TEMA 7

Eventos

Todos los S,O. están constantemente atendiendo a los eventos generados por los usuarios. Estos eventos van desde pulsar una tecla, mover el ratón, hacer clic sobre un menú, sobre un botón.... Es el sistema operativo quién comunica a las aplicaciones que se están produciendo estos eventos para que sean ellas las que decidan si responder o no a estos eventos y de qué manera.

Modelo de delegación de eventos

Las principales características del sistema de delegación de eventos deben ser las siguientes:

- Que sea simple y fácil de aprender.
- Que soporte una clara separación entre el código de la aplicación y el código del interfaz.
- Que facilite la creación de robustos controladores de eventos, con menos posibilidad de generación de errores.
- Suficientemente flexible para permitir el flujo y propagación de eventos.
- Para herramientas visuales, permitir en tiempo de ejecución ver cómo se generan estos eventos y quien lo hace.

Los eventos están organizados en jerarquías de clases de eventos:

- Fuentes de Eventos (**Source**) Es un objeto que tiene la capacidad de detectar eventos y notificar a los receptores de eventos que se han producido estos eventos

- Receptor de Eventos (**Listener**) Es un objeto que está preparado para ser notificado de la ocurrencia de un evento. Una vez que el objeto receptor está registrado para ser notificado de esos eventos, el suceso

de un evento en esta clase automática, invocará al método sobrescrito del objeto receptor.

Algunas clases de eventos, como los de ratón, involucran a un determinado conjunto de eventos diferentes. Una clase receptora de eventos que implemente el interfaz que recoja estos eventos debe sobrescribir todos los métodos declarados en el interfaz.

Para prevenir esto, de forma que no sea tan tedioso y no hay que sobrescribir métodos que no se van a utilizar, se han definido un conjunto de clases intermedias, conocidas como Adaptadores (Adapter)

Clases Adapter

Estas clases Adaptadores implementan los interfaces receptores y sobrescriben todos los métodos del interfaz con métodos vacíos. Una clase receptor puede estar definida como clase que extienda de una clase Adapter, en lugar que implemente el interfaz completo.

Cuando se hace esto, la clase receptora solamente necesitará sobrescribir aquellos métodos que sean de interés para la aplicación, porque los otros métodos ya estarán resueltos por la clase Adapter

```
import javax.swing.*;
import java.awt.event.*;

public class ejemplolFrame {
   public static void main( String args[] ) {
      // Aqui se instancia un objeto de tipo Interfaz Hombre-Maquina
      frame ihm = new frame();
      }
   }

//Clase que dibuja un JFrame y añade un receptor de eventos a la ventana

class frame extends JFrame{
   // Constructor de la clase
   public frame() {
      // Se crea un objeto Frame
      JFrame ventana = new JFrame();

      // El metodo setSize() reemplaza al metodo resize() del JDK 1.0
      ventana.setSize( 300,200 );
      ventana.setTitle( "Curso de Java. Eventos" );
      // El metodo setVisible() reemplaza al metodo show() del JDK 1.0
      ventana.setVisible( true );
...
```

```
...
        // Se crean dos objetos receptores que procesaran los
        // eventos de la ventana
        listener1 proceso1 = new listener1( ventana );
        listener2 proceso2 = new listener2();

        // Se registran los dos objetos receptores para que sean
        // notificados de los evetnos que genere la ventana, que es el
        // objeto origen de los eventos
        ventana.addWindowListener( proceso1 );
        ventana.addWindowListener( proceso2 );
        }
    }
```

Ejemplo 7.1: Método para la creación de una ventana añadiendo receptores de eventos al JFrame.

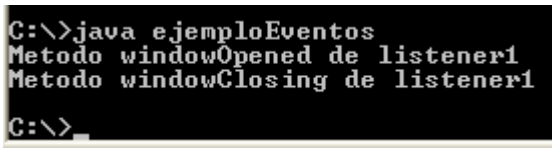
```
class listener1 implements WindowListener {
 // Variable utilizada para guardar una referencia al objeto JFrame
 JFrame ventanaRef;

 // Constructor que guarda la referencia al objeto JFrame
 Proceso1( JFrame vent ){
    this.ventanaRef = vent;
    }

 public void windowClosed( WindowEvent evt ) {
    System.out.println( "Metodo windowClosed de listener1" );
    }

 public void windowIconified( WindowEvent evt ) {
    System.out.println( "Metodo windowIconified de listener1" );
    }

 public void windowOpened( WindowEvent evt ) {
    System.out.println( "Metodo windowOpened de listener1" );
    }

 public void windowClosing( WindowEvent evt ) {
    System.out.println( "Metodo windowClosing de listener1" );
    // Se oculta la ventana
    ventanaRef.setVisible( false );
    }

 public void windowDeiconified( WindowEvent evt ) {
    System.out.println( "Metodo windowDeiconified listener1" );
    }

 public void windowActivated( WindowEvent evt ) {
    System.out.println( "Metodo windowActivated de listener1" );
    }

 public void windowDeactivated( WindowEvent evt ) {
    System.out.println( "Metodo windowDeactivated de listener1" );
    }
 }
```

