




2 DE MAYO DE 2019

EVENTOS Y CONTROLES EN JAVA

ALUMNO: MARTÍN ALEJANDRO DOMÍNGUEZ LÓPEZ
DOCENTE: ING. MARIO ALBERTO MOERNO AGUILAR
PROGRAMACIÓN VISUAL
IDS 3°A



Eventos

Cuando el usuario de un programa o applet mueve el ratón, o hace un clic o usa el teclado, genera un Evento. En Java los eventos, como cualquier otra cosa, se representan como instancias u objetos de alguna clase. Para programar una interfaz gráfica es necesario aprender a utilizar los eventos.

Cuando el usuario interactúa sobre los diversos componentes del awt, estos generan eventos. La siguiente tabla muestra los eventos que cada tipo de componente puede generar y cuándo los genera.

Tipo Componente	Eventos generados	Hechos que los generan
Button	ActionEvent	El usuario hace un clic sobre el botón.
Checkbox	ItemEvent	El usuario selecciona o deselecciona el interruptor (Checkbox)
CheckboxMenuItem	ItemEvent	El usuario selecciona o deselecciona el interruptor (Checkbox)
Choice	ItemEvent	El usuario selecciona o deselecciona un elemento de la lista
Component	ComponentEvent	El componente se mueve, cambia de tamaño, se esconde o se exhibe
	FocusEvent	El componente gana o pierde el foco
	KeyEvent	El usuario pulsa o suelta una tecla
	MouseEvent	El usuario pulsa o suelta un botón del ratón, el cursor del ratón entra o sale o el usuario mueve o arrastra el ratón
Container	ContainerEvent	Se agrega o se quita un componente al contenedor
List	ActionEvent	El usuario hace doble clic en un elemento de la lista
	ItemEvent	El usuario selecciona o deselecciona un elemento de la lista
MenuItem	ActionEvent	El usuario selecciona un elemento del menú
Scrollbar	AdjustmentEvent	El usuario mueve la barra de desplazamiento

TextComponent	TextEvent	El usuario hace un cambio en el texto
TextField	ActionEvent	El usuario termina de editar el texto (hace un intro)
Window	WindowEvent	La ventana se abre, se cierra, se minimiza, se reestablece o se cierra.

Todos los eventos mencionados en la tabla están en el paquete **java.awt.event**.

Para facilitar la tarea del programador se han creado una serie de interfaces que deben implementarse cuando se quieren procesar algunos de estos eventos. La siguiente tabla presenta estas interfaces con sus métodos.

Interfaz	Métodos
ActionListener	actionPerformed(ActionEvent)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)
ComponentListener	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
ItemListener	itemStateChanged(ItemEvent)
KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
TextListener	textValueChanged(TextEvent)
WindowListener	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)

¿Cómo se utilizan estas interfaces para programar una interfaz gráfica?

Supongamos que deseamos que en un Componente que estamos desarrollando (típicamente un Applet, Marco, Diálogo, Ventana o Panel) responda a los eventos generados por el usuario sobre el mismo componente o sobre algunos otros (típicamente contenidos en él). Para ello convertimos a este componente en "escucha" (Listener) de ciertos eventos generados por él o por los otros componentes. Convertir a un componente en escucha de un tipo de eventos consiste en:

1. declarar que implementa la interfaz correspondiente,
2. implementar los métodos de la interfaz y
3. agregarlo a la lista de escuchas del o de los componentes que originan ese tipo de eventos. Esto se hace normalmente usando un método como addActionListener, addMouseListener, etc...

Por ejemplo, si queremos que un Applet responda a los movimientos del ratón sobre el applet y a los clics sobre un botón b colocado en el applet, ser necesario declarar en su línea definitoria que implementa MouseMotionListener y ActionListener, luego habrá que implementar los métodos de ambas interfaces y también habrá que agregar el applet a la lista de escuchas de eventos del ratón del propio applet y a la lista de escuchas de ActionEvent del botón. Un ejemplo podría ser:

```
public class miApplet extends Applet implements
    MouseMotionListener, ActionListener {
    public void init() {
        this.addMouseMotionListener(this); b.addActionListener(this);
    }
    /* -- MouseMotionListener methods --*/
    public void mouseDragged(MouseEvent e) {
    }
    public void mouseMoved(MouseEvent e) {
    }
    /* -- ActionListener method --*/
    public void actionPerformed(ActionEvent e) {
        ...
    }
}
```

- Cualquier clase puede recibir y manejar los eventos. Normalmente:
 - componentes generarán eventos en respuesta a las acciones de los usuarios.
 - objetos del usuario escucharán y atenderán los eventos generados.
- Se definen varios tipos de eventos. En java 1.0 sólo existía un super-evento que representaba todo. Cada tipo de evento tiene campos y métodos específicos.
- Los objetos escuchadores deben registrarse en los generadores para que estos les envíen los eventos. Cuando se produce un evento, el generador invoca un método en todos los objetos escuchadores registrados. El método que se invoca depende del tipo de evento. Estos métodos se definen en varias interfaces llamadas escuchadoras. Las clases escuchadoras deben implementar las interfaces escuchadores asociadas a los tipos de eventos que quieran atender.
- Los métodos de los generadores para registrar y dar de baja a los escuchadores son addXXX y removeXXX, donde XXX es el nombre de la interfaz escuchadora.

