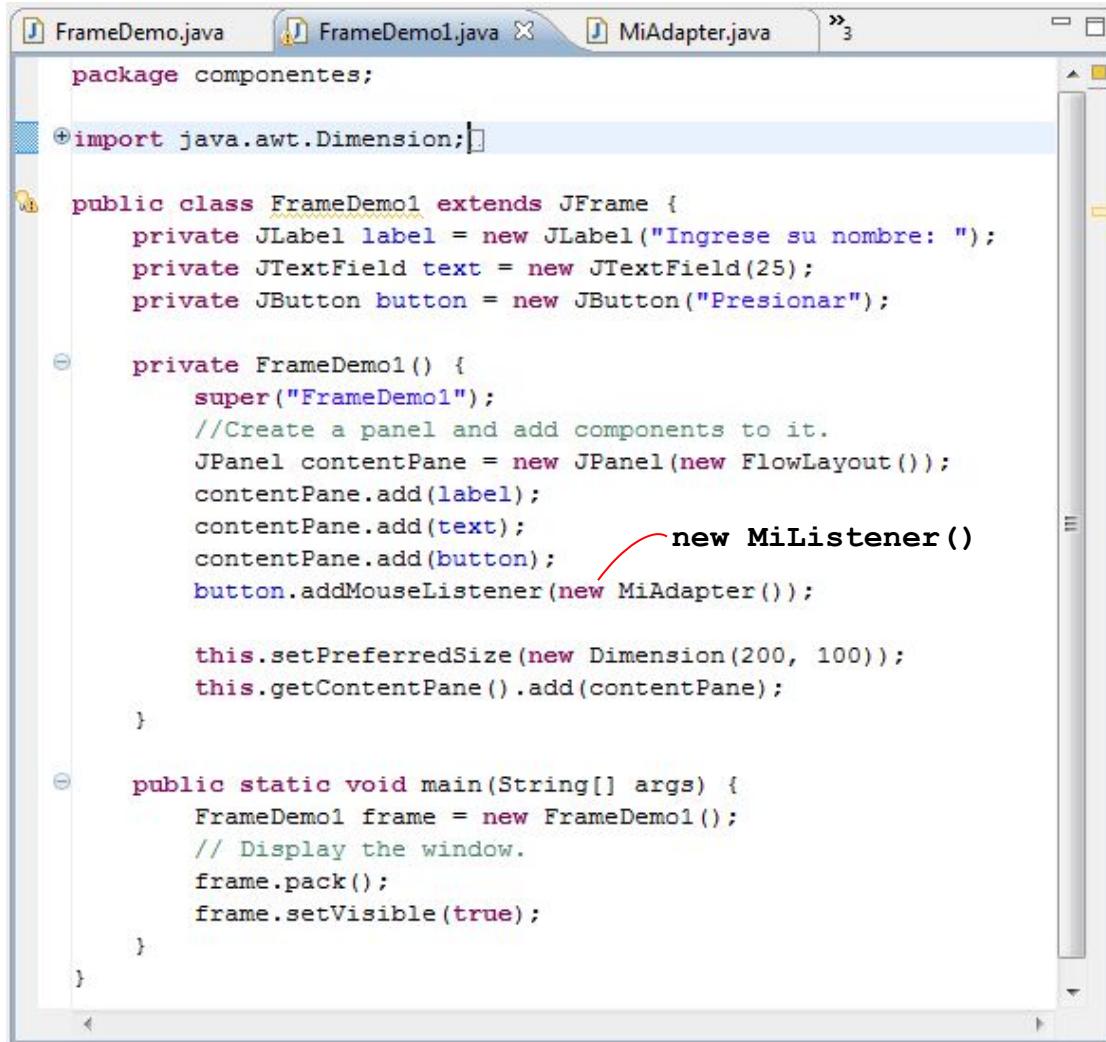
→ **unListener** es un instancia de
una clase que implementa
MouseListener

Registrando Listeners

Dos estrategias para registrar Listeners:

- a) definir la clase **MiListener** que **implementa** la interface **MouseListener**.
- b) definir la clase **MiAdapter** que **extiende** la clase **MouseAdapter**.

Ambas cumplen la misma función.



```
package componentes;

import java.awt.Dimension;

public class FrameDemo1 extends JFrame {
    private JLabel label = new JLabel("Ingrese su nombre: ");
    private JTextField text = new JTextField(25);
    private JButton button = new JButton("Presionar");

    private FrameDemo1() {
        super("FrameDemo1");
        //Create a panel and add components to it.
        JPanel contentPane = new JPanel(new FlowLayout());
        contentPane.add(label);
        contentPane.add(text);
        contentPane.add(button);
        button.addActionListener(new MiListener());
        button.addMouseListener(new MiAdapter());

        this.setPreferredSize(new Dimension(200, 100));
        this.getContentPane().add(contentPane);
    }

    public static void main(String[] args) {
        FrameDemo1 frame = new FrameDemo1();
        // Display the window.
        frame.pack();
        frame.setVisible(true);
    }
}
```



Listeners de Eventos

¿Dónde se ubican las clases listeners?

