

# Eventos

---

GESTIÓN DE EVENTOS EN JAVA

# Eventos

---

- Todo sistema operativo que utiliza interfaces gráficas de usuario debe estar constantemente monitorizando el entorno para capturar y tratar los eventos que se producen
- El sistema operativo informa de estos eventos a los programas que se están ejecutando y entonces cada programa decide que hace para dar respuesta a esos eventos
- Los eventos pueden estar producidos por el sistema o por el usuario

# Eventos en Java

---

- Cada vez que el usuario realiza una determinada acción sobre una aplicación Java se produce un evento que el sistema operativo transmite a Java.
- Java crea un objeto de una determinada clase de evento, y este evento es transmitido a un determinado método para que lo gestione.
- El modelo de eventos de Java está basado en *delegación* (la responsabilidad de gestionar un evento que ocurre en un objeto *-source-* la tiene otro objeto *-listener-*)

# Elementos del modelo de eventos en Java

---

- Fuentes de Eventos (*event sources*)
- Receptores de Eventos (*event listener*)
- Adaptadores (*adapter classes*)

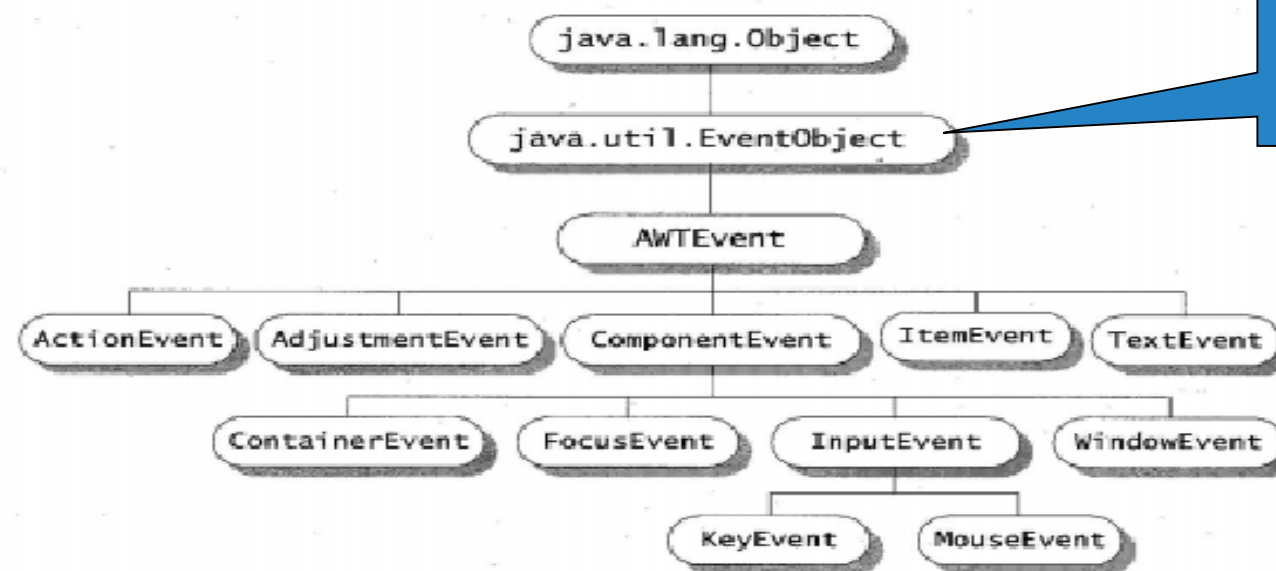
# Fuentes de eventos

---

- Los **fuentes de eventos** (*event sources*) son objetos que detectan eventos y notifican a los receptores que se han producido dichos eventos.
- Ejemplos de fuentes :
  - **Botón** sobre el que se pulsa
  - **Campo de texto** que pierde el foco
  - **Area de texto** sobre la que se presiona una tecla
  - **Ventana** que se cierra
  - ...

# Clase del evento

- Cuando se produce un evento, dicho evento es representado como un objeto de una clase que descende de *EventObject*.
- Los eventos están organizados en jerarquías de clases de eventos: \*



Clase EventObject tiene un método getSource() que devuelve el componente que produjo el evento

\* En este gráfico se representan algunos de los eventos más representativos

# Receptores del evento

---

- Cuando tiene lugar un evento, el objeto fuente necesita llamar a un **objeto receptor**
- Los **receptores de eventos** (*event listener*) son objetos instanciados de una clase que implementa un interface específico, denominado *interface listener*.
- Ejemplos de *interfaces listener* son: *ActionListener*, *FocusListener*, *ItemListener*, *KeyListener*, *MouseListener*, *WindowListener*, ...

