

1.00 Clase 16

El modelo de eventos de Swing

Modelo de eventos de la GUI

- El sistema operativo (Windows, JVM) ejecuta la demostración:
 - Controla el teclado, el ratón, otros eventos de E/S a partir de fuentes.
 - Envía mensajes de eventos a los programas que necesitan saber de ellos.
 - Cada programa decide qué hacer cuando el evento tiene lugar.
- Esto es lo contrario a la programación orientada a consola, donde el programa le pide, cuando quiere, al sistema operativo (SO) que obtenga los datos de entrada.
- Fuentes de eventos: menús, botones, barras de desplazamiento, etc.
 - Poseen métodos que permiten a los oyentes de eventos registrarse con ellos.
 - Cuando el evento tiene lugar, la fuente envía un mensaje (un objeto evento) a todos los objetos oyentes registrados.
 - La superclase es *EventObject*.
 - *ActionEvent*, *MouseEvent*, etc., son las subclases que utilizamos.
- Oyentes de evento: objetos de su programa que responden a los eventos
 - La delegación de eventos permite que el programador escoja el objeto.

Delegación de eventos

- El sistema en tiempo de ejecución de Java genera objetos `AWTEvent` cuando el usuario hace algo con el ratón o con el teclado.
- Los componentes UI le permiten la suscripción a esos eventos, de modo que, cuando tiene lugar un evento al que usted se suscribió, un método del código es invocado.
- En el siguiente ejemplo utilizamos un `JButton`.

Eventos: Ejemplo con JButton

- **Importe el paquete `java.awt.event.*`**
 - Proporciona clases para distintos tipos de eventos y un conjunto de interfaces.
- **Cree un objeto `JButton` y añádalo al panel.**
- **Cree un objeto (puede ser el panel) para implementar la interfaz `ActionListener`. Para cumplir los requisitos, la clase debe tener un método.**

```
void actionPerformed(ActionEvent e)
```

 - El método `actionPerformed` posee el código que usted desea que se ejecute siempre que se pulse el botón.
- **Indique al objeto `JButton` que envíe un `ActionEvent` al objeto que es el `ActionListener`. Si `obj` es el oyente, y `b` es una referencia al objeto `JButton`:**

```
b.addActionListener(obj);
```

Ejemplo de *Button*

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

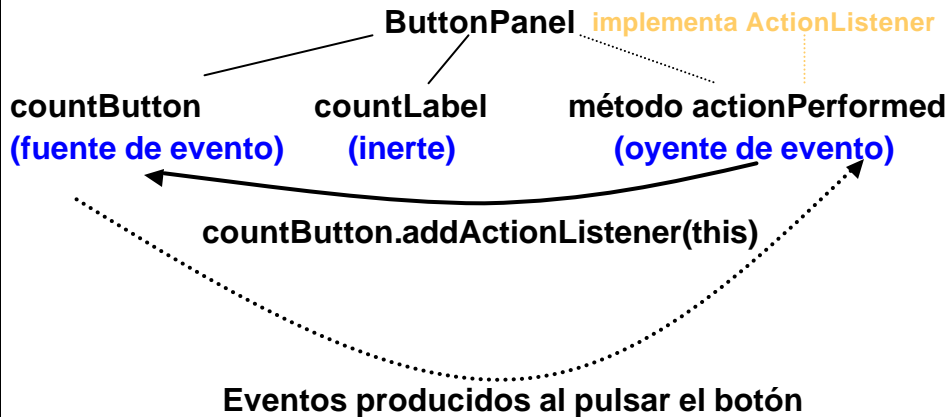
public class Button1 {
    public static void main(String[] args) {
        BFrame frame= new BFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.show();    }    }

class BFrame extends JFrame {
    public BFrame() {
        setTitle("Ejemplo de Button");
        setSize(XSIZE, YSIZE);
        ButtonPanel panel= new ButtonPanel();
        Container contentPane= getContentPane();
        contentPane.add(panel);    }
    public static final int XSIZE= 200;
    public static final int YSIZE= 200;    }
```

Ejemplo de *Button* , 2ª parte

