Java First-Tier: Aplicações

Construção de Interface gráfica

Grupo de Linguagens de Programação
Departamento de Informática
PUC-Rio

Processo Básico: OO + Eventos

- Instanciar os componentes de interface
  - por exemplo, janelas, botões, campos de textos, etc
- Adicionar os componentes em containers
  - por exemplo, como os componentes podem ser agrupados e qual o layout de diagramação
- Estabelecer o tratamento de eventos de interface
  - por exemplo, o que deve ocorrer quando o usuário clicar um botão ou como alterar o conteúdo de um componente quando um outro sofre alguma alteração

2

Exemplo

3

Exemplo

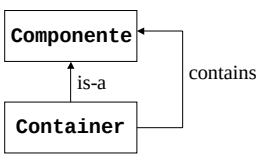
```
import java.awt.event.*;
import javax.swing.*;

public class JanelaSimples {
  public JanelaSimples() {
    /* Cria o botão */
    final JButton botaoLimpa = new JButton("Limpa");
    /* Cria o campo de texto */
    final JTextField campoTexto = new JTextField(10);
    /* Cria uma janela */
    final JFrame janela = new JFrame ("Janela Simples");
    janela.setSize(300,100);
    /* Adiciona os componentes na janela */
    JPanel painel = new JPanel();;
    painel.add (botaoLimpa);
    painel.add (campoTexto);
    janela.getContentPane().add(painel);
    /* Quando o usuário clicar no botao, limpa o campo de texto */
    botaoLimpa.addActionListener (new ActionListener() {
      public void actionPerformed (ActionEvent e) {
        campoTexto.setText("");
      }});
    /* Exibe a janela */
    janela.setVisible(true);
  }
}
```

4

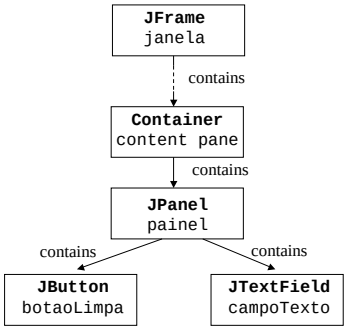
Hierarquia de Composição

- **Componente**
  - qualquer elemento de interface
- **Container**
  - é um **Componente**
  - agrega **Componentes**



5

Exemplo



6

Elementos de Interface Swing

- Uma janela é um ***top-level container***: onde os outros componentes são desenhados.
- Um painel é um ***container* intermediário**: serve para facilitar o agrupamento de outros componentes.
- Botões e campos de texto são exemplos de **componentes atômicos***:* elementos de interface que não agrupam outros componentes. Mostram para o usuário e/ou obtêm dele alguma informação. A API do Swing oferece muitos componentes atômicos.

7

Alguns Componentes Atômicos

- Campo de Texto
- *Label*
- Botão
- *CheckBox*
- Lista
- Lista *dropdown*
- *Canvas*

8

Classe **JComponent**

- Superclasse de muitos elementos do Swing,
- Funcionalidade comum aos componentes
- Disponibiliza métodos para controlar:
  - *Tool tips*
  - Bordas
  - Propriedades

9

javax.swing.JLabel

- Modela um texto e/ou imagem não editável, isto é, sem interação com o usuário

10

Exemplo de **JLabel**

```
/* Cria um label com texto */
JLabel label1 = new JLabel("Label1: Apenas Texto");

/* Cria um label com texto e imagem */
JLabel label2 = new JLabel("Label2: Imagem e texto",
                    new ImageIcon("javalogo.gif"),
                    JLabel.CENTER);
label2.setVerticalTextPosition(JLabel.BOTTOM);
label2.setHorizontalTextPosition(JLabel.CENTER);
```

11

javax.swing.JTextField

- Modela um campo de edição de texto de uma linha

12

Exemplo de **JTextField**

```
/* Cria um campo de nome */
JTextField campoNome = new JTextField(10);
JLabel labelNome = new JLabel ("Nome: ");
labelNome.setLabelFor (campoNome);
labelNome.setDisplayedMnemonic('n'); // Alt-n

/* Cria um campo de email */
JTextField campoEmail = new JTextField(10);
JLabel labelEmail = new JLabel ("Email: ");
labelEmail.setLabelFor (campoEmail);
labelEmail.setDisplayedMnemonic('E'); // Alt-e
```

