# Linguaxes de programación orientados a obxetos. Java

# TEMA 3: EVENTOS.

## INDICE

# Tema 3. Eventos.

## 1.Introducción

Java es un lenguaje orientado a objetos, por lo que los objetos (las clases) son los elementos más importantes en el diseño y desarrollo de una aplicación. También podemos afirmar que Java es un lenguaje orientado a eventos, puesto que nos proporciona todos los elementos necesarios para definirlos y utilizarlos; los eventos son objetos fundamentales en el desarrollo de la mayor parte de las aplicaciones.

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 (Swing), éstos generan *eventos*. La siguiente tabla muestra los *eventos* que cada tipo de componente puede generar y cuándo los genera.

| Tipo de 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. En Java hay otros eventos aparte de los mencionados en la tabla. Los eventos en general se representan como objetos de subclases de la clase java.util.EventObject.

La siguiente tabla presenta estas interfaces con sus métodos.

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

Todos los métodos de estas interfaces son de tipo public void.

# 2.Mecanismo de programación de eventos.

Supongamos que deseamos que en un Componente que estamos desarrollando (típicamante 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:

- Declarar que implementa la interfaz correspondiente.
- Implementar los métodos de la interfaz
- 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...

A continuación se detalla cómo se gestionan los eventos según el modelo de Java.Cada objeto que puede recibir un evento (event source), registra uno o más objetos para que los gestione. Esto se hace con un método que tiene la forma:

```
eventSourceObjecto.addEventListener(eventListenerObject);
```

donde eventSourceObject es el objeto en el que se produce el evento y eventListenerObject es el objeto que deberá gestionar los eventos.

La relación entre ambos se establece a través de una interfaz Listener que la clase del eventListenerObject debe implementar. Esta inferface proporciona la declaración de los métodos que serán llamados cuando se produzca el evento. La interfaz a implementar depende del tipo de evento. Cuando se implementa una interface es obligatorio redefinir todos sus métodos.

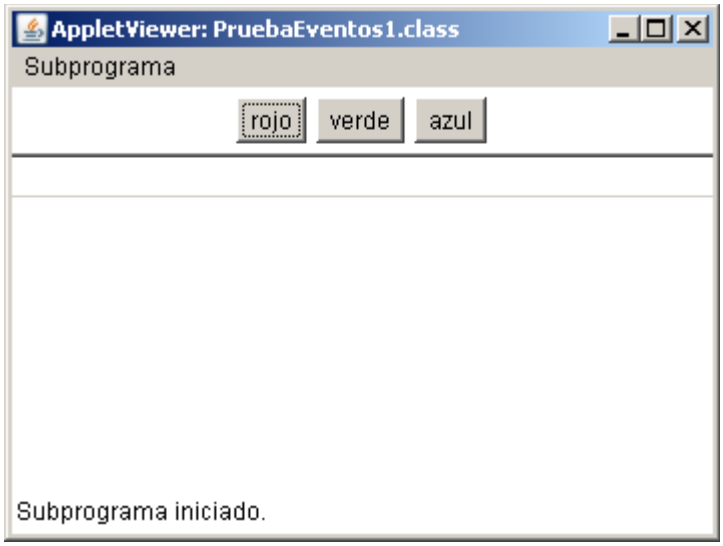
La interface a implementar depende del tipo de evento. La Tabla relaciona los distintos tipos de eventos, con la interface que se debe implementar para gestionarlos. Se indican también los métodos declarados en cada interface. Es importante observar la correspondencia entre eventos e interfaces Listener. Cada evento tiene su interface, excepto el ratón que tiene dos interfaces MouseListener y MouseMotionListener. La razón de esta duplicidad de interfaces se encuentra en la peculiaridad de los eventos que se producen cuando el ratón se mueve. Estos eventos, que se producen con muchísima más frecuencia que los simples clicks, por razones de eficiencia son gestionados por una interface especial: MouseMotionListener.

El nombre de la interface coincide además con el nombre del evento sustituyendo la palabra Event por Listener.