```
class ButtonPanel extends JPanel implements ActionListener {
    public ButtonPanel() {
        JButton countButton= new JButton("Cuenta avanzada");
        JLabel countLabel= new JLabel("Cuenta= 0");
        Font fontShow= new Font("Monospaced", Font.PLAIN, 24);
        countLabel.setFont(fontShow);
        countButton.setFont(fontShow);
        add(countButton);
        add(countLabel);
        countButton.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        i++;
        countLabel.setText("Cuenta= " + i);
        repaint();
    }
    private int i= 0;
    private JLabel countLabel;
}
```

Estructura del ejemplo *Button*



Oyentes de eventos

- Puede seleccionar cualquier objeto, siempre que éste implemente a *ActionListener* para ser el oyente del evento. Tiene tres posibilidades:
 - Añadir el método `actionPerformed` a la clase del elemento de la GUI.
 - Crear una nueva clase como oyente.
 - Crear una "clase interna" como oyente (se tratará más adelante).
- El próximo ejemplo, *ComboBox*, posee múltiples fuentes de evento (3), por lo que debemos oír y distinguir entre 3 tipos de eventos.
 - El ejemplo muestra fuentes seleccionadas por el usuario.
 - Se eligen la familia, el estilo y el tamaño de las fuentes.

ComboBoxApp

```
public class ComboBoxApp
    extends JFrame {
    public ComboBoxApp() {
        setTitle("Ejemplo de ComboBox");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(XSIZE, YSIZE);

        ComboPanel panel= new ComboPanel();
        Container contentPane= getContentPane();
        contentPane.add(panel, "Center" );
    }
    public static final int XSIZE= 600;
    public static final int YSIZE= 400;
}
```

ComboBoxApp, 2

```
class ComboPanel extends JPanel implements ActionListener
{
    public ComboPanel() {
        String[][] fontOptions= {
            {"Monospaced", "Serif", "SansSerif"},
            {"NORMAL", "NEGRITA", "CURSIVA"},
            {"10", "12", "14", "18", "24", "36"} };

        setLayout( new BorderLayout() );
        chFamily= new JComboBox(fontOptions[0]);
        chStyle= new JComboBox(fontOptions[1]);
        chSize= new JComboBox(fontOptions[2]);
        showFont= new JLabel();
        showFont.setHorizontalAlignment( SwingConstants.CENTER );
        showFont.setFont(new Font(curFamily, styleIndex(curStyle),
            curSize));
        showFont.setText(curFamily+" "+curStyle+" "+curSize);
    }
}
```

ComboBoxApp, 3

```
JPanel comboPanel = new JPanel();
comboPanel.add(chFamily);
comboPanel.add(chStyle);
comboPanel.add(chSize);
add( comboPanel, "North" );
add( showFont, "Center" );
chFamily.addActionListener(this);
chStyle.addActionListener(this);
chSize.addActionListener(this);
}

private String curFamily= "Monospaced";
private String curStyle= "NORMAL";
private int curSize= 10;
private JLabel showFont;
private JComboBox chFamily;
private JComboBox chStyle;
private JComboBox chSize;
```

ComboBoxApp, 4

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == chFamily)
        curFamily= (String) chFamily.getSelectedItem();
    else if (e.getSource() == chStyle)
        curStyle= (String) chStyle.getSelectedItem();
    else if (e.getSource() == chSize)
        curSize= Integer.parseInt(
            (String) chSize.getSelectedItem());
    showFont.setFont( new Font( curFamily,
        styleIndex(curStyle), curSize) );
    showFont.setText(curFamily + " " + curStyle + " " + curSize);
}

public int styleIndex(String s) {
    if (s.equals("NEGRITA")) return Font.BOLD;
    else if (s.equals("CURSIVA")) return Font.ITALIC;
    else return Font.PLAIN;
}
```

Tipos de eventos

- **Eventos semánticos frente a eventos de bajo nivel**
 - Los eventos semánticos son generalmente más significativos, a menudo constituyen el resultado de una secuencia de eventos de bajo nivel. Ejemplos:
 - `ActionEvent`: acción del usuario sobre el objeto (clic en el botón).
 - `AdjustmentEvent`: ajuste de valor (barra de desplazamiento).
 - `ItemEvent`: cambio de objeto seleccionable (combo box).
 - `TextEvent`: cambio de valor de texto.
 - Los eventos de bajo nivel anuncian una manipulación directa por parte de los dispositivos de entrada del usuario, ej., ratón, teclado. Ejemplos:
 - `MouseEvent`: ubicar el ratón sobre, retirarlo, mantenerlo pulsado, soltar y hacer clic.
 - `MouseEvent`: movimiento y arrastre del ratón.