En los ejemplos que venimos analizando, la clase de la GUI es independiente de las clases que manejan eventos. En el primer ejemplo, el listener solo imprime en la consola, no requiere de información de la GUI.

¿Qué pasa si los listener necesitan acceder a las componentes de la GUI?

Supongamos que el listener **MiAdapter** quiere acceder a la variable de instancia **text**.

```
FrameDemo.java FrameDemo1.java MiAdapter.java
package componentes;
import java.awt.event.MouseAdapter;
public class MiAdapter extends MouseAdapter {
    private JTextField text = null;
    public MiAdapter(JTextField text) {
        this.text = text;
    }
    @Override
    public void mouseClicked(MouseEvent arg0) {
        System.out.print("Se presionó el mouse "+text.getText());
    }
}
```

```
FrameDemo.java FrameDemo1.java MiAdapter.java
package componentes;
import java.awt.Dimension;
public class FrameDemo1 extends JFrame {
    private JLabel label = new JLabel("Ingrese su nombre: ");
    private JTextField text = new JTextField(25);
    private JButton button = new JButton("Presionar");
    private FrameDemo1() {
        . . .
        button.addMouseListener(new MiAdapter(text));
        this.setPreferredSize(new Dimension(200, 100));
        this.getContentPane().add(contentPane);
    }
    public static void main(String[] args) {
    }
}
Clase de la GUI o Vista
```

Si tenemos que pasar todos los campos se hace muy complicado!! Para evitar esto, se suele definir a las clases *listener* dentro de la clase que modela a la GUI, como *clases internas*.

Listeners de Eventos

Clases internas

Se recomienda **escribir el Listener (MiAdapter o MiListener)** como una **clase interna** es decir directamente dentro de la clase que define la Interfaz de Usuario Gráfica o Vista de la aplicación.

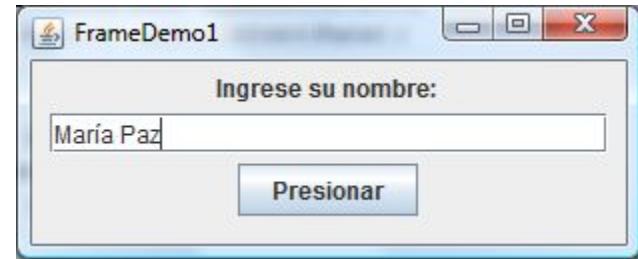
```
FrameDemo1.java
import java.awt.Dimension;
public class FrameDemo1 extends JFrame {
    private JLabel label = new JLabel("Ingrese su nombre:");
    private JTextField text = new JTextField(25);
    private JButton button = new JButton("Presionar");

    private FrameDemo1() {
        super("FrameDemo1");
        JPanel contentPane = new JPanel(new FlowLayout());
        contentPane.add(label);
        contentPane.add(text);
        contentPane.add(button);
        button.addMouseListener(new MiAdapter());
        this.setPreferredSize(new Dimension(200, 100));
        this.getContentPane().add(contentPane);
    }

    public static void main(String[] args) {
        FrameDemo1 frame = new FrameDemo1();
        frame.pack();
        frame.setVisible(true);
    }
}

private class MiAdapter extends MouseAdapter {
    @Override
    public void mouseClicked(MouseEvent arg0) {
        System.out.print("Se presionó el mouse " + text.getText());
    }
}
```

Manejador de evento



Las **clases internas** son **miembros** de la **clase de alto nivel** (que la contiene) y por lo tanto tienen acceso a todos sus miembros, inclusive los privados.

Las **clases internas** solo tienen sentido en la **clase contenedora**, por ello se recomienda definirlas **privadas**. Es una **clase desconocida afuera** de la clase contenedora.