# Registro

---

- Cuando ya se dispone del objeto fuente y del objeto receptor es necesario asociarlos, es decir, hay que registrar el objeto receptor de eventos con el objeto fuente de los mismos:

`objetoFuenteEvento.addEventListener(objetoReceptorEvento)`



# Construcción de la clase receptora

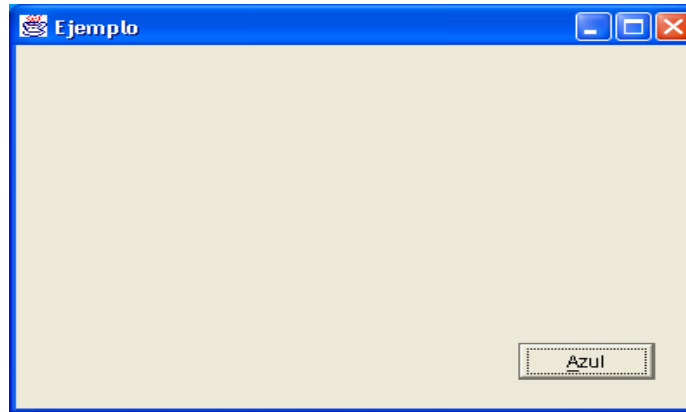
---

- Por lo general, la clase receptora necesita acceder a miembros de la misma clase que contiene el objeto fuente (ejemplo: *VentanaPrincipal* en nuestras prácticas). Para conseguirlo existen dos posibilidades:
  - Crear la clase receptora como una clase interna de la clase *VentanaPrincipal*. Una clase interna es una clase definida dentro de otra clase y que tiene acceso a los miembros de la clase que la encierra.
  - Crear la clase receptora como una clase externa a la *VentanaPrincipal* y pasar una referencia a dicha clase en el constructor de la clase receptora con lo que ya podrá acceder a los miembros de la clase *VentanaPrincipal*

# Ejemplo 1. ActionEvent

---

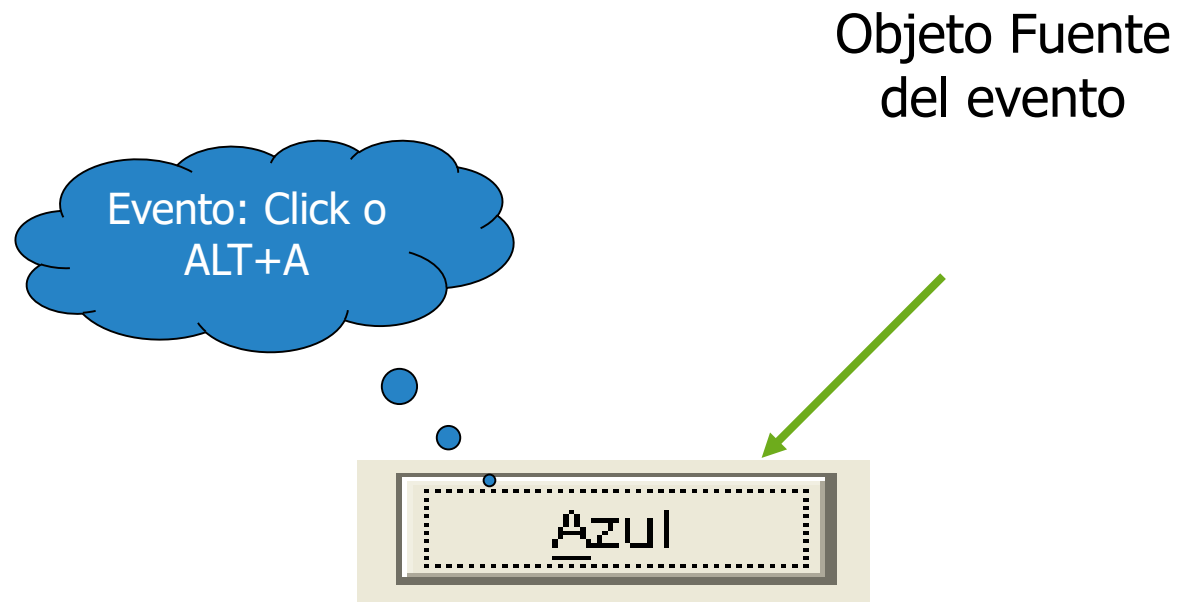
Cambiar a color azul el fondo del panel de contenidos pulsando un botón