Ejemplo de *Mousing* (acción del ratón)

- A. Descargar `Mousing.java` :
- Es necesario que obtenga la ruta del directorio que ha montado. En *Explorer view*, haga clic en la pestaña *FileSystems*. La ruta de este directorio se muestra como enlace en su árbol *Filesystems* (si posee muchos directorios montados, elija el de nivel superior).
 - En su navegador web, vaya a:
web.mit.edu/1.00/www/Lectures/Lecture16/Mousing.java
Guarde el archivo descargado en el directorio montado (haga clic con el botón derecho y seleccione 'Save Link As' (Guardar enlace como))
 - Vuelva a *FileSystems*, en la vista *Explorer*. Haga un clic con el botón derecho sobre el directorio montado y seleccione la opción *refresh* (actualizar). Ahora debería aparecer el directorio recién guardado.
- B. Cree un proyecto llamado `Mousing` y añada el archivo `Mousing.java` (para añadir un archivo a su proyecto, clic derecho en el archivo y seleccione *Tools->Add To Project* (Herramientas-> Añadir al proyecto)).

Ejemplo de Mousing, 2

- C. *Mousing* es un programa muy sencillo que crea un botón que permanece a la escucha tanto del `ActionEvent`, de alto nivel, como de los eventos de bajo nivel del ratón y los muestra todos por pantalla. Lea el código para ver el modo en el que se reúnen.
- D. Podría preguntarse cómo averiguar qué eventos puede generar un componente. Observe qué dice Javadoc sobre `JButton` y busque los métodos `addEventTypeListener`. Cuando encuentre uno (compruebe también las clases básicas), siga el hipervínculo a la interfaz de oyente de evento. Eso enumerará un conjunto de tipos de eventos y rellamadas que el componente puede generar. Cuando haya encontrado todos los métodos `add...Listener`, habrá encontrado todos los eventos.
- E. Compile y ejecute *Mousing*. Experimente con su ratón para determinar exactamente bajo qué condiciones un botón enviará un `ActionEvent`. Por ejemplo, ¿obtiene usted un `ActionEvent` en el momento de pulsar con el ratón sobre un botón? ¿Qué ventajas tiene para usted oír `ActionEvents` sobre oír eventos del ratón de bajo nivel?

Clases internas anónimas

- Java ofrece un atajo para la creación de clases que facilita la escritura de oyentes de eventos.
- Estudiaremos más a fondo este rasgo en la clase 28.
- El `ComboBoxApp` permanece a la escucha de 3 `JComboBoxes` y prueba `ActionEvents` para determinar su origen.
- ¿No sería más fácil si existiese un modo económico de escribir un oyente de acciones independiente para cada uno de ellos?

AnonComboBoxApp

```
class AnonComboPanel
    extends JPanel //implements ActionListener {
    . . .
    //chFamily.addActionListener(this);
    chFamily.addActionListener( new ActionListener() {
        public void actionPerformed( ActionEvent e ) {
            curFamily= (String) chFamily.getSelectedItem();
            showFont.setFont( new Font( curFamily,
                styleIndex(curStyle), curSize) );
            showFont.setText(curFamily + " " + curStyle +
                " " + curSize);
        }
    } );
    //chStyle/chSize.addActionListener(this);
    //no joint actionPerformed() method
```

AnonComboBoxApp, 2

- AnonComboBox ya no implementa a ActionListener ni posee un método actionPerformed.
- Las 3 clases internas anónimas asumen el control de esa función.
- Parece como si estuviéramos creando una interfaz, lo cual sería ilegal. En realidad, estamos creando una clase sin nombre que sólo poseerá una instancia, la que creamos aquí mismo para el método addActionListener().
- La llamada al nuevo constructor NO puede tener argumentos.

```
chFamily.addActionListener(
    new ActionListener() { . . . }
);
```

AnonComboBoxApp, 3

- Debemos facilitar una implementación del método `actionPerformed()` en la clase anónima.
- Éste puede acceder a los miembros estáticos y de instancia de la clase que los contiene, incluso a los privados.

```
chFamily.addActionListener( new ActionListener() {
    public void actionPerformed( ActionEvent e ) {
        curFamily= (String) chFamily.getSelectedItem();
        showFont.setFont( new Font( curFamily,
                                   styleIndex(curStyle), curSize) );
        showFont.setText(curFamily + " " + curStyle +
                        " " + curSize);
    } // fin de actionPerformed
} // fin de ActionListener
);
```

Ejemplo *Clock* (reloj)

- A. Descargue `Clock.java` y `ClockTest.java` y añádalos a un nuevo proyecto llamado *Clock*.
- B. La clase *Clock* define (de forma no sorprendente) un objeto *clock*. Échele un vistazo rápido, pero no pierda mucho tiempo tratando de comprender los detalles. Aquí mostramos un resumen de los métodos:
 - `Clock()`: este es el constructor. Todo lo que necesita saber es que construye la GUI (por ahora no debe preocuparse por los detalles):
 - Instancia dos botones: “*Tick*”, “*Reset*” (hacer tic tac, reajuste).
 - Instancia dos etiquetas (inicializadas a “12” y “00”) para mostrar la hora en modo numérico.
 - Añade los botones y etiquetas al panel.