Ejemplo 7.2: Implementación de una clase listener para una ventana

```
class listener2 extends WindowAdapter {
   public void windowIconified( WindowEvent evt ) {
      System.out.println( "--- Metodo windowIconified de listener2" );
      }

   public void windowDeiconified( WindowEvent evt ) {
      System.out.println( "---Metodo windowDeiconified de listener2" );
      }
   }
```

Ejemplo 7.2 bis: Dos formas de implementar una clase listener para una ventana.

La visualización en la consola cuando se pulse el aspa de cierre será la siguiente:



Como se ha observado en el ejemplo anterior, en algunos casos es mejor idea heredar los métodos de una clase Adapter, sobrescribiendo los métodos que se desee desarrollar a implementar todos los métodos de una interface.

Acción que produce el evento	Tipo de oyente
El usuario pulsa un botón, presiona Return mientras teclea en un campo de texto, o elige un ítem de menú.	**ActionListener**
El usuario elige un frame (ventana principal).	**WindowListener**
El usuario pulsa un botón del ratón mientras el cursor está sobre un componente.	**MouseListener**
El usuario mueve el cursor sobre un componente.	**MouseMotionListener**
El componente se hace visible.	**ComponentListener**
El componente obtiene obtiene el foco del teclado.	**FocusListener**
Cambia la tabla o la selección de una lista.	**ListSelectionListener**

Ejemplos de eventos producidos y los listener que escuchan estos tipos de eventos

A continuación se ofrece una tabla con datos significativos acerca de las interfaces, sus clases Adapter, si las tienen, el paquete al cual pertenecen y los métodos que contiene.

Interface	Clase Adapter	Paquete	Métodos
ActionListener	ninguna	java.awt.event	actionPerformed
CaretListener	ninguna	javax.swing.event	caretUpdate
ChangeListener	ninguna	javax.swing.event	stateChanged
ComponentListener	ComponentAdapter	java.awt.event	componentHidden componentMoved componentResized componentShown
ContainerListener	ContainerAdapter	java.awt.event	componentAdded componentRemoved
DocumentListener	ninguna	javax.swing.event	changedUpdate insertUpdate removeUpdate
FocusListener	FocusAdapter	java.awt.event	focusGained focusLost
InternalFrameListener	InternalFrameAdapter	javax.swing.event	internalFrameActivated internalFrameClosed internalFrameClosing internalFrameDeactivated internalFrameDeiconified internalFrameIconified internalFrameOpened
ItemListener	ninguna	java.awt.event	itemStateChanged
KeyListener	KeyAdapter	java.awt.event	keyPressed keyReleased keyTyped
ListSelectionListener	ninguna	javax.swing.event	valueChanged
MouseListener	MouseAdapter MouseInputAdapter*	java.awt.event javax.swing.event	mouseClicked mouseEntered mouseExited mousePressed mouseReleased
MouseMotionListener	MouseMotionAdapter MouseInputAdapter*	java.awt.event javax.swing.event	mouseDragged mouseMoved
UndoableEditListener	ninguna	javax.swing.event	undoableEditHappened
WindowListener	WindowAdapter	java.awt.event	windowActivated windowClosed windowClosing windowDeactivated windowDeiconified windowIconified windowOpened

* Swing proporciona la clase **MouseInputAdapter** por conveniencia. Implementa los interfaces **MouseListener** y **MouseMotionListener** haciendo más fácil el manejo de ambos tipos de eventos.

Añadiendo escuchador de eventos

Por ejemplo, si se desea que un objeto escucha los eventos de otro objeto se emplea el método **add[nombre_evento]Listener**, en el ejemplo anterior (Ejemplo 28) al objeto "manejador" le añadimos un escuchador de eventos de ventana (addWindowListener).

