Modelo de Eventos de Java Event

¿Que son?

- Representan la actividad entre el sistema, los programas y los usuarios.

- El modelo de gestión de eventos de AWT está basado en la Herencia. Para que un programa capture eventos de un interfaz, los Componentes deben ser subclases del interfaz y sobreescribir los métodos *action()* y *handleEvent()*.

- Cuando uno de los dos métodos anteriores devuelve true, el evento ya no es procesado más allá, en caso contrario, el evento se propaga a través de la jerarquía de componentes del interfaz hasta que el evento sea tratado o alcance la raíz de la jerarquía. El resultado de este modelo es que los programas tienen dos elecciones para estructurar su código de manejo de eventos:

- Cada componente individual puede hacerse subclase para manejar específicamente un conjunto de eventos

- Todos los eventos para una jerarquía completa (o subconjunto de ella) pueden ser manejados por un contenedor determinado

- En este modelo de Herencia, no hay posibilidad de filtrar eventos. Los eventos son recibidos por los Componentes, independientemente de que los manejen o no. Este es un problema general de rendimiento, especialmente con eventos que se producen con mucha frecuencia, como son los eventos de ratón. Con el nuevo modelo, todos los sistemas debería ver incrementado su rendimiento, especialmente los sistemas basados en *Solaris*.

Modelo de Funcionamiento

- 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.
  Debe importarse `java.awt.event.*`
- Los objetos escuchadores deben registrarse en los generadores para que estos les envien 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.

- Ejemplo:
- `boton.addActionListener(objeto_escuchador);`
- `boton.removeActionListener(objeto_escuchador);`
- 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 vacio.
  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 `KeyListerner`, 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)`

- El modelo de eventos de java 1.1 soporta también el modelo 1.0, aunque este dejará de soportarse en futuras versiones.
  No deben mezclarse ambos modelos en una misma aplicación.

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.
  ```
  ```