Tipos de eventos, sus generadores e interfaces escuchadores:

Tipo de Evento (tipos 1.0)	Componente Generador	Interfaz
ACTION_EVENT	Button, List, MenuItem, TextField	ActionListener
	CheckBox, Choice	ItemListener
GOT_FOCUS	Component	FocusListener
LOST_FOCUS		
KEY_ACTION	Component	KeyListener
KEY_ACTION_RELEASE		
KEY_PRESS		
KEY_RELEASE		
LIST_DESELECT	Checkbox, CheckboxMenuItem, Choice, List	ItemListener
LIST_SELECT		
MOUSE_DOWN	Canvas, Dialog, Frame, Panel, Window	MouseListener
MOUSE_ENTER		
MOUSE_EXIT		
MOUSE_UP		
MOUSE_DRAG	Canvas, Dialog, Frame, Panel, Window	MouseMotionListener
MOUSE_MOVE		
SCROLL_ABSOLUTE	Scrollbar	AdjustmentListener
SCROLL_BEGIN		
SCROLL_END		
SCROLL_LINE_DOWN		
SCROLL_LINE_UP		
SCROLL_PAGE_DOWN		
SCROLL_PAGE_UP		
WINDOW_DEICONIFY	Dialog, Frame	WindowListener
WINDOW_DESTROY		
WINDOW_EXPOSE		

WINDOW_ICONIFY		
WINDOW_MOVED	Dialog, Frame	ComponentListener

- Los componentes Label no generan ningún tipo de evento. Las listas generan un evento ItemEvent al seleccionar o deseleccionar elementos, y un evento ActionEvent al hacer doble click sobre un elemento.
- Existen clases adaptadoras cuyo objetivo es evitar que las clases escuchadoras tengan que implementar todo el interfaz escuchador. Las clases adaptadoras implementan los métodos de las interfaces escuchadoras con un cuerpo vacío. Así, las clases escuchadoras sólo tienen que extender a los adaptadores y redefinir únicamente el cuerpo de los métodos que necesitan. Por ejemplo, un escuchador puede extender la clase KeyAdapter en vez de implementar la interfaz KeyListener, y así sólo redefinir los métodos que le afecten.
- Métodos de las interfaces escuchadoras:

Interfaz	Métodos
ActionListener	actionPerformed(ActionEvent)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)
ComponentListener ComponentAdapter	componentHidden(componentEvent) componentShown(componentEvent) componentMoved(componentEvent) componentResized(componentEvent)
FocusListener FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)
KeyListener KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener MouseMotionAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
WindowListener WindowAdapter	windowOpened(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent)

	windowDeactivated(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent)
ItemListener	itemStateChanged(itemEvent)
TextListener	textValueChanged(TextEvent)

Manejo de los eventos de acción

- Clase de evento: `ActionEvent`.
- Debe implementarse el interfaz `ActionListener`.
Tiene un único método:

- `public abstract void actionPerformed(ActionEvent event);`

- Métodos de la clase `ActionEvent`:

- `public String getActionCommand()`
- `// devuelve la etiqueta del generador`

- Métodos de la clase `EventObject`:

- `public Object getSource()`
- `// Objeto donde se genero el evento.`

- Ejemplo:

```
• import java.awt.event.*;
• import java.awt.*;
• import java.applet.*;
•
• public class Colores extends Applet
•     implements ActionListener {
•     Color c;
•
•     public void init() {
•         Button b = new Button("Pulsame");
•         add(b);
•         b.addActionListener(this);
•         c = Color.black;
•     }
•
•     public void paint(Graphics g) {
•         g.setColor(c);
•         g.drawString("Texto rojo o negro", 55, 80);
•     }
•
•     public void actionPerformed(ActionEvent event) {
•         if (c == Color.black)
•             c = Color.red;
•         else
•             c = Color.black;
•         repaint();
•     }
• }
```

Manejo de los eventos de teclado

- Clase de evento: KeyEvent.
- Debe implementarse el interfaz KeyListener.
Tiene tres métodos:

- `public abstract void keyTyped(KeyEvent event);`
- `public abstract void keyPressed(KeyEvent event);`
- `public abstract void keyReleased(KeyEvent event);`

- Métodos de la clase KeyEvent:

- `public int getKeyCode()`
- `// devuelve el código de la tecla.`
-
- `public char getKeyChar()`
- `// devuelve el carácter de la tecla.`
- `// solo para teclas normales.`
-
- `public boolean isActionKey();`
- `// indica si es una tecla de acción:`
- `// F1..F12, home, flechas, etc.`