Una vez registrado el objeto que gestionará el evento, perteneciente a una clase que implemente la correspondiente interface Listener, se deben definir los métodos de dicha interface. Siempre hay que definir todos los métodos de la interface, aunque algunos de dichos métodos puedan estar “vacíos”.

| Evento | Interface Listener | Métodos de Listener |
|---|---|---|
| ActionEvent | ActionListener | actionPerformed() |
| AdjustmentEvent | AdjustementListener | adjustementValueChanged() |
| ComponentEvent | ComponentListener | componentHidden(), componentMoved(), componentResized(), componentShown() |
| ContainerEvent | ContainerListener | componentAdded(), componentRemoved() |
| FocusEvent | FocusListener | focusGained(), focusLost() |
| ItemEvent | ItemListener | itemStateChanged() |
| KeyEvent | KeyListener | keyPressed(), keyReleased(), keyTyped() |
| MouseEvent | MouseListener | mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased() |
| | MouseMotionListener | mouseDragged(), mouseMoved() |
| TextEvent | TextListener | textValueChanged() |
| WindowEvent | WindowListener | WindowActivated(), windowDeactivated(), windowClosing(), windowClosed(), windowIconified(), windowDeiconified(), windowOpened() |

Vamos a ver el funcionamiento a través de un pequeño ejemplo el cual consta sólo de tres botones con títulos rojo, verde y azul y un campo de texto en el que se escribe el evento generado al hacer clic sobre cualquiera de los botones. Cuando se pulsa uno de los botones el color de fondo del applet cambia al que indica el botón.




```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class buttonsApplet extends Applet implements ActionListener {
        Button b_rojo, b_verde, b_azul;
        TextField tf;
        public void init () {
                add(b_rojo=new Button("rojo"));
                add(b_verde=new Button("verde"));
                add(b_azul=new Button("azul"));
                add(tf=new TextField(80));
                tf.setBackground(Color.white);
                b_rojo.addActionListener(this);
                b_verde.addActionListener(this);
                b_azul.addActionListener(this);
        }
/* ---------------------- Action Listener ------------------- */
        public void actionPerformed(ActionEvent e) {
                tf.setText(e.paramString());
                if (e.getSource()==b_rojo) {
                        setBackground(Color.red);
```

```
                } else if (e.getSource()==b_verde) {
                        setBackground(Color.green);
                } else if (e.getSource()==b_azul) {
                        setBackground(Color.blue);
                }
        }
}
```

Para poder distinguir cual es el botón que se ha pulsado, se hace utilizando e.getSource() que devuelve precisamente el objeto que produjo el evento. De ahí que para reconocer el botón pulsado se hagan comparaciones de cada botón con e.getSource(). Otra manera de distinguir los botones es comparando e.getActionCommand() con las etiquetas de los botones (esto debe hacerse usando el método equals de String y no la simple igualdad == entre objetos).

# 3. Clases Adapter.

Java proporciona ayudas para definir los métodos declarados en las interfaces Listener. Una de estas ayudas son las clases Adapter, que existen para cada una de las interfaces Listener que tienen más de un método. Su nombre se construye a partir del nombre de la interface, sustituyendo la palabra "Listener" por "Adapter".

Hay 7 clases Adapter: ComponentAdapter, ContainerAdapter, FocusAdapter, KeyAdapter, MouseAdapter, MouseMotionAdapter y WindowAdapter. Las clases Adapter derivan de Object, y son clases predefinidas que contienen definiciones vacías para todos los métodos de la interface. Para crear un objeto que responda al evento, en vez de crear una clase que implemente la interface Listener, basta crear una clase que derive de la clase Adapter.

Las clases Adapter derivan de Object, y son clases predefinidas que contienen definiciones vacías para todos los métodos de la interface. Para crear un objeto que responda al evento, en vez de crear una clase que implemente la interface Listener, basta crear una clase que derive de la clase Adapter correspondiente, y redefina sólo los métodos de interés.

Veamos el funcionamiento a través de otro ejemplo:

```
import java.awt.*;
import java.awt.event.*;
class VentanaCerrable2 extends Frame {
 // constructores
public VentanaCerrable2() { super(); }
public VentanaCerrable2(String title) {
   super(title);
```

```
        setSize(500,500);
        CerrarVentana cv = new CerrarVentana();
        this.addWindowListener(cv);
    }
    } // fin de la clase VentanaCerrable2
    // definición de la clase CerrarVentana
    class CerrarVentana extends WindowAdapter {
    void windowClosing(WindowEvent we) { System.exit(0); }}
```

Se define una clase auxiliar que deriva de la clase WindowAdapter. Dicha clase hereda definiciones vacías de todos los métodos de la interface WindowListener. Lo único que tiene que hacer es redefinir el único método que se necesita para cerrar las ventanas. El constructor de la clase VentanaCerrable crea un objeto de la clase CerrarVentana en la  y lo registra como event listener. La palabra this es opcional: si no se incluye, se supone que el event source es el objeto de la clase en la que se produce el evento, en este caso la propia ventana.

