**Módulo 13**  Manejo de eventos generados por GUI

# Objetivos

1. Definir eventos y manejo de eventos
2. Examine el modelo de eventos Java SE
3. Describir el comportamiento de la GUI
4. Determinar la acción del usuario que originó un evento
5. Desarrollan oyentes de eventos
6. Describir la simultaneidad en los GUIs basados en Swing y describe las características de la clase SwingWorker

¿Qué es un evento?

1. Eventos:

objetos que describen lo sucedido

1. Fuentes de eventos:

el generador de un evento

1. Controladores de eventos:

método que recibe un objeto de evento, lo descifra y procesa la interacción del usuario

JFrame

JPanel

JButton

El usuario hace clic en el botón

Algún controlador de eventos

actionPerformed(ActionEvent e) {

...

}

ActionEvent

# Modelo de delegación

1. Un evento se puede enviar a muchos controladores de eventos.

JFrame

JPanel

JButton

El usuario hace clic en el botón

Otro controlador de eventos

Un controlador de eventos

actionPerformed(ActionEvent e) {

...

}

actionPerformed(ActionEvent e) {

...

}

ActionEvent

1. Los controladores de eventos se registran con componentes cuando están interesados en los eventos generados por ese componente.

# Modelo de delegación

1. Los objetos de cliente (controladores) se registran con un GUIcomponent que desean observar.
2. Los componentes gui solo desencadenan los controladores para el tipo de evento que se ha producido.
3. La mayoría de los componentes pueden desencadenar más de un tipo de evento.
4. El modelo de delegación distribuye el trabajo entre las clases de información.

# A Listener Example

1. import java.awt.\*;
2. import javax.swing.\*;
3. public class TestButton {
4. private JFrame f;
5. private JButton b;

6

1. public TestButton() {
2. f = new JFrame("Test");
3. b = new JButton("Press Me!");
4. b.setActionCommand("ButtonPressed");
5. }

12

1. public void launchFrame() {
2. **b.addActionListener(new ButtonHandler());**
3. f.add(b,BorderLayout.CENTER);
4. f.pack();
5. f.setVisible(true);
6. }

# A Listener Example

19

1. public static void main(String args[]) {
2. TestButton guiApp = new TestButton();
3. guiApp.launchFrame();
4. } 24 }

El código para el detector de eventos tiene el siguiente aspecto:

1 import java.awt.event.\*;

2

1. public **class ButtonHandler implements ActionListener** {
2. public void **actionPerformed(ActionEvent e)** {
3. System.out.println("Action occurred");
4. System.out.println("Button’s command is: "
5. + e.getActionCommand());
6. }
7. }

# Event Categories

# Method Categories and Interfaces

|  |  |  |
| --- | --- | --- |
| **Category** | **Interface Name** | **Methods** |
| Action | ActionListener | actionPerformed(ActionEvent) |
| Item | ItemListener | itemStateChanged(ItemEvent) |
| Mouse | MouseListener | mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent) |
| Mouse motion | MouseMotionListener mouseDragged(MouseEvent) mouseMoved(MouseEvent) | |
| Key | KeyListener keyPressed(KeyEvent)  keyReleased(KeyEvent) keyTyped(KeyEvent) | |

# Method Categories and Interfaces

**Category Interface Name Methods**

Focus FocusListener focusGained(FocusEvent)

focusLost(FocusEvent)

Adjustment AdjustmentListener adjustmentValueChanged

(AdjustmentEvent)

Component ComponentListener componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)

|  |  |
| --- | --- |
| **Category** | **Interface Name Methods** |
| Window | WindowListener windowClosing(WindowEvent)  windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent) |
| Container | ContainerListener componentAdded(ContainerEvent) componentRemoved (ContainerEvent) |
| Window  state | WindowStateListener windowStateChanged(WindowEvent e) |
| Window  focus | WindowFocusListener windowGainedFocus(WindowEvent e) windowLostFocus(WindowEvent e) |

# Categorías de método e interfaces

Categorías de método e interfaces

**Category Interface Name Methods**

|  |  |  |
| --- | --- | --- |
| Mouse wheel | MouseWheelListener mouseWheelMoved (MouseWheelEvent e) | |
| Input methods | InputMethodListener | caretPositionChanged (InputMethodEvent e) inputMethodTextChnaged (InputMethodEvent e) |
| Hierarchy | HierarchyListener | hierarchyChanged (HierarchyEvent e) |
| Hierarchy bounds | HierarchyBoundsList ener | ancestorMoved(HierarchyEvent e) ancestorResized(HierarchyEvent e) |
| AWT | AWTEventListener | eventDispatched(AWTEvent e) |
| Text | TextListener | textValueChanged(TextEvent) |

# Ejemplo complejo

1. import java.awt.\*;
2. import java.awt.event.\*;
3. import javax.swing.\*;
4. public class TwoListener
5. **implements MouseMotionListener, MouseListener** {
6. private JFrame f;
7. private JTextField tf;

8

1. public TwoListener() {
2. f = new JFrame("Two listeners example");
3. tf = new JTextField(30);
4. }

# Ejemplo complejo

13

1. public void launchFrame() {
2. JLabel label = new JLabel("Click and drag the mouse");
3. Añadir componentes al marco
4. f.add(label, BorderLayout.NORTH);
5. f.add(tf, BorderLayout.SOUTH);
6. Agregue este objeto como un oyente
7. f.addMouseMotionListener(this);
8. f.addMouseListener(this);
9. Dimensione el marco y haga visible
10. f.setSize(300, 200);
11. f.setVisible(true);
12. }

# Complex Example

26

1. // These are MouseMotionListener events
2. public void **mouseDragged**(MouseEvent e) {
3. String s = "Mouse dragging: X = " + e.getX()
4. + " Y = " + e.getY();
5. tf.setText(s);
6. }

33

1. public void **mouseEntered**(MouseEvent e) {
2. String s = "The mouse entered";
3. tf.setText(s);
4. }

38

1. public void **mouseExited**(MouseEvent e) {
2. String s = "The mouse has left the building";
3. tf.setText(s);
4. }

# Complex Example

43

1. // Método MouseMotionListener no utilizado.
2. // All methods of a listener must be present in the46 // class even if they are not used.