Al hacer doble click sobre el botón **Presionar**, se imprime en la consola

Listeners de Eventos

Clases anónimas

Otra alternativa para escribir Listeners es utilizar **clases anónimas**. Las **clases anónimas son clases internas sin nombre**, cumplen la misma función.

```
FrameDemo.java FrameDemo1.java MiListener.java MiAdapter.java > 2

package componentes;

import java.awt.Dimension;
public class FrameDemo1 extends JFrame {
    private JLabel label = new JLabel("Ingrese su nombre:");
    private JTextField text = new JTextField(25);
    private JButton button = new JButton("Presionar");

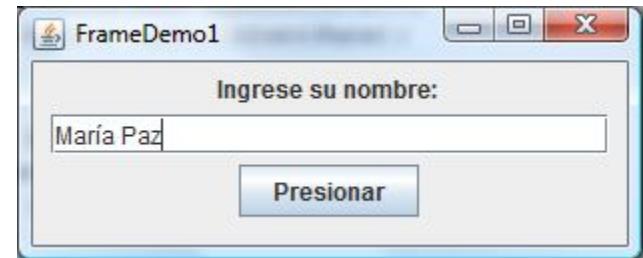
    private FrameDemo1() {
        super("FrameDemo1");
        JPanel contentPane = new JPanel(new FlowLayout());
        contentPane.add(label);
        contentPane.add(text);
        contentPane.add(button);
        button.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent arg0) {
                System.out.print("Se presionó el mouse " + text.getText());
            }
        });
        this.setPreferredSize(new Dimension(200, 100));
        this.getContentPane().add(contentPane);
    }

    public static void main(String[] args) {
        FrameDemo1 frame = new FrameDemo1();
        frame.pack();
        frame.setVisible(true);
    }
}

Problems @ Javadoc Declaration Console
FrameDemo1 [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (23/10/2011 20:45:04)
Se presionó el mouse María PazSe presionó el mouse María Paz
```

¿Cuándo es apropiado implementar listener como clases anónimas?

- El código del listener es específico de la componente y no se comparte con otras.
- El código del listener es simple, contiene pocas líneas de código.



Al hacer doble click sobre el botón **Presionar**, se imprime en la consola

Listeners de Eventos

Eventos del Teclado

Los eventos de teclado (*key events*) son disparados por algunas componentes de GUI cuando el usuario presiona o libera una tecla del teclado. Las notificaciones son acerca de dos clases de eventos:

- La tipificación de un carácter Unicode (*key pressed*)
- El evento de presionar o liberar alguna tecla (*key-pressed or key-released*)

```
package juego;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
.
.
public class MoverConTeclado implements KeyListener {
    public void keyReleased(KeyEvent evt) {
    }
    public void keyPressed(KeyEvent evt) {
        int ckey = evt.getKeyCode();
        Point p = pac.getLocation();
        if (ckey == 38) { //Up      ckey == KeyEvent.VK_UP
            pac.setLocation(new Point((int)p.getX(), (int)p.getY()-10));
            pac.setIcon(pacimg[3]);
        }
        if (ckey == 40) { //Down
            pac.setLocation(new Point((int)p.getX(), (int)p.getY()+10));
            pac.setIcon(pacimg[2]);
        }
        .
        .
    }
    public void keyTyped(KeyEvent evt) {
    }
    .
}
```

Métodos de la interface

Se implementa la interface Keylistener

Se implementa el método keyPressed() para analizar la tecla presionada

ckey == KeyEvent.VK_UP

ckey == KeyEvent.VK_DOWN



Listeners de Eventos

Eventos del Teclado

```
package juego;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
. .
public class MoverConTeclado {
    public ImageIcon[] pacimg = new ImageIcon[4];
    private JLabel pac = null;

    private void createAndShowGUI() {
        JPanel gamePanel = new JPanel();
        this.LoadGraphics();
        pac = new JLabel(pacimg[0]);
        pac.setLocation(50, 50);
        pac.setFocusable(true);
        pac.addKeyListener(new ManejaEventosTeclado());
        gamePanel.add(pac);
        frame = new JFrame("PacMan ALONE");
        frame.setPreferredSize(new Dimension(300, 500));
        frame.getContentPane().add(gamePanel);
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        MoverConTeclado pacman = new MoverConTeclado();
        pacman.createAndShowGUI();
    }
. .
}
```

Para que una componente dispare eventos del teclado, debe tener el foco del teclado

En este caso se extiende la clase KeyAdapter y es una clase interna de MoverConTeclado

```
class ManejaEventosTeclado extends KeyAdapter {
    public void keyPressed(KeyEvent evt) {
        int ckey = evt.getKeyCode();
        Point p = pac.getLocation();
        if (ckey == 38) { ckey == KeyEvent.VK_UP
            pac.setLocation(
                new Point((int) p.getX(), (int) p.getY() - 10));
            pac.setIcon(pacimg[3]);
            NewKeydir = 1; // UP
        }
        if (ckey == 40) { ckey == KeyEvent.VK_DOWN
            pac.setLocation(
                new Point((int) p.getX(), (int) p.getY() + 10));
            pac.setIcon(pacimg[2]);
            NewKeydir = 3; // Down
        }
        . .
    }
}
```