# 4.Eventos de ratón y de movimiento de ratón.

Los eventos de ratón se capturan haciendo uso de los siguientes interfaces (oadaptadores) y eventos:

| Evento | Interface | Adaptador |
|---|---|---|
| MouseEvent | MouseListener | MouseAdapter |
| MouseEvent | MouseMotionListener | MouseMotionAdapter |
| MouseWheelEvent | MouseWheelListener | MouseWheelListener |

Los dos primeros interfaces y adaptadores se usan mucho más a menudo que el tercero, por lo que centraremos las explicaciones en ellos. Como se puede observar en la tabla anterior, el evento MouseEvent se utiliza en los métodos pertenecientes a los interfaces MouseListener y MouseMotionListener. El interfaz Mouse WheelListener sólo consta de un método, por lo que no resulta interesante incorporarle un adaptador.

Se produce un ***MouseEvent*** cada vez que el cursor movido por el ratón entra o sale de un componente visible en la pantalla, al clicar, o cuando se pulsa o se suelta un botón del ratón. Los métodos de la interface ***MouseListener*** se relacionan con estas acciones, y son los siguientes ***mouseClicked()***, ***mouseEntered()***, ***mouseExited()***, ***mousePressed()*** y ***mouseReleased()***.

La clase MouseEvent define una serie de constantes int que permiten identificar lostipos de eventos que se han producido: MOUSE_CLICKED, MOUSE_PRESSED, MOU-

SE_RELEASED, MOUSE_MOVED, MOUSE_ENTERED, MOUSE_EXITED, MOUSE_DRAGGED.

Además, el método ***Component getComponent()***, heredado de ***ComponentEvent***, devuelve el componente sobre el que se ha producido el evento.

La interface ***MouseMotionListener***, implementa los métodos cuyos están relacionados con el ***movimiento del ratón***. Se llama a un método de la interface ***MouseMotionListener*** cuando el usuario utiliza el ratón (o un dispositivo similar) para mover el cursor o arrastrarlo sobre la pantalla. Los métodos de la interface ***MouseMotionListener*** son ***mouseMoved()*** y ***mouseDragged()***.

A continuación, se presenta un resumen de los objetos más importantes en el proceso de tratamiento de eventos de ratón:

**MouseListener:**

- **mouseClicked(MouseEvent e)** Se hace click (presión y liberación) con un botón del ratón
- **mouseEntered(MouseEvent e)** Se introduce el puntero del ratón en el interior de un componente
- **mouseExited(MouseEvent e)** Se saca el puntero del ratón del interior de un componente
- **mousePressed(MouseEvent e)** Se presiona un botón del ratón
- **mouseReleased(MouseEvent e)** Se libera un botón del ratón (que había sido presionado)

**MouseMotionListener**

- **mouseDragged(MouseEvent e)** Se presiona un botón y se arrastra el ratón
- **mouseMoved(MouseEvent e)** Se mueve el puntero del ratón en un componente (sin usar un botón)

**Mouse WheelListener**

- **Mouse WheelMoved (MouseWheelEvent e)** Se mueve la rueda del ratón

A continuación se muestran algunos de los métodos de la clase **MouseEvent:**

- **int getButton()** Indica qué botón del ratón ha cambiado su estado (en caso de que alguno haya cambiado de estado). Se puede hacer la consulta comparando con los **valores BUTTON1, BUTTON2 y BUTTON3.**
- **int getClickCount()** Número de clics asociados con este evento.
- **Point getPoint()** Devuelve la posición X, Yen la que se ha generado este evento (posición relativa al componente).
- **int getX()** Devuelve la posición X en la que se ha generado este evento (posición relativa al componente).
- **int getY()** Devuelve la posición Y en la que se ha generado este evento (posición relativa al componente)

- **Object getSource()** Método perteneciente a la clase *EventObject* (superclase de *MouseEvent)*. Indica que se produjo el evento.