# Objeto fuente del evento

---

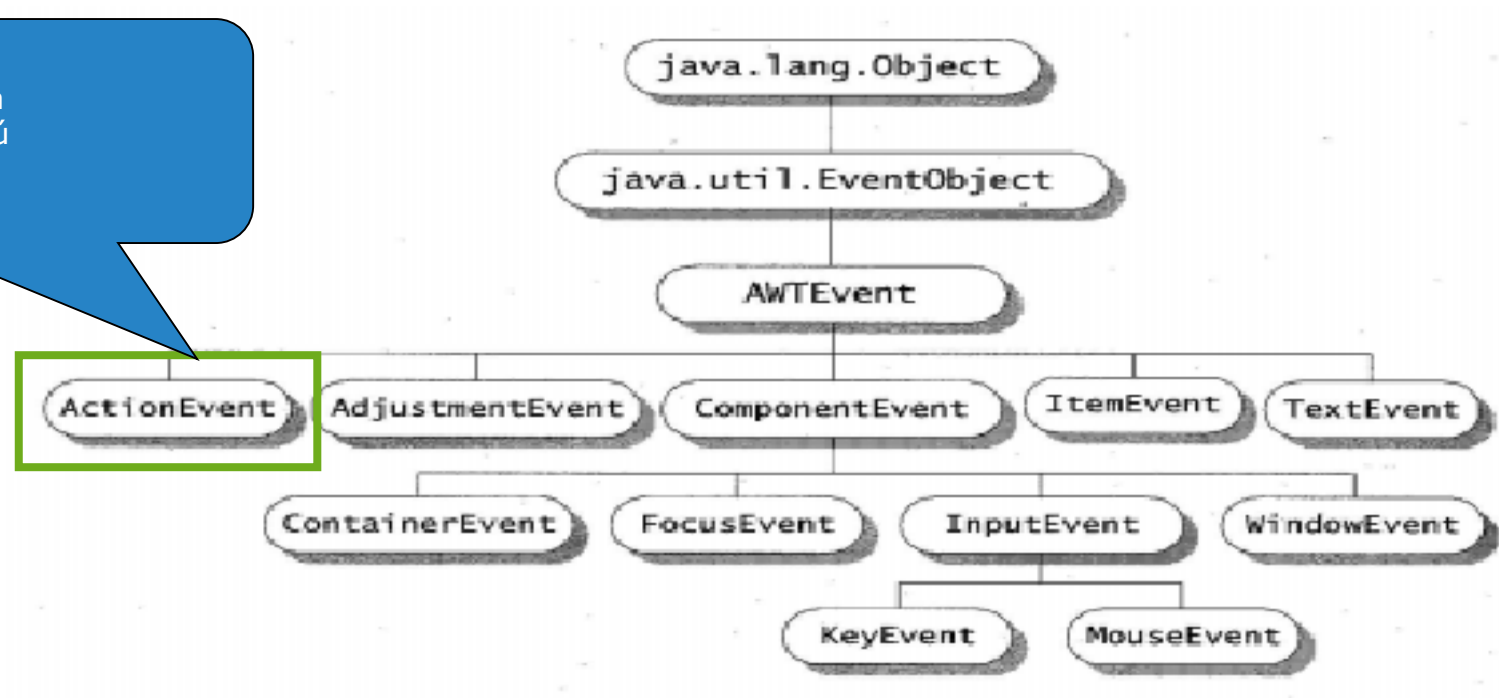
- Es el objeto que detecta el evento y notifica a los receptores que se ha producido dicho evento
- En el ejemplo, sería el componente *btAzul*



# Clase del evento

En el caso del ejemplo que nos ocupa, el evento que se genera al presionar un botón es de tipo *ActionEvent*