Ejemplo *Clock* (reloj)

C. (Cont.)

- `paintComponent(Graphics g)`: este método dibuja el reloj y las manecillas de las horas y minutos según el valor de los `minutes` (Una vez más, no se preocupe por los detalles de cómo se hace esto. Examinaremos este tipo de diseño en la próxima clase).
- `tick()`: este método incrementa en uno a `minutes` y luego vuelve a dibujar el reloj.
 - `repaint()` llamará al método *paintComponent* que volverá a dibujar el reloj con las manecillas ajustadas al nuevo valor de `minutes`).
- `reset()`: Este método repone los `minutes` a cero y luego vuelve a dibujar el reloj.
- `getMinutes()`: Método de acceso que devuelve `minutes`.

Ejemplo *Clock* (reloj)

- C. (cont.) Resumiendo, todo lo que necesita saber sobre esta clase (y he aquí las maravillas de la POO) es el comportamiento que dictan sus métodos públicos, es decir, `tick()`, `getMinutes()` y `reset()`, sin tener que preocuparse de cómo se dibujará el reloj.
- D. Clase *TestClock*: amplía a `JFrame`. El método *main* crea una instancia de *TestClock*, la cual lanzará al constructor y creará y añadirá un reloj.
- E. Compile y ejecute su proyecto. Puede observar la GUI, pero de momento, el programa no hará nada. En este ejercicio, vamos a construir el modelo de evento de la GUI para este reloj.

Cómo hacer que el reloj avance

- A. Es necesario que el reloj avance cada vez que el usuario pulse el botón *"Tick"*. Realice los cambios necesarios para que `clock` implemente la interfaz `ActionListener` (observe el ejemplo *Button*). Al implementar `ActionListener`, podría aparecer un cuadro de diálogo, recomendando cambios en la clase. Puede elegir *process All* (procesar todos) para aceptar los cambios, lo que añadirá el método `actionPerformed`, o bien cerrar el diálogo y añadir este método manualmente. Deje el método `actionPerformed` vacío por el momento. Construya su proyecto para asegurarse de que no olvidó nada.
- B. Suscriba el objeto `ActionListener` (`clock`) a `tickButton` (un buen lugar para hacer esto es al final del constructor `clock`).
- C. Escriba ahora los contenidos del método `actionPerformed` para hacer que avancen las manecillas del reloj (esto ocuparía una línea de código). Compile y ejecute. Pruebe el botón *"Tick"*. Debería observar el movimiento de las manecillas del reloj. Las etiquetas no deberían hacer nada todavía.

Reajustar el reloj

- A. Ahora es necesario que el reloj se reajuste a las 12:00 siempre que pulsemos el botón *reset* (reajustar). Suscriba el objeto `ActionListener` a `resetButton`.
- B. Modifique el método `actionPerformed(ActionEvent e)` para que éste pueda distinguir a los eventos procedentes de `resetButton` de los procedentes de `tickButton` (observe el ejercicio de `ComboBox`).
- C. Haga que las manecillas del reloj se vuelvan a posicionar a las 12:00. Compile y ejecute.

Uso de clases internas anónimas

A continuación, reescriba el manipulador de eventos para usar dos clases internas anónimas en vez del método simple `actionPerformed()`. (Observe el ejemplo `AnonComboBox`).

Compile y realice las pruebas.

Cómo mostrar la hora (Opcional)

- A. Queremos mostrar la hora actual en `hourLabel` (etiqueta de las horas) y `minuteLabel` (etiqueta de los minutos). En `Clock`, cree un método con la siguiente firma:

```
public void setLabels()
```

Este método debería hacer lo siguiente:

- Calcular dos variables:
 - número de horas enteras contenidas en los minutos.
 - número de minutos después de quitar el número de horas.
- Mostrar estas dos variables en `hourLabel` y `minuteLabel`. Para mostrar una cadena de caracteres en una etiqueta, debería utilizar `setText(String)` (ej., `hourLabel.setText(String);`)

Pistas: 1. `Integer.toString(int)` toma como argumento un entero y devuelve una cadena.

2. Si `minutes < 60`, la etiqueta hora debería mostrar 12.

3. La etiqueta minuto debería mostrar siempre dos dígitos (ej., 12:08)

- B. Llame a este método desde el lugar adecuado. Compile y ejecute.