47 public void mouseMoved(MouseEvent e) { }

48

1. Métodos MouseListener no utilizados.
2. public void mousePressed(MouseEvent e) { }
3. public void mouseClicked(MouseEvent e) { }
4. public void mouseReleased(MouseEvent e) { }

53

1. public static void main(String args[]) {
2. TwoListener two = new TwoListener();
3. two.launchFrame();
4. }
5. }

# Múltiples oyentes

1. Multiplelistenerscauseunrelatedpartsofaprogrameact al mismo evento.
2. Se llama a los controladores de todos los agentes de escucha registrados cuando se produce el evento.

# Adaptadores de eventos

1. Las clases de agente de escucha que defina pueden extender adapterclasses e invalidar solo los métodos que necesita.
2. Un ejemplo es:
3. import java.awt.\*;
4. import java.awt.event.\*;
5. import javax.swing.\*;

4

5 public class MouseClickHandler extends MouseAdapter { 6

1. Sólo necesitamos el controlador mouseClick, así que usamos
2. un adaptador para evitar tener que escribir todos los
3. métodos de controlador de eventos

10

11 public void mouseClicked(MouseEvent e) { 12 // Hacer cosas con el clic del ratón...

1. }
2. }

# Manejo de eventos mediante clases internas

1. import java.awt.\*;
2. import java.awt.event.\*;
3. import javax.swing.\*;
4. public class TestInner {
5. private JFrame f;
6. private J**TextField tf**; // used by inner class

7

1. public TestInner() {
2. f = new JFrame("Inner classes example");
3. tf = new JTextField(30);
4. }

12

1. class **MyMouseMotionListener extends MouseMotionAdapter** {
2. public void mouseDragged(MouseEvent e) {
3. String s = "Mouse dragging: X = "+ e.getX()
4. + " Y = " + e.getY();
5. **tf.setText(s)**;
6. }
7. }

# Manejo de eventos mediante clases internas

20

1. public void launchFrame() {
2. JLabel label = new JLabel("Click and drag the mouse");
3. Añadir componentes al marco
4. f.add(label, BorderLayout.NORTH);
5. f.add(tf, BorderLayout.SOUTH);
6. Agregue un agente de escucha que use una clase Inner
7. f.addMouseMotionListener(**new MyMouseMotionListener()**);
8. f.addMouseListener(new MouseClickHandler());
9. // Dimensione el marco y haga visible
10. f.setSize(300, 200);
11. f.setVisible(true);
12. }

33

1. public static void main(String args[]) {
2. TestInner obj = new TestInner();
3. obj.launchFrame();
4. }
5. }

# Manejo de eventos mediante clases anónimas

1. import java.awt.\*;
2. import java.awt.event.\*;
3. import javax.swing.\*;

4

1. public class TestAnonymous {
2. private JFrame f;
3. private JTextField tf;

8

1. public TestAnonymous() {
2. f = new JFrame("Anonymous classes example");
3. tf = new JTextField(30);
4. }

13

1. public static void main(String args[]) {
2. TestAnonymous obj = new TestAnonymous();
3. obj.launchFrame();
4. }

18

# Manejo de eventos mediante clases anónimas

1. public void launchFrame() {
2. JLabel label = new JLabel("Click and drag the mouse");
3. Añadir componentes al marco
4. f.add(label, BorderLayout.NORTH);
5. f.add(tf, BorderLayout.SOUTH);
6. Agregue un agente de escucha que use una clase anónima
7. f.addMouseMotionListener(**new MouseMotionAdapter() {**
8. public void mouseDragged(MouseEvent e) {
9. String s = "Mouse dragging: X = "+ e.getX()
10. + " Y = " + e.getY();
11. tf.setText(s);
12. }
13. **}**); **// <- note the closing parenthesis**
14. f.addMouseListener(new MouseClickHandler()); No se muestra
15. Dimensione el marco y haga visible
16. f.setSize(300, 200);
17. f.setVisible(true);
18. }
19. }

# Simultaneidad en swing

Para manejar una INTERFAZ GRÁFICA de usuario de manera eficiente, el programa Swing necesita diferentes hilos para:

1. Ejecutar el código de aplicación (subprocesos actuales)
2. Controlar los eventos que surgen de la GUI (subprocesos eventdispatch)
3. Controlar tareas en segundo plano que pueden ser timeconsuming (subprocesos de trabajo)

Cada tarea de un subproceso de trabajo está representada por una instancia de javax.swing.SwingWorker.

# La clase SwingWorker

La clase SwingWorker tiene métodos para cumplir los siguientes requisitos:

1. Proporcionar comunicación y coordinación entre las tareas de subprocesos de trabajo y las tareas en otros subprocesos:
2. Propiedades: estado y progreso • Para ejecutar tareas en segundo plano sencillas:
3. método doInBackground
4. Para ejecutar tareas que tengan resultados intermedios:
5. método de publicación
6. Para cancelar los subprocesos en segundo plano:
7. cancelar el método