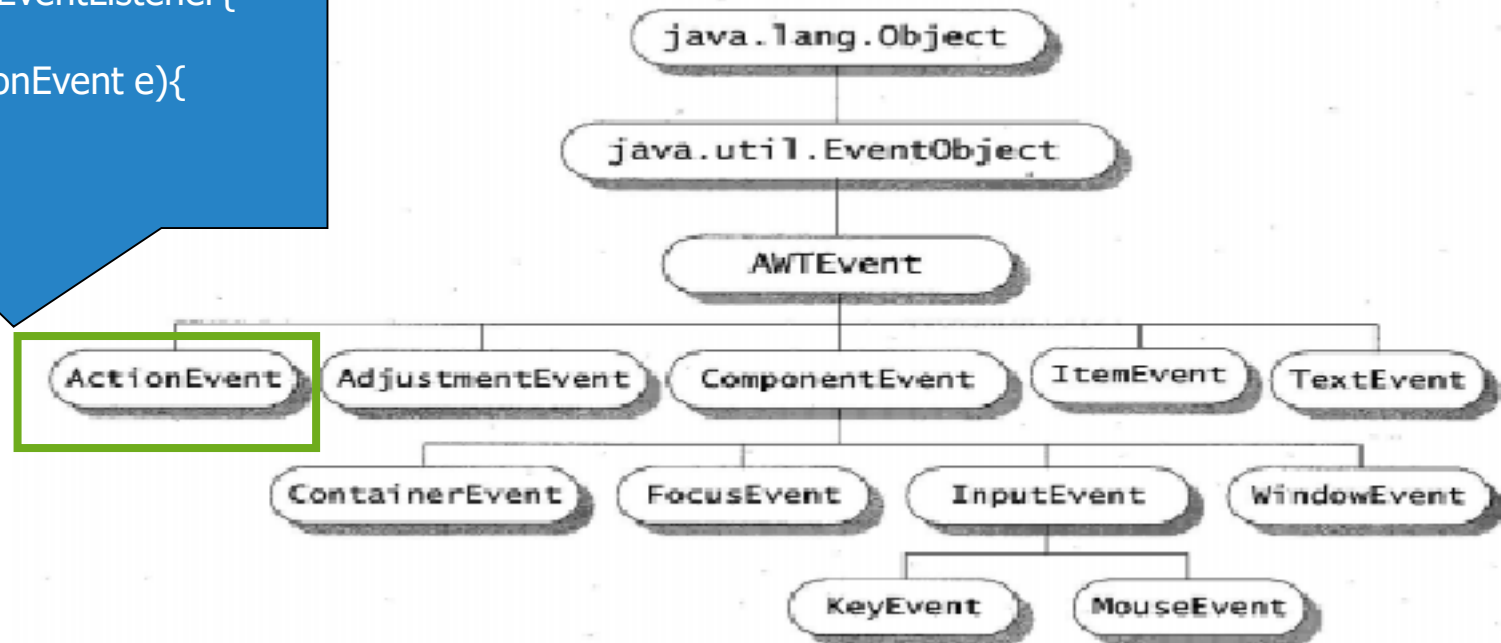
- Pulsar un botón
- Seleccionar un elemento de una lista
- Seleccionar un elemento de un menú
- Pulsar [Enter] en un campo de texto
- ...



# Clase receptora del evento

La clase receptora para un evento *ActionEvent* debe ser una clase que implemente el interface *ActionListener* que únicamente contiene un método: *actionPerformed*.

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



# Construcción de la clase receptora

---

- En este caso, la clase receptora, que vamos a denominar "ProcesaAccion", se construye como una clase interna de la clase VentanaPrincipal lo que permitirá el acceso a miembros de dicha clase.
- El método que hay que redefinir es el único de la Interfaz ActionListener: `public void actionPerformed(ActionEvent e)` que se encargará de invocar la método de la clase VentanaPrincipal con "lo que queremos que pase" cuando el usuario haga clic en el botón.

```
public class VentanaPrincipal extends JFrame {  
    ...  
    private void cambiaColorPanel() {  
        getPanelPrincipal().setBackground(Color.blue);  
    }  
  
    class ProcesaAccion implements ActionListener { ← Clase receptora  
        public void actionPerformed(ActionEvent e) {  
            cambiaColorPanel();  
        }  
    }  
    ...  
}
```

# Creación del objeto receptor del evento

---

```
class VentanaPrincipal extends JFrame{
```

```
    private ProcesaAccion pA;
```

```
    ...
```

```
    private void cambiaColorPanel(){
```

```
        panelPrincipal.setBackground(Color.blue);
```

```
    } ...
```

```
class ProcesaAccion implements ActionListener {
```

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

```
        cambiaColorPanel();
```

```
    }
```

```
    ...
```

```
public VentanaPrincipal(){
```

```
    pA = new ProcesaAccion();
```

← Objeto receptor

```
    ...}
```

```
}
```

# Registro

---

- Finalmente, es necesario registrar el objeto receptor de eventos con el objeto fuente de los mismos. Lo hacemos en el método `get` del botón, una vez esté creado:

```
btAzul.addActionListener(pA)
```

- Una vez registrado, cada vez que se produce el evento (se pulsa el botón) automáticamente se llama al método apropiado (*actionPerformed*), pasándole como parámetro un objeto que es una instancia de `ActionEvent`.



