Tratamento de eventos

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

Eventos

Eventos

Correspondem à interações com os componentes.

- Cada componente responde a tipos de eventos diversos.
- Ouvintes de eventos (*Listeners*) são os objetos notificados quando um evento ocorre.
- Para que um componente ou contêiner possa responder a eventos, é necessário instalar um *Listener*.
- Eventos são derivados de java.awt.event.AWTEvent.
- Eventos e *Listeners* são disponibilizados pelos pacotes java.awt.event.* e javax.swing.event.*.

Eventos

- Eventos de baixo nível compreendem:
  - eventos de contêiner (inserção ou remoção de componente).
  - eventos de foco (componente ganhou ou perdeu foco).
  - eventos de entrada: teclado e mouse.
  - eventos de janela: resize, open, close, minimize, etc...
- Eventos semânticos compreendem:
  - eventos de ação: notificam a ação de um componente específico. Ex: botão “clicado”.
  - eventos de ajuste: a barra de rolagem foi ajustada.
  - eventos de item: um elemento de lista, *radio button* ou *checkbox* foi alterado.
  - eventos de texto: indica alteração de um texto em um JTextArea ou JTextField.

Fontes de eventos do AWT

Listener	Métodos	Componentes
ActionListener	actionPerformed()	AbstractButton, Button, ButtonModel, ComboBoxEditor, JComboBox, JFileChooser, JTextField, List, MenuItem, TextField, Timer
AdjustmentListener	Changed()	JScrollBar, Scrollbar
ComponentListener	componentHidden(), componentMoved(), componentResized(), componentShown()	Component
ContainerListener	componentAdded(), componentRemoved()	Container
FocusListener	focusGained(), focusLost()	Component
ItemListener	itemStateChanged()	AbstractButton, ButtonModel, Checkbox, CheckboxMenuItem, Choice, ItemSelectable, JComboBox, List
KeyListener	keyPressed(), keyReleased(), keyTyped()	Component
MouseListener	mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()	Component
MouseMotionListener	mouseDragged(), mouseMoved()	Component
TextListener	textValueChanged()	TextComponent
WindowListener	windowActivated(), windowClosed(), windowClosing(), windowDeactivated(), windowDeiconified(), windowIconified(), windowOpened()	Window

Fontes de eventos do Swing (1/2)

Listener	Métodos	Componentes
AncestorListener	ancestorAdded(), ancestorMoved(), ancestorRemoved()	Action, JComponent
CaretListener	caretUpdate()	JTextComponent
CellEditorListener	editingCanceled(), editingStopped()	CellEditor,
ChangeListener	stateChanged()	AbstractButton, BoundedRangeModel, ButtonModel, JProgressBar, JSlider, JTabbedPane, JViewport, MenuSelectionManager, SingleSelectionModel
HyperlinkListener	hyperlinkUpdate()	JEditorPane
InternalFrameListener	internalFrameActivated(), internalFrameClosed(), internalFrameClosing(), internalFrameDeactivated(), internalFrameDeiconified(), internalFrameIconified() internalFrameOpened()	
ListDataListener	contentsChanged(), intervalAdded(), intervalRemoved()	AbstractListModel, ListModel
ListSelectionListener	valueChanged()	JList, ListSelectionModel

Fontes de eventos do Swing (2/2)

Listener	Métodos	Componentes
MenuDragMouseListener	menuDragMouseDragged(), menuDragMouseEntered(), menuDragMouseExited(), menuDragMouseReleased()	JMenuItem
MenuKeyListener	menuKeyPressed(), menuKeyReleased(), menuKeyTyped()	JMenuItem
MenuListener	menuCanceled(), menuDeselected(), menuSelected()	JMenu
PopupMenuListener	popupMenuCanceled(),	JPopupMenu
	popupMenuWillBecome-	
	Invisible(),	
	popupMenuWillBecome-	
	Visible()	
TreeExpansionListener	treeCollapsed(), treeExpanded()	JTree
TreeSelectionListener	valueChanged()	JTree
TreeWillExpandListener	treeWillCollapse(), treeWillExpand()	JTree
java.beans.-	propertyChange()	Action, JComponent,
PropertyChangeListener		UIDefaults, UIManager
java.beans.-	vetoableChange()	JComponent
VetoableChangeListener		