## 4.1 Eventos de ratón

En este apartado vamos a desarrollar un ejemplo que muestran el uso del interfaz MouseListener.

```
import java.awt.event.*;
public class InterrupcionDeRaton extends Object implements MouseListener{
   public void mouseClicked(MouseEvent EventoQueLlega){
        System.out..println("Click de raton"); }
   public void mousePressed(MouseEvent EventoQueLlega){
        System.out..println("Presion de raton"); }
  public void mouseReleased(MouseEvent EventoQueLlega){
        System.out..println("Se ha levantado el botón del raton"); }
  public void mouseEntered(MouseEvent EventoQueLlega){
        System.out..println("Focus de raton"); }
  public void mouseExited(MouseEvent EventoQueLlega){
        System.out..println("Blur de raton"); } }
```

Para conseguir que la clase InterrupcionDeRaton actúe sobre uno o varios componentes, debemos instanciarla y enlazarla con los componentes deseados.

```
import java.awt.*;
public class PruebaEventosRaton {
        public static void main(String[] args) {
                Frame MiFrame = new Frame ("Prueba eventos de raton");
                Panel MiPanel = new Panel();
                Button Hola=new Button ("Saludo");
                Button Adios=new Button ("Despedida);
                MiPanel.add(Hola); MiPanel.add(Adios);
                MiFrame.add(MiPanel);
                MiFrame.setSize(200,100);
                MiFrame.show();
```

```
            //Crea una instancia de la clase InterrupcionDeRaton y la enlaza al botón
            Hola.addMouseListener(new InterrrupcionDeRaton());
            Adios.addMouseListener(new InterrrupcionDeRaton());
}
```

## 4.2 Eventos de ratón

Los eventos de movimiento de ratón nos permiten asociar acciones a cada movimiento del ratón a través de un componente; de esta manera se podría implementar una aplicación que dibuje al trayectoria del ratón con origen en el punto donde pulsamos el ratón y con final en cada posición por donde movemos el ratón con el botón pulsado. Tal y como vimos anteriormente, los métodos involucrados en el interfaz MouseMotionListener son mouseMoved y mouseDragged.

# 5. Eventos Teclado y Ventana

Los eventos de teclado y ventana se capturan haciendo uso de los siguientes interfacer (o adaptadores) y eventos:

| Evento | Interface | Adaptador |
|---|---|---|
| KeyEvent | KeyListener | KeyAdapter |
| WindowEvent | WindowListener | WindowAdapter |

A continuación, se presenta un resúmen de los objetos más importantes en el proceso de tratamiento de eventos de teclado y de ventana:

**KeyListener**

- **keyPressed(KeyEvent e):** Se ha presionado una tecla
- **keyRelease(KeyEvent e):** Se ha terminado la presión de una tecla
- **keyTyped(KeyEvent e):** Se ha presionado (en algunas ocasiones presionado y soltado) una tecla.

**WindowListener**

- **windowActivated(windowEvent e):** La ventana pasa a ser la activa.
- **windowDeactivated(windowEvent e):** La ventana deja de ser la activa.
- **windowOpened(windowEvent e):** La primera vez que la ventana se hace visible.
- **windowClosing(windowEvent e):** El usuario indica que se cierra la ventana
- **windowClosedwindowEvent e):** La ventana se ha cerrado.

- **windowIconified(windowEvent e):** La ventana pasa de estado normal a un estado minimizado.
- **windowDeIconified(windowEvent e):** La ventana pasa de estado minimizado a un estado normal.

**KeyEvent**

- **Char getKetCgar():** Devuelve el carácter asociado con la tecla pulsada.
- **Int getKeyCode():** Devuelve el valor entero que representa la tecla pulsada.
- **String getKeyText():** Devuelve un texto que representa el código de la tecla.
- **Object getSource():** Método perteneciente a la clase EventObject. Indica el objeto que produjo el evento.

**WindowEvent**:

- **Window getWindow():** Devuelve la ventana que origina el evento.
- **Window getOppositeWindow():** Devuelve la ventana involucrada en el camcio de activación o de foco.
- **int getNewState():** Para WINDOW_STATE_CHANGED indica el nuevo estado de la ventana.
- **int getOldState():** Para WINDOW_STATE_CHANGED indica el antiguo estado de la ventana.
- **Object getSource():** Método perteneciente a la clase EventObject. Indica el objeto que produjo el evento.