# Registro (II)

---

```
class VentanaPrincipal extends JFrame{
    private ProcesaAccion pA;
    ...
    void cambiaColorPanel(){
        panelPrincipal.setBackground(Color.blue);
    } ...
    class ProcesaAccion implements ActionListener {
        public void actionPerformed(ActionEvent e){
            cambiaColorPanel();
        }
    }
}
```

```
private JButton getBtAzul() {
    if (btAzul == null) {
        btAzul = new JButton();
        ...
        btAzul.addActionListener (pA);
    }
    return btAzul }

public VentanaPrincipal(){
    pA = new ProcesaAccion();
    ...}
}
```

← Registro

# Código generado por WindowBuilder

```
public class VentanaPrincipal extends JFrame {
```

```
private JPanel contentPane;
```

```
private JButton btAzul;
```

```
...
```

```
private JButton getBtAzul() {
```

```
    if (btAzul == null) {
```

```
        btAzul = new JButton();
```

```
        btAzul.addActionListener(new java.awt.event.ActionListener() {  
            public void actionPerformed(java.awt.event.ActionEvent e) {  
                cambiaColorPanel();  
            }  
        });
```

```
    };
```

```
}
```

```
return btAzul;
```

```
}
```

```
...
```

Objeto Fuente

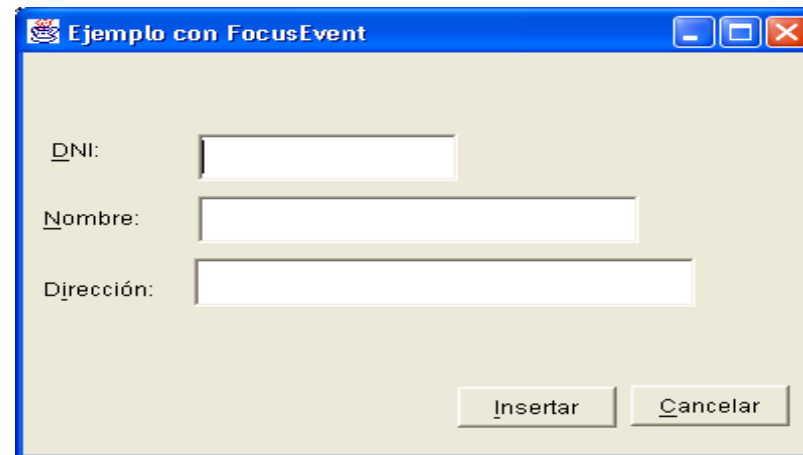
Registro

Objeto Receptor creado a partir de una clase interna sin nombre que implementa la interface ActionListener

## Ejemplo 2. FocusEvent

---

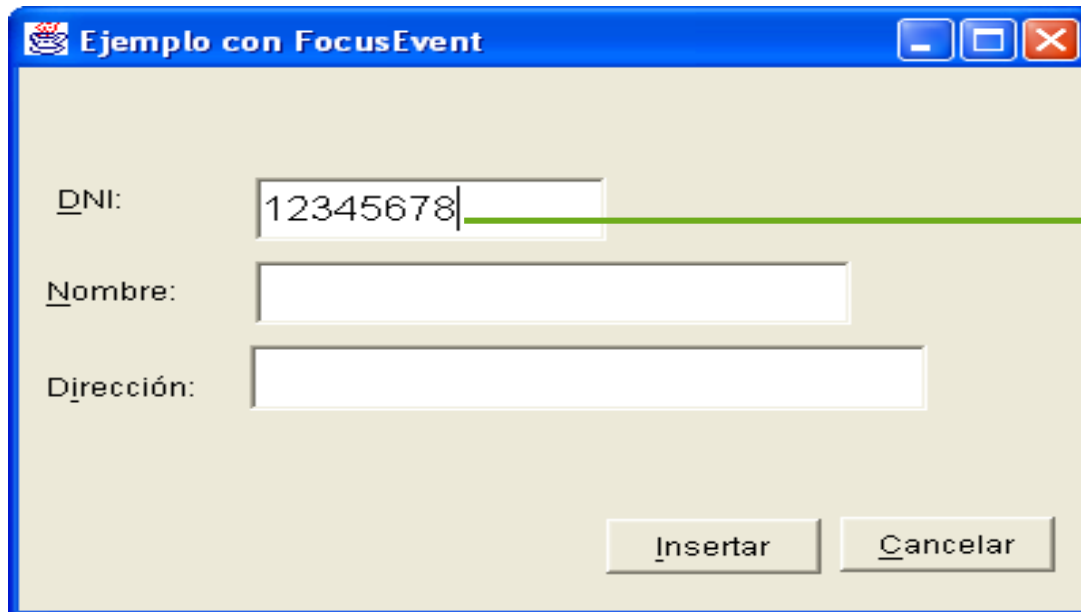
Validar la longitud de un campo de texto (JTextField) cuando dicho componente pierda el foco.