- Los códigos de las teclas son:

Código
KeyEvent.F1-KeyEvent.F12
KeyEvent.LEFT
KeyEvent.RIGHT
KeyEvent.UP
KeyEvent.DOWN

KeyEvent.END
KeyEvent.HOME
KeyEvent.PGDN
KeyEvent.PGUP
KeyEvent.PRINT_SCREEN
KeyEvent.SCROLL_LOCK
KeyEvent.CAPS_LOCK
KeyEvent.NUM_LOCK
KeyEvent.PAUSE
KeyEvent.INSERT
KeyEvent.DELETE
KeyEvent.ENTER
KeyEvent.TAB
KeyEvent.BACK_SPACE
KeyEvent.ESCAPE

- Los modificadores de teclas y de ratón se definen en la clase `InputEvent`. Tanto la clase `KeyEvent` como la clase `MouseEvent` se derivan de `InputEvent`. En la sección **La clase `InputEvent`** se define el manejo de los modificadores.

Manejo de los eventos de ratón

- Clase de evento: `MouseEvent`.
- Existen dos interfaces escuchadores de eventos de ratón:
 - `MouseListener`: todo menos movimiento.
 - `MouseMotionListener`: eventos de movimiento.
- Métodos del interfaz `MouseListener`:

- `public abstract void mousePressed(MouseEvent event);`

- `public abstract void mouseReleased(MouseEvent event);`
- `public abstract void mouseClicked(MouseEvent event);`
- `public abstract void mouseEntered(MouseEvent event);`
- `public abstract void mouseExited(MouseEvent event);`

- Métodos del interfaz `MouseMotionListener`:

- `public abstract void mouseMoved(MouseEvent event);`
- `public abstract void mouseDragged(MouseEvent event);`

- Métodos de `MouseEvent`:

- `public int getClickCount();`
- `public synchronized Point getPoint();`
- `public int getX();`
- `public int getY();`

La clase `InputEvent`

- De esta clase se derivan las clases `KeyEvent` y `MouseEvent`. Aquí se define el manejo de los modificadores de teclas y de ratón. Los modificadores son las teclas de control, mayúsculas, alternativa, meta y los botones del ratón:
- Para obtener los modificadores involucrados en un evento usar el método:

- `public int getModifiers();`

Devuelve un bitmap con los modificadores activos.
Los códigos para crear el bitmap son:

Codigo
<code>InputEvent.ALT_MASK</code>
<code>InputEvent.CTRL_MASK</code>
<code>InputEvent.META_MASK</code>

InputEvent.SHIFT_MASK
InputEvent.BUTTON1_MASK
InputEvent.BUTTON2_MASK
InputEvent.BUTTON3_MASK

Ejemplo:

```
InputEvent evt;  
if ((evt.getModifiers() & InputEvent.ALT_MASK) != 0) {  
    // la tecla ALT estaba pulsada.  
}
```

¿Que es un frame?

Se podría decir que un frame es una clase utilizada en Swing(Es un biblioteca de clases).Esto frame nos sirven para generar ventanas de usuario ,en las cuales se le pueden añadir diferentes objetos (JButton, JTable, etc)con estos objetos el usuario podrá interactuar con el programa

Características de un frame

- Nacen invisibles. Necesita el método **setVisible()** para hacerlos visibles en la pantalla de la pc.
- Nacen con un tamaño predeterminado por java. Se necesita el método **setSize()** para darle un ajuste al frame.
- Se debe indicar que debe hacer el programa cuando se cierre el frame.

Herramientas de JFrame

- **setSize(ancho,altura):** Este tipo de método nos permite cambiar el ancho y altura .
 - setSize(500, 300);
- **setLocation(x,y):** El método nos permite mover o cambiar de posición de un componente de la ventana.Su esquina superior izquierda se especifica
 - setLocation(500, 300);
- **setBounds(x,y,ancho,altura):** Este método es mas completo que **setSize()** y **setlocation()**: en la cual une los dos para ser uno solo
 - setBounds(500, 300, 250, 250);
- **setTitle(String title):**Pone el nombre para el **JFrame** mediante la especificación de tipo **String**
 - setTitle(«Primer programa con JFrame»);
- **setResizable(boolean):**Nos permite dejar si el frame se puede modificar o no en tiempo de ejecución
 - setResizable(false);
- **setExtendedState(JFrame.MAXIMIZED_BOTH):** Tiene como funcionalidad poner el JFrame en tamaño completo de la pantalla
 - setExtendedState(JFrame.MAXIMIZED_BOTH);

- **setDefaultCloseOperation(miFrame.EXIT_ON_CLOSE):** Método que nos permite cerrar todas las ejecuciones que se este dando dentro de un **JFrame**
 - setDefaultCloseOperation(miFrame.EXIT_ON_CLOSE);
- **setVisible(boolean):** para hacerlos visibles en la pantalla de la pc.
 - setVisible(true);