La siguiente tabla muestra una lista de eventos, interfaces y métodos que se utilizan para añadir esta funcionalidad a un objeto.

Eventos, interfaces y métodos de escucha	**Componentes que lo generan**
ActionEvent ActionListener addActionListener()	JButton, JList, JTextField, JmenuItem, JCheckBoxMenuItem, JMenu, JpopupMenu
AdjustmentEvent AdjustmentListener addAdjustmentListener()	JScrollbar y cualquier objeto que implemente la interface Adjustable
ComponentEvent ComponentListener addComponentListener	Component, JButton, JCanvas, JCheckBox, JComboBox, Container, JPanel, JApllet, JScrollPane, Window, JDialog, JFileDialog, JFrame, JLabel, JList, JScrollbar, JTextArea, JtextField
ContainerEvent ContainerListener addContainerListener()	Container, JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog, JFrame
FocusEvent FocusListener addFocusListener()	Component, JButton, JCanvas, JCheckBox, JComboBox, Container, JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog, JFrame, JLabel, JList, JScrollbar, JTextArea, JTextField
KeyEvent KeyListener addKeyListener()	Component, JButton, JCanvas, JCheckBox, JComboBox, Container, JPanel, JApplet,

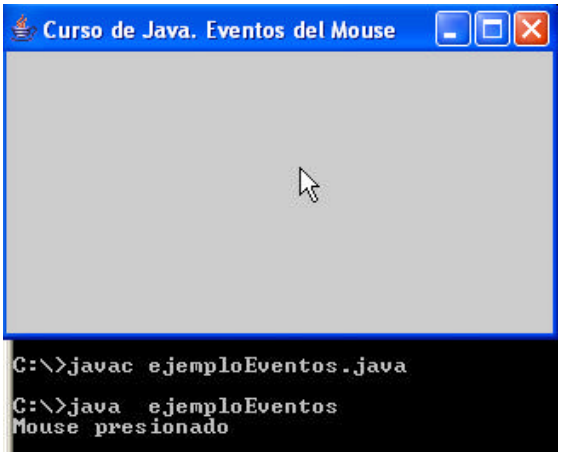
	JScrollPane, Window, JDialog, JFileDialog, JFrame, JLabel, JList, JScrollbar, JTextArea, JTextField
MouseEvent MouseListener addMouseListener()	Component, JButton, JCanvas, JCheckBox, JComboBox, Container, JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog, JFrame, JLabel, JList, JScrollbar, JTextArea, JTextField
MouseEvent MouseMotionListener addMouseMotionListener()	Component, JButton, JCanvas, JCheckBox, JComboBox, Container, JPanel, JApplet, JScrollPane, Window, JDialog, JFileDialog, JFrame, JLabel, JList, JScrollbar, JTextArea, JTextField
WindowEvent WindowListener addWindowListener()	Window, JDialog, JFileDialog, JFrame
ItemEvent ItemListener addItemListener()	JCheckBox, JCheckBoxMenuItem, JComboBox, JList
TextEvent TextListener addTextListener()	JTextComponent, JTextArea, JTextField

En el siguiente ejemplo muestra la utilización del interface MouseListener para escuchar los eventos del ratón.

```
import javax.swing.*;
import java.awt.event.*;

class frame extends JFrame {
        public frame()  {
          setTitle("Curso de Java. Eventos del Mouse");
          setSize(300,200);
          //Le indicamos que tiene un  manejador de eventos asociado.
          addMouseListener(new manejador());
        }
  }

//La clase manejador extiende de la clase MouseAdapter por lo que hereda sus métodos y
sólo se tendrá que sobrescribir el método que nos interesa.
class manejador extends MouseAdapter{
        public void mousePressed(MouseEvent e)
          {System.out.println("Mouse presionado");}
  }

//Por último escribimos la clase principal
public class ejemploEventos
  {
        public static void main(String[] args) {
          JFrame frame = new frame();
          frame.setVisible(true);
        }
  }
```

Ejemplo 7.3: Clase con un escuchador de eventos del Mouse

La visualización que se produciría sería, al igual que en ejemplos anteriores, una ventana, con la salvedad que cuando se hiciese clic en ella se mostraría el mensaje “Mouse presionado”:



Laboratorio

1. Este programa mostrará información del componente que se pulse en el interfaz de usuario.

```
import java.awt.*;
import java.awt.event.*;

public class ejemploEventos2 {
   public static void main( String args[] ) {
      // Aqui se instancia un objeto de tipo Interfaz Hombre-Maquina
      IHM ihm = new IHM();
      }
   }

// Esta clase se utiliza para instaciar un objeto de tipo interfaz de
// usuario
class IHM {
   public IHM() {
      // Crea un objeto visual Campo de Texto (TextField)
      TextField miTexto = new TextField( "Cadena Inicial" );
      miTexto.setName( "CampoTexto" );

      // Crea un objeto visual Boton (Button)
      Button miBoton = new Button( "Pulsa Aqui" );
      miBoton.setName( "Boton" );

      // Crea un objeto visual Frame
      Frame miFrame = new Frame();
      miFrame.setSize( 300,200 );
      miFrame.setTitle( "Curso de Java, Eventos" );
      miFrame.setName( "Frame" );

      // Incorpora el Campo de Texto y el Boton al objeto de tipo Frame
      miFrame.add( "North",miBoton );
      miFrame.add( "South",miTexto );
      miFrame.setVisible(true);

      // Se instancia y registra in objeto MouserListener que procesara
      // todos los eventos del raton que se produzcan o sean generados
      // por los objetos Frame, Button y TextField
      ProcesoRaton procesoRaton = new ProcesoRaton();
      miFrame.addMouseListener( procesoRaton );
      miTexto.addMouseListener( procesoRaton );
      miBoton.addMouseListener( procesoRaton );

      // Se instancia y registra un objeto Receptor que hara que
      // concluya la ejecucion del programa cuando el usuario cierre
      // la ventana
      Proceso1 procesoVentana1 = new Proceso1();
      miFrame.addWindowListener( procesoVentana1 );
      }
   }

...
```

```
...

// Monitor de eventos de Bajo-Nivel
// Esta clase Receptor monitoriza todos los eventos mousePressed() de
// bajo-nivel. Cuando se produce un evento de pulsacion del raton,
// el controlador de eventos obtiene y presenta informacion variada
// acerca del objeto que ha generado el evento
class ProcesoRaton extends MouseAdapter {
   public void mousePressed( MouseEvent evt ) {
      System.out.println( "Nombre = " + evt.getComponent().getName() );
      try {
         System.out.println(
            "Nombre del padre = " +
            evt.getComponent().getParent().getName() );
      } catch( NullPointerException exception ) {
         System.out.println( "No hay ningun padre a este nivel" );
         }
      System.out.println( "Localizacion = " +
         evt.getComponent().getLocation().toString() );
      System.out.println( "Tamano Minimo = " +
         evt.getComponent().getMinimumSize().toString() );
      System.out.println( "Tamano = " +
         evt.getComponent().getSize().toString() );
      System.out.println();
      }
   }


// Este repector de eventos de la ventana se utiliza para concluir
// la ejecucion del programa cuando el usuario pulsa sobre el boton
// de cierre del Frame
class Proceso1 extends WindowAdapter {
   public void windowClosing( WindowEvent evt ) {
      System.exit( 0 );
      }
   }
```

2. Otro ejemplo de captura de eventos. En este caso detectaremos la también la pérdida del focus del botón del interfaz. Analicémosla clase por clase:

Esta es la clase principal, que instancia un objeto de tipo Interfaz Hombre-Máquina.

```
import java.awt.*;
import java.awt.event.*;

public class ejemploEventos3 {
   public static void main( String args[] ) {
            IHM ihm = new IHM();
      }
   }
```