The screenshot shows a Java Swing window titled "Ejemplo con FocusEvent". Inside the window, there are three text input fields (JTextField) arranged vertically. The first field is labeled "DNI:", the second "Nombre:", and the third "Dirección:". Below the fields, there are two buttons: "Insertar" and "Cancelar". The window has a standard Windows-style title bar with minimize, maximize, and close buttons.

# Fuente del evento

---



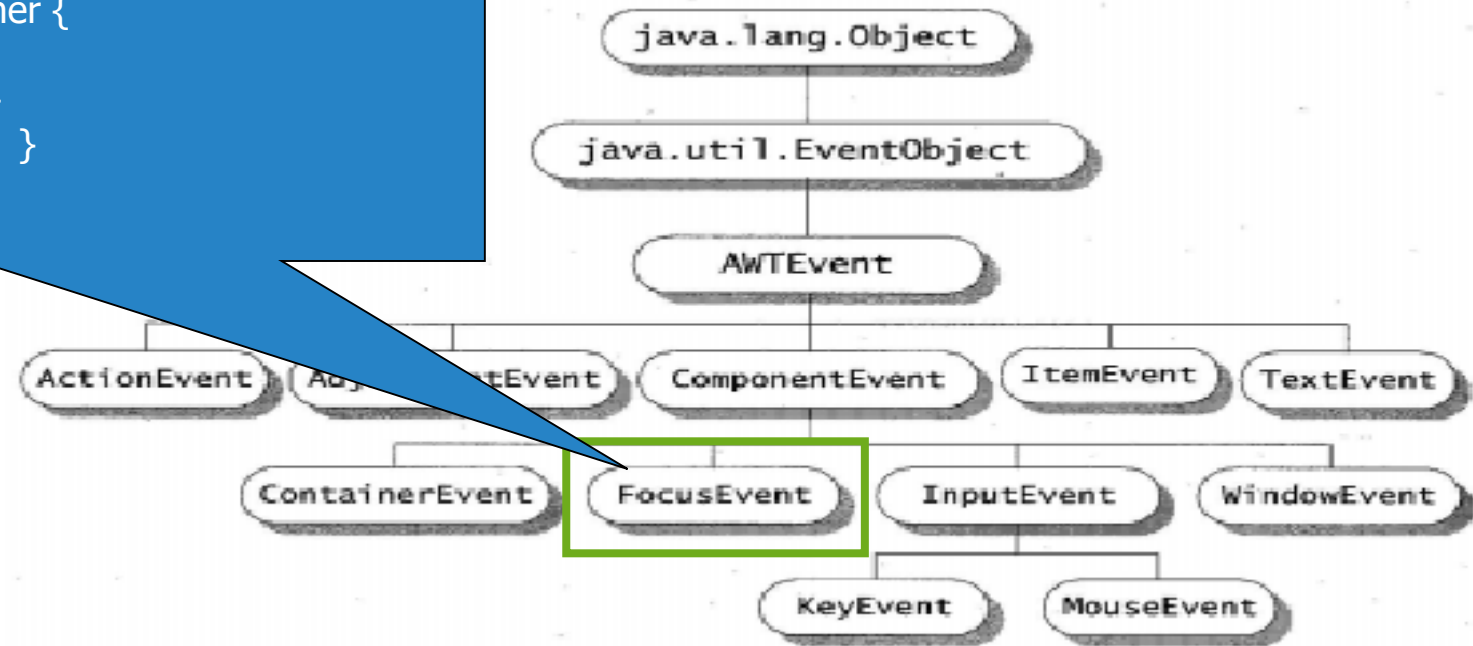
The screenshot shows a Java Swing window titled "Ejemplo con FocusEvent". Inside the window, there is a form with three text input fields. The first field, labeled "DNI:", contains the text "12345678". The second field, labeled "Nombre:", is empty. The third field, labeled "Dirección:", is also empty. At the bottom of the window, there are two buttons: "Insertar" and "Cancelar". A green arrow originates from the right side of the "DNI:" text field and points towards the text "Fuente del evento (txDNI)" located to the right of the window.