Eventos: exemplo

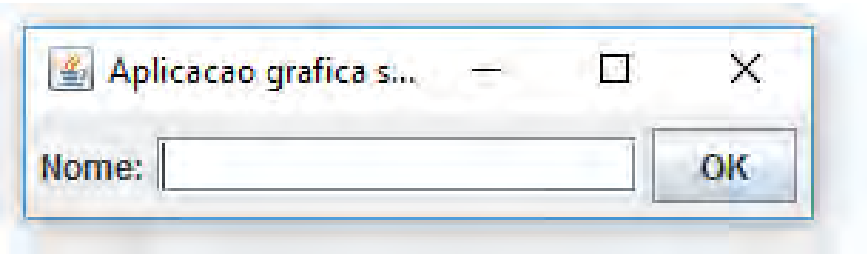
```
public class JanelaGrafica extends JFrame {
  private static final long serialVersionUID = 1L;
  private JButton butOK;
  private JTextField campo;
  private JLabel texto;

  public JanelaGrafica() {
    super("Aplicacao grafica simples");

    Container cPane = this.getContentPane();

    // Define layout do container
    cPane.setLayout(new FlowLayout());

    // Cria os componentes
    texto = new JLabel("Nome:");
    campo = new JTextField(15);
    butOK = new JButton("OK");
```

Eventos: exemplo

```
        // Adiciona os componentes a janela ou container
        cPane.add(texto);
        cPane.add(campo);
        cPane.add(butOK);

         this.addWindowListener(new AppListener());

        // Adciona um listener para o botão de OK
        butOK.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            texto.setText(campo.getText());
            // Copia conteúdo digitado para o label
          }
        });

        // Ajusta tamanho/janela conforme os componentes
        this.pack();
    }
}
```

Eventos: exemplo

```
private class AppListener extends WindowAdapter {
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
}

public class AplicacaoGrafica {
  public static void main(String args[]) {
    JanelaGrafica apg = new JanelaGrafica();
    apg.setVisible(true);
  }
}
```



Exercícios

- Construa um programa em Java que solicita o nome, o sexo, a idade e o estado civil de uma pessoa e, quando o botão de OK é clicado, ele instancia um objeto da classe Pessoa e limpa os campos da tela.
- Implemente um método `toString()` na classe Pessoa que exibe os dados pessoais da pessoa.
- Utilize o `JOptionPane` para exibir a pessoa cadastrada.
- Se o usuário clicar o botão CANCEL os campos são limpos mas a frase não é criada.

Tratamento de eventos centralizado

- Utiliza-se o evento semântico “ação realizada” (*action performed*).
- Um único objeto implementa a interface ActionListener.
- Esse objeto é registrado nos diversos componentes da interface:
- O método **public void** actionPerformed(ActionEvent e) é implementado uma única vez e deve identificar qual componente disparou o evento.

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

Exemplo: evento centralizado

```
public class JanelaGrafica extends JFrame implements ActionListener {
    private JButton butOk = new JButton("Ok");
    private JButton butCancel = new JButton("Cancel");
    private JTextField campo = new JTextField(15);
    private JLabel texto = new JLabel("Nome:");

    public JanelaGrafica() {
        super("Aplicacao grafica simples");

        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container cPane = this.getContentPane();
        cPane.setLayout(new FlowLayout());

        // Tratamento de eventos centralizado
        butOk.addActionListener(this);
        butCancel.addActionListener(this);

        cPane.add(texto);          cPane.add(campo);
        cPane.add(butOk);          cPane.add(butCancel);
        this.pack();
    }
```

Exemplo: evento centralizado

```
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Item: " + e.getActionCommand());
        String componente = e.getActionCommand();
        if (componente.equals("Ok")) {
            JOptionPane.showMessageDialog(this,
                    "Bem vindo, " + campo.getText() + ".",
                    "Boas vindas", JOptionPane.PLAIN_MESSAGE);
        } else {
            campo.setText("");
        }
    }
}
```

Evento centralizado

- Principais problemas:
  - não é escalável, pois concentra a complexidade do tratamento de eventos.
  - o texto como único mecanismo de identificação de um campo pode ser problemático.
- É importante conhecer, pois é utilizado.