Esta es la clase que dibuja la interfaz de usuario

```
class IHM {
   public IHM() {
      // Crea un objeto visual Campo de Texto (TextField)
      TextField miTexto = new TextField( "Cadena Inicial" );
      miTexto.setName( "CampoTexto" );

      // Crea un objeto visual Boton (Button)
      Button miBoton = new Button( "Púlsame" );
      miBoton.setName( "Boton" );

      // Crea un objeto visual Frame
      Frame miFrame = new Frame();
      miFrame.setSize( 300,200 );
      miFrame.setTitle( "Curso de Java, Eventos" );
      miFrame.setName( "Frame" );

      // Incorpora el Campo de Texto y el Boton al objeto de tipo Frame
      miFrame.add( "North",miBoton );
      miFrame.add( "South",miTexto );
      miFrame.setVisible( true );

      // Se instancia y registra un objeto ActionListener que
      // monitorizara todos los eventos de acciones que tengan
      // su origen en el Campo de Texto y en el Boton
      ProcesoAccion procesoAccion = new ProcesoAccion();
      miTexto.addActionListener( procesoAccion );
      miBoton.addActionListener( procesoAccion );
...
```

```
...
        // Se instancia y registra un objeto FocusListener que
        // monitorizara todos los eventos producidos por cambios
        // en el foco que tengan su origen en el Campo de Texto y
        // en el Boton
        ProcesoFoco procesoFoco = new ProcesoFoco();
        miTexto.addFocusListener( procesoFoco );
        miBoton.addFocusListener( procesoFoco );

        // Se instancia y registra un objeto MouserListener que procesara
        // todos los eventos del raton que se produzcan o sean generados
        // por los objetos Frame, Button y TextField
        ProcesoRaton procesoRaton = new ProcesoRaton();
        miFrame.addMouseListener( procesoRaton );
        miTexto.addMouseListener( procesoRaton );
        miBoton.addMouseListener( procesoRaton );

        // Se instancia y registra un objeto receptor de eventos
        // para terminar la ejecucion del programa cuando el
        // usuario decida cerrar la ventana
        Proceso1 procesoVentana1 = new Proceso1();
        miFrame.addWindowListener( procesoVentana1 );
        }
    }
```

Esta clase es la receptora de eventos semánticos, implementa la interface ActionListener y se utiliza para instanciar un objeto Receptor que monitorice todos los eventos Action que se generen en los objetos TextField y Button. Cuando se produce un evento de tipo actionPreformed(), se presenta en pantalla el ActionCommand y la identificacion del componente que ha generado el evento. El objeto receptor distingue entre los componentes que de envian eventos sobre la base del nombre que se ha asignado a cada uno de los objetos y que se encuentra embebido en el objeto que es pasado como parametro cuando se produce el evento.

```
class ProcesoAccion implements ActionListener {
    public void actionPerformed(ActionEvent evt ) {
        System.out.println("evt.getActionCommand() = " +
            evt.getActionCommand());

        if( evt.toString().indexOf("on CampoTexto") != -1 ) {
            System.out.println(
                "Capturado actionPerformed sobre el objeto CampoTexto");
            }
        if( evt.toString().indexOf("on Boton") != -1 ) {
            System.out.println(
                "Capturado actionPerformed sobre el objeto Boton");
            }
        }
    }
```

Esta clase es la receptora de eventos de bajo nivel que monitoriaza los eventos relacionados con el foco que se generen en los objetos TextField y Button. Cuando se produce un evento de tipo focusLost() o focusGained(), se presenta en pantalla la identificacion del componente que ha generado el evento. El objeto receptor distingue entre los componentes que de envian eventos sobre la base del nombre que se ha asignado a cada uno de los objetos y que se encuentra embebido en el objeto que es pasado como parametro cuando se produce el evento.

```
class ProcesoFoco implements FocusListener{
   public void focusGained( FocusEvent evt ) {
      if( evt.toString().indexOf("on CampoTexto") != -1 ) {
         System.out.println(
            "Capturado focusGained sobre el objeto CampoTexto" );
         }
      if( evt.toString().indexOf("on Boton") != -1 ) {
         System.out.println(
            "Capturado focusGained sobre el objeto Boton" );
         }
      }

   public void focusLost( FocusEvent evt ) {
      if( evt.toString().indexOf("on CampoTexto") != -1 ) {
         System.out.println(
            "Capturado focusLost sobre el objeto CampoTexto" );
         }
      if( evt.toString().indexOf("on Boton") != -1 ) {
         System.out.println(
            "Capturado focusLost sobre el objeto Boton" );
         }
      }
   }
```

Esta clase es la clase receptora de eventos de bajo nivel que monitorizan los eventos de pulsacion de los botones del raton sobre el objeto Frame, el objeto Button y el objeto TextField. El mensaje indentifica el componente que ha generado el evento.El receptor distingue entre los componentes que de envian eventos sobre la base del nombre que se ha asignado a cada uno de los objetos y que se encuentra embebido en el objeto que es pasado como parametro cuando se produce el evento.

```
class ProcesoRaton extends MouseAdapter {
   public void mousePressed( MouseEvent evt ) {
      if( evt.toString().indexOf("on Frame") != -1 ) {
         System.out.println(
            "Capturado mousePressed sobre el objeto Frame" );
         }
      if( evt.toString().indexOf("on CampoTexto") != -1 ) {
         System.out.println(
            "Capturado mousePressed sobre el objeto CampoTexto" );
         }
      if( evt.toString().indexOf("on Boton") != -1 ) {
         System.out.println(
            "Capturado mousePressed sobre el objeto Boton" );
         }
      }
   }
```

Por último la clase receptora de eventos de bajo nivel que se encarga de monitorizar los eventos de la ventana, es decir, la conclusión del programa cuando el usuario pulsa sobre el botón de cierre del Frame

```
class Proceso1 extends WindowAdapter {
   public void windowClosing( WindowEvent evt ) {
      System.exit( 0 );
      }
   }
```

Ejercicios

- Crear una aplicación que muestre un JFrame y un botón que debe tener como texto "Pulsa aquí para empezar".