Fuente del  
evento  
(txDNI)

# Clase receptora del evento

Debe ser una clase que implemente el interface *FocusListener* que contiene dos métodos: *focusLost* y *focusGained*.

```
Interface FocusListener extends EventListener {  
    public void focusLost (FocusEvent e ) { }  
    public void focusGained (FocusEvent e ) { }  
}
```



# Clase y objeto receptor del evento (II)

---

```
class VentanaPrincipal extends JFrame{
    private ProcesaFoco pF ;
    ...

    public VentanaPrincipal() {
        pF = new ProcesaFoco();
        ...
    }
    class ProcesaFoco implements FocusListener {
        public void focusGained (FocusEvent e) { } // se deja vacío porque no se va a utilizar
        public void focusLost (FocusEvent e) {
            compruebaLongitud(); }
    }...
    private void compruebaLongitud() {
        if (txDNI.getText().length() !=8 )
            //acción a llevar a cabo
        .. .}
    ...}
}
```

# Registro

---

....

```
private JTextField getTxDNI() {  
    if (txDNI == null) {  
        txDNI = new JTextField();  
        ....  
        txDNI.addFocusListener(pF);  
    }  
}  
...  
...
```

# Adaptadores o Clases Adaptadoras

---

- Los adaptadores (adapter classes) tienen como objetivo evitar la tarea de tener que implementar todos los métodos de las *interfaces listener*.
- En java existe una clase adaptadora por cada *interface listener* **que tenga más de un método**: `MouseAdapter`, `WindowAdapter`, `KeyAdapter`, `MouseMotionAdapter`, `FocusAdapter`, etc.
- Las clases adaptadoras implementan todos los *métodos de la interface correspondiente como vacíos*
- En el ejemplo que nos ocupa, *FocuListener* tiene dos métodos: *focusGained* y *focusLost*, pero realmente sólo queremos dar implementación a uno de ellos: *focusLost*. Para ello acudimos a la clase adaptadora *FocusAdapter*.



# Adaptadores (modificación del ejemplo 2)

```
class VentanaPrincipal extends JFrame{
```

```
    private ProcesaFoco pF;
```

```
    ...
```

```
    private void compruebaLongitud(){
```

```
        if (txDNI.getText().length()!=8)
```

```
        ...}
```

```
class ProcesaFoco extends FocusAdapter {
```

```
    public void focusLost (FocusEvent e) {
```

```
        compruebaLongitud(); }
```

```
    }...
```

```
    public VentanaPrincipal() {
```

```
        pF = new ProcesaFoco();
```

→ Sólo se proporciona implementación al método *focusLost* ya que es el que nos interesa en este ejemplo

# Componentes que responden al mismo evento

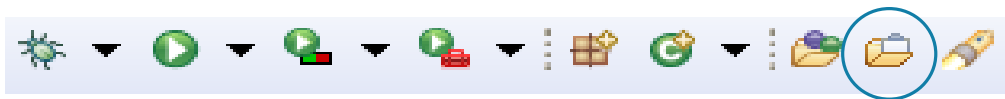
---

- Si varios componentes responden ante el mismo evento realizando la misma acción, sólo es necesario crear **un objeto receptor** y realizar el registro de todos los objetos fuente del evento con este único objeto receptor.
  - Ejemplo: botón “Cancelar” y opción “Nuevo” en una ventana que gestione un pedido
- Por lo tanto, para la gestión de estos dos eventos (clic en “Cancelar” y clic en “Nuevo”) se incorporaría en el código:
  - **Una** clase receptora del evento
  - **Un** objeto receptor
  - **Dos** registros

## Ejemplo 3. MouseEvent

---

- Pintar y despintar el borde de varios botones de una barra de herramientas cuando se acerque y se aleje el puntero del ratón (eventos *MouseEntered* y *MouseExited*).



- Problema: ¿Cómo sabemos qué botón concreto de todos los registrados en el mismo objeto receptor disparó el evento?
- Solución: hacemos uso del objeto evento (*e*, en el siguiente ejemplo) para averiguar el objeto fuente concreto en el que se produjo el evento.

## Código para el tratamiento de los eventos de ratón

---

```
public class VentanaPrincipal extends JFrame {  
    private ProcesaBorde pB;  
    ...  
    class ProcesaBorde extends MouseAdapter{  
        public void mouseExited(MouseEvent e) {  
            pintaBorde(e, false);  
        }  
        public void mouseEntered(MouseEvent e) {  
            pintaBorde(e, true);  
        }  
    }  
    private void pintaBorde(MouseEvent e, boolean b) {  
        JButton boton= (JButton)e.getSource();  
        boton.setBorderPainted(b);  
    }  
}
```

Se obtiene el botón concreto que provocó el evento

```
private JButton getBoton1() {  
    if (boton1 == null) {  
        boton1 = new JButton();  
        boton1.setBorderPainted(false);  
        boton1.addMouseListener(pB);  
    }  
    return boton1;  
}  
  
private JButton getBoton2() {  
    if (boton2 == null) {  
        boton2 = new JButton();  
        boton2.setBorderPainted(false);  
        boton2.addMouseListener(pB);  
    }  
    return boton2;  
}  
  
public VentanaPrincipal() {  
    pB = new ProcesaBorde();  
    ...}  
}
```