Abordagem das classes aninhadas

- Pode-se definir classes internas para tratar eventos.
- Classes internas têm seu uso restrito à classe em que foi definida.
- Classes internas têm acesso aos membros privados da classe em que foi definida.
- Compilador cria a classe de nome `ClasseExterna$ClasseInterna.class`

Exemplo: classes internas

```
public class JanelaGrafica extends JFrame {
    private JButton butOk = new JButton("Ok");
    private JButton butCancel = new JButton("Cancel");
    private JTextField campo = new JTextField(15);
    private JLabel texto = new JLabel("Nome:");

    public JanelaGrafica() {
        super("Aplicacao grafica simples");

        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container cPane = this.getContentPane();
        cPane.setLayout(new FlowLayout());

        // Tratamento de eventos com classe aninhadas
        butOk.addActionListener(new ButOkEvent());
        butCancel.addActionListener(new ButCancelEvent());

        cPane.add(texto);          cPane.add(campo);
        cPane.add(butOk);          cPane.add(butCancel);
        this.pack();
    }
```

Exemplo: classes internas

```
    class ButOkEvent implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(null,
                    "Bem vindo, " + campo.getText() + ".",
                    "Boas vindas", JOptionPane.PLAIN_MESSAGE);
        }
    }

    class ButCancelEvent implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            campo.setText("");
        }
    }
}
```

Classes internas anônimas

- Nomear classes internas pode ser um problema em janelas complexas.
- Classes internas anônimas seguem a mesma regra das classes nomeadas.
- Por não possuírem nome, não podem ser reaproveitadas.
- Por não possuírem nome, sua instância é referenciada pelo seu supertipo (tipo da classe pai).

```
new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Faz alguma coisa
    }
}
```

Exemplo: classes internas anônimas

```
public class JanelaGrafica extends JFrame {
   private JButton butOk = new JButton("Ok");
   private JButton butCancel = new JButton("Cancel");
   private JTextField campo = new JTextField(15);
   private JLabel texto = new JLabel("Nome:");

   public JanelaGrafica() {
      super("Aplicacao grafica simples");

      this.setDefaultCloseOperation(EXIT_ON_CLOSE);
      Container cPane = this.getContentPane();
      cPane.setLayout(new FlowLayout());

      cPane.add(texto);          cPane.add(campo);
      cPane.add(butOk);          cPane.add(butCancel);
```

Exemplo: classes internas anônimas

```
        // Classes aninhadas anonimas
        butOk.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null,
                        "Bem vindo, " + campo.getText() + ".",
                        "Boas vindas", JOptionPane.PLAIN_MESSAGE);
            }
        });

        butCancel.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                campo.setText("");
            }
        });

        this.pack();
    }
}
```

Classes Adaptadoras

- Uma classe que implementa um ouvinte de eventos precisa implementar todos os métodos desse *Listener*.
- Classes adaptadoras (*Adapters*) fornecem uma implementação vazia dos *Listeners*, que pode ser aproveitada.

```
class WindowAdapter implements WindowListener {
    public void windowClosing(WindowEvent e) {}
    public void windowOpened(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowClosed(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
}
```

Exercício

- Faça uma calculadora simples, sem teclado numérico, apenas com os botões das 4 operações: adição, subtração, multiplicação e divisão.
- Usuário deve digitar o primeiro operando.
- Ao clicar no botão do operador, o 1° operando deve ser armazenado e o campo deve ficar limpo para receber o segundo operando.
- Ao clicar igual, a operação deve ser realizada e o resultado deve ser exibido.