Utilizando la siguiente línea en el método main, cambie la apariencia del entorno gráfico:

```
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
```

Además, utiliza el método siguiente, invocado sobre un botón:

```
setToolTipText("si pulsas sobre este botón...");
```

Añade el código necesario para que, al pulsar sobre el botón, su texto muestre un mensaje indicando el número de veces que se ha pulsado: "Has pulsado N veces".

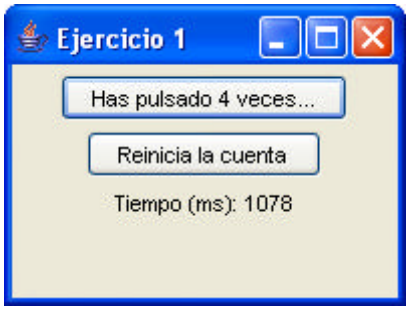
Para ello, declara un atributo de clase que sirva de contador e imprime el mensaje en el código de gestión del evento.:

```
public void actionPerformed(ActionEvent e) {
  ... // aquí va tu código
}
```

Añade otro botón de tal forma que, al pulsarlo, ponga a cero la cuenta de pulsaciones. Como texto, puedes poner "Volver a empezar" (o algo así).

Por último, añadir una etiqueta (JLabel) a la ventana, de tal forma que, al pulsar cualquiera de los botones, indique el tiempo transcurrido (en segundos) desde la última vez que se produjo alguna pulsación.

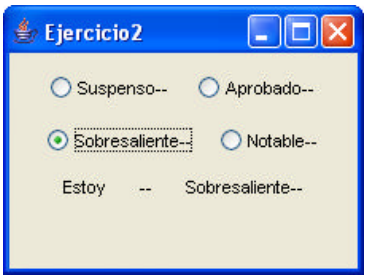
Para calcular el tiempo, utiliza el método estático currentTimeMillis de la clase System, que devuelve el tiempo actual en milisegundos. Al pulsar el botón, guarda en un atributo de clase el valor del tiempo actual. Al volver a pulsar, la diferencia entre el tiempo actual y el tiempo almacenado en el atributo será el tiempo transcurrido entre ambas pulsaciones.



- Crear una aplicación compuesta por una ventana con una etiqueta (es decir, un campo de texto con un valor asignable) y un grupo de objetos *JRadioButton* formado por cinco opciones: **Suspenso**, **Aprobado**, **Notable**, **Sobresaliente** y **Matrícula**.

La etiqueta debe tener el valor por defecto inicial "¿Cuál es mi nota?". Cada vez que se seleccione un *JRadioButton* de los anteriores, la etiqueta debe cambiar su valor por la cadena "Estoy ...", donde "..." es el texto del *JRadioButton* seleccionado.

Los *RadioButton* están asociados a un *ButtonGroup*, y cada uno de ellos debe tener asociado un *ItemListener*, encargado de responder a los eventos de cambio de estado (seleccionado o no seleccionado).



Construir una aplicación que muestre una ventana con los siguientes componentes:

- Un botón con el texto "Copia".
- Un botón con una imagen.
- Dos campos de texto.
- Un JCheckBox con tres opciones.
- Un JRadioButton con dos opciones.
- Un JComboBox.

Para incluir una imagen en un botón, usar el siguiente código:

```
ImageIcon icono = new ImageIcon("icono.gif");
JButton b1 = new JButton("boton1", icono);
```

Al ser pulsado, el botón con la imagen debe cerrar la ventana, y por tanto la aplicación.

De los dos campos de texto, uno debe ser editable (es decir, el usuario debe poder escribir en dicho campo) y otro no (la escritura en dicho campo estará inhabilitada). Además, el valor por defecto de dichos campos será "Aquí escribo" para el campo editable y "Aquí copio" para el campo no editable.

Al pulsar el botón "Copia", se debe copiar lo que hay en el campo editable al campo no editable.