# consume

---

- El método consume invocado sobre un evento evita que dicho evento sea procesado o, dicho de otro modo, que no sea tratado por el objeto receptor del evento.
- Ejemplo: Queremos evitar que en un campo de texto (txtDNI) se introduzcan caracteres alfabéticos:

```
class ProcesaTecla extends KeyAdapter {  
    public void keyTyped(KeyEvent e) {  
        char teclaPulsada = e.getKeyChar();  
        if (Character.isAlphabetic(teclaPulsada))  
            e.consume();  
    }  
}
```

```
private JTextField getTextDNI() {  
    if (txtDNI == null) {  
        txtDNI = new JTextField();  
        txtDNI.addKeyListener(pT); (*)  
        ...  
    }  
    return txtDNI;  
}
```

(\*) pT = new ProcesaTecla();

# Deshacer el registro entre fuente y receptor

---

- Para que un objeto fuente deje de estar atento a un determinado evento, se invoca al método `removeEventListener` sobre dicho objeto fuente.
- Continuando con el ejemplo anterior: queremos que en función del valor de un checkbox se evite o no que en el campo de texto (txtDNI) se introduzcan caracteres alfabéticos:

```
private JTextField getTxtDNI() {  
    if (txtDNI == null) {  
        txtDNI = new JTextField();  
        txtDNI.addKeyListener(pT); (*)  
        ...  
    }  
    return txtDNI;  
}
```

```
(*) pT = new ProcesaTecla();
```

```
class ProcesaAccionCheck implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        JCheckBox chkFuente = (JCheckBox)e.getSource();  
        if (chkFuente.isSelected())  
            getTxtDNI().addKeyListener(pT);  
        else  
            getTxtDNI().removeKeyListener(pT);  
    }  
}
```

# Eventos – Resumen (I)

Evento	Interfaz Oyente	Clase Adaptadora	Métodos de la interfaz	Cómo se producen
ActionEvent	ActionListener		actionPerformed(ActionEvent)	<ul style="list-style-type: none"> <li>■ Pulsar un botón</li> <li>■ Seleccionar un elemento de una lista</li> <li>■ Seleccionar un elemento de menú</li> <li>■ Pulsar enter en un campo de texto</li> </ul>
ComponentEvent	ComponentListener	ComponentAdapter	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)	<ul style="list-style-type: none"> <li>■ Cambiar de tamaño, mover, mostrar u ocultar un componente.</li> </ul>
ContainerEvent	ContainerListener	ContainerAdapter	componentAdded (ContainerEvent) componentRemoved (ContainerEvent)	<ul style="list-style-type: none"> <li>■ Añadir o eliminar algún componente</li> </ul>
FocusEvent	FocusListener	FocusAdapter	focusGained (FocusEvent) focusLost (FocusEvent )	<ul style="list-style-type: none"> <li>■ Cuando un componente recibe o pierde el foco</li> </ul>

# Eventos – Resumen (II)

Evento	Interfaz Oyente	Clase Adaptadora	Métodos de la interfaz	Cómo se producen
KeyEvent	KeyListener	KeyAdapter	keyPressed (KeyEvent) keyReleased (KeyEvent) keyTyped (KeyEvent)	■ Cuando se ha pulsado o liberado una tecla
MouseEvent	MouseListener	MouseAdapter	mouseClicked (MouseEvent) mouseEntered (MouseEvent) mouseExited (MouseEvent) mousePressed (MouseEvent) mouseReleased (MouseEvent)	■ Cuando se realizan ciertas operaciones con el ratón
MouseEvent	MouseMotionListener	MouseMotionAdapter	mouseDragged ((MouseEvent) mouseMoved (MouseEvent)	■ Cuando se realizan ciertas operaciones con el ratón



# Eventos – Resumen (III)

Evento	Interfaz Oyente	Clase Adaptadora	Métodos de la interfaz	Cómo se producen
WindowEvent	WindowListener	WindowAdapter	windowOpened (WindowEvent) windowClosing (WindowEvent) windowClosed (WindowEvent) windowActivated (WindowEvent) windowDeactivated (WindowEvent) windowIconified (WindowEvent) windowDeiconified (WindowEvent)	■ Cuando se realiza una determinada operación sobre una ventana
ItemEvent	ItemListener		itemStateChanged (ItemEvent)	■ Cuando se selecciona una casilla de verificación o una lista de ítems
TextEvent	TextListener		textValueChanged (TextEvent)	■ Cuando se cambia algo en un cuadro o área de texto
...	...	...	...	...