## 5.1 Eventos de ventana

En este apartado, se va a desarrollar un ejemplo que muestra el uso del interfaz WindowListener, en el que únicamente aparece implementado el método windowClosing().

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Marco extends Frame implements WindowListener {
   Marco(String título,int x,int y,int dx,int dy) {
            super(título);
            setBackground(Color.red);
            Label lbl=new Label("Frame",Label.CENTER);
            setBackground(Color.red);
            add("Center",lbl);
```

```
            MenuBar menubar=new MenuBar();
            setMenuBar(menubar);
            Menu menu1=new Menu("menu1");
            menu1.add(new MenuItem("item 1.1"));
            menu1.add(new MenuItem("item 1.2"));
            menu1.add(new MenuItem("item 1.3"));
            Menu menu2=new Menu("menu2");
            menu2.add(new MenuItem("item 2.1"));
            menu2.add(new MenuItem("item 2.2"));
            menubar.add(menu1);
            menubar.add(menu2);
            pack();
            setSize(dx,dy);
            setLocation(x,y);
            setVisible(true);
            addWindowListener(this);
    }
    /*------------ WindowListener ----------*/
    public void windowOpened(WindowEvent e){}
    public void windowActivated(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowClosed(WindowEvent e){}
    public void windowClosing(WindowEvent e){
            setVisible(false);
    }
}
```

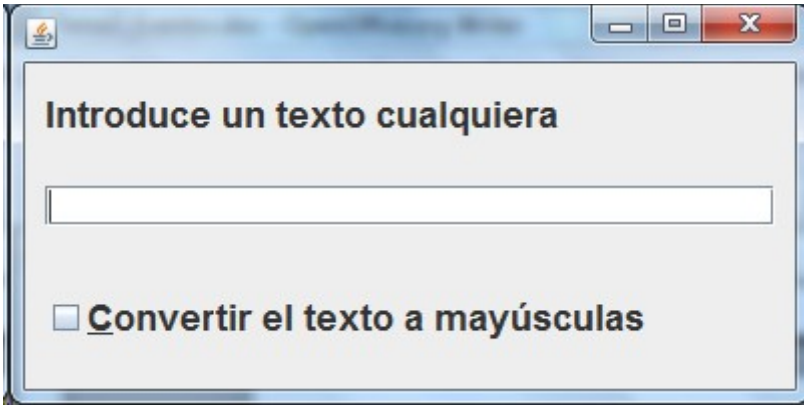
## 5.2 Eventos de teclado

En este apartado, se va a desarrollar un ejemplo que muestra el uso del interfaz

```
public class Interfaz_frase extends JFrame implements ItemListener, KeyListener {
    Font letra ;
    JLabel JLtexto;
    JCheckBox JChBmayuscula;
    JTextField JTFtexto;
    Container conpane ;
    GridBagConstraints restricciones;
    Boolean pulsada=false;

    //Constructor
    public Interfaz_frase(){
            conpane = getContentPane();
            conpane.setLayout(new GridBagLayout());
            restricciones = new GridBagConstraints();
            letra=new Font("SansSerif",Font.BOLD,18);

            JChBmayuscula = new JCheckBox("Convertir el texto a mayúsculas");
            JChBmayuscula.setFont(letra);

            try {
                    JChBmayuscula.setDisplayedMnemonicIndex(0);
            }
            catch (IllegalArgumentException e){
                    System.err.print(e.getMessage());
            }

            restricciones.fill=GridBagConstraints.HORIZONTAL;
            restricciones.insets=new Insets(10,10,10,10);


            JLtexto=new JLabel("Introduce un texto cualquiera");
            JLtexto.setAlignmentX(CENTER_ALIGNMENT);
```

```
        JLtexto.setFont(letra);
        añadeGrid(JLtexto ,0,0,1.0,0.4);

        JTFtexto=new JTextField(40);
        añadeGrid(JTFtexto ,0,1,1.0,0.0);

        // Poner la casilla de verificacion a la escucha
        JChBmayuscula.addItemListener(this);
        añadeGrid(JChBmayuscula ,0,2,1.0,1.0);

        //Poner a la escucha de los eventos del teclado
        JTFtexto.addKeyListener(this);
        JChBmayuscula.addKeyListener(this);
        }

// Método privado que crea las restricciones de cada componente
private void añadeGrid(Component c, int x, int y, double wx, double wy){
        restricciones.gridx=x;
        restricciones.gridy=y;
        restricciones.weightx=wx;
        restricciones.weighty=wy;
        add(c,restricciones);
}

//Comprobar si la casilla de verificacion esta marcada o no
public void itemStateChanged(ItemEvent e){

        if (e.getStateChange()==ItemEvent.SELECTED){
                JTFtexto.setText(JTFtexto.getText().toUpperCase());
        }
        else{
                JTFtexto.setText(JTFtexto.getText().toLowerCase());
        }
```

```
}
public static void main(String args[]){
        Interfaz_frase ventana= new Interfaz_frase();
        //Poner la ventana a la escucha de los eventos del teclado
        ventana.addKeyListener(ventana);
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setSize(400,200);
        ventana.setVisible(true);
}
@Override
public void keyPressed(KeyEvent arg0) {
        // TODO Auto-generated method stub
}
@Override
public void keyReleased(KeyEvent arg0) {
        // TODO Auto-generated method stub
}
@Override
public void keyTyped(KeyEvent arg0) {
        // TODO Auto-generated method stub
        if (arg0.isAltDown()){
                pulsada=true;
        }
        if (pulsada &&(arg0.getKeyChar()=='c' || arg0.getKeyChar()=='C' )){
                if (JChBmayuscula.isSelected())
                        JChBmayuscula.setSelected(false);
                else
                        JChBmayuscula.setSelected(true);
        }       }}
```




# 6.Eventos de Acción, Enfoque y Elemento

A veces es interesante enterarnos de cuando un componente java -un botón, un campo de texto, etc- gana o pierde el foco. Por ejemplo, podemos querer que cambie el color del botón que tiene el foco, de forma que resalte sobre los demás.

Para enterarnos de cuando un componente java gana o pierde el foco, hay que añadirle un `FocusListener`, es decir, una clase que implemente la interface `FocusListener`.

El siguiente código, por ejemplo, hace que un botón se ponga de color rojo cuando gana el foco y recupere su color original cuando lo pierde

```
JButton boton = new JButton ("Pulsame");
// Guardamos el color de background por defecto.
final Color colorBackgroundDefecto = boton.getBackground();
...
boton.addFocusListener(new FocusListener() {
  public void focusGained(FocusEvent e) {
    e.getComponent().setBackground(Color.RED);
    // e.getOpositeComponent() devuelve el Component que le
    // cede el foco a nuestro botón.
  }
  public void focusLost(FocusEvent e) {
    e.getComponent().setBackground(colorBackgroundDefecto);
    // get.OpositeComponent() devuelve el Component al que
    // nuestro botón cede el foco.
  }
});
```

**FocusEvent**

- **`GetOpositeComponent():`** Obtenemos el elemento que ha ganado o perdido el foco. . Basta con cambiarle el color a ese `Component`, que es el que origina el evento y es, por tanto, el botón.
- **`GetOpositeComponent():`**, que nos permite saber qué `Component` le ha cedido el foco a nuestro botón o a qué `Component` le cede el foco nuestro botón.

  Por ejemplo, si tiene el foco un `JTextField` y el foco pasa a nuestro `JButton`, `getComponent()` nos devolverá el `JButton` y `getOpositeComponent()` nos devolverá el `JTextField`. Y al revés, si el `JButton` tiene el foco y se lo pasa al `JTextField`, `getComponent()` seguirá devolviendo el `JButton` y `getOpositeComponent()` seguirá devolviendo el `JTextField`.

Si no tenemos interés en ambos casos, pérdida y ganancia del foco, sino sólo en uno de ellos, en vez de usar `FocusListener` e implementar ambos métodos, podemos usar `FocusAdapter` e implementar sólo uno de ellos. `FocusAdapter` es una clase que implementa a `FocusListener`, pero cuyos métodos no hacen nada. Es tarea nuestra redefinir sólo aquellos que queramos que hagan algo.

```
boton.addFocusListener (new FocusAdapter() {
  public void focusGained (FocusEvent e) {
    // El botón gana el foco
  }
  // No hace falta definir focusLost()
});
```