13

javax.swing.JButton

- Modela um *push-button*

14

Exemplo de **JButton**

```
/* Cria um botao com texto */
JButton botao1 = new JButton ("Botão Desabilitado");
botao1.setEnabled(false);
botao1.setToolTipText("Exemplo de um botão de texto");
botao1.setMnemonic(KeyEvent.VK_D);  // Alt-D

/* Cria um botao com texto e imagem */
JButton botao2 = new JButton("Botão Habilitado", new
   ImageIcon("javalogo.gif"));
botao2.setToolTipText("Botão de texto e imagem");
botao2.setMnemonic(KeyEvent.VK_H);  // Alt-H
botao2.setPressedIcon(new ImageIcon("javalogo2.gif"));
```

15

Alguns Containers

- **Janela**
- **Diálogo**
- **Applet**

} Top Level Containers

- **Painel**
- **Scroll Pane**

} Containers Intermediários

16

javax.swing.JFrame

- Representa uma janela do sistema nativo
- Possui título e borda
- Pode possuir menu

17

Exemplo

```
JFrame janela = new JFrame("Exemplo de Janela");
janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JLabel mensagem = new JLabel("Olá Mundo!");
janela.getContentPane().add(mensagem);
janela.setLocationRelativeTo(null); // centraliza
janela.setIconImage(new
   ImageIcon("javalogo2.gif").getImage());
JMenuBar menuBar = new JMenuBar();
menuBar.add(new JMenu("Menu"));
janela.setJMenuBar (menuBar);
janela.pack();
janela.show();
```

18

javax.swing.JPanel

- Modela um *container* sem decoração.
- Representa um grupo de elementos
- Normalmente usado para estruturar a interface
  - associado a um diagramador

19

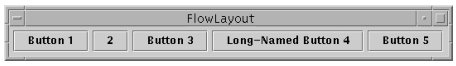
Diagramadores

- Arrumam um grupo de elementos
- Estão associados aos *containers*
- Diferentes estilos de arrumação
  - como fluxo de texto
  - orientada pelas bordas
  - em forma de grade
  - e outros...

20

java.awt.FlowLayout

- Coloca os componentes lado a lado, uma linha após a outra
- Alinhamento: centralizado (*default*), à esquerda ou à direita
- *Default* para **o JPanel**



21

Exemplo de FlowLayout

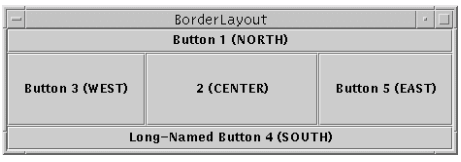
```
Container contentPane = janela.getContentPane();
contentPane.setLayout(new FlowLayout());

contentPane.add(new JButton("Button 1"));
contentPane.add(new JButton("2"));
contentPane.add(new JButton("Button 3"));
contentPane.add(new JButton("Long-Named Button 4"));
contentPane.add(new JButton("Button 5"));
```

22

java.awt.BorderLayout

- Divide o *container* em 5 áreas: norte, sul, leste, oeste e centro
- *Default* para ***content pane*** **do JFrame**



23

Exemplo de BorderLayout

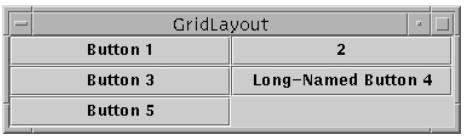
```
Container contentPane = janela.getContentPane();
//contentPane.setLayout(new BorderLayout()); // Desnecessário

contentPane.add(new JButton("Button 1 (NORTH)"),
   BorderLayout.NORTH);
contentPane.add(new JButton("2 (CENTER)"),
   BorderLayout.CENTER);
contentPane.add(new JButton("Button 3 (WEST)"),
   BorderLayout.WEST);
contentPane.add(new JButton("Long-Named Button 4 (SOUTH)"),
   BorderLayout.SOUTH);
contentPane.add(new JButton("Button 5 (EAST)"),
   BorderLayout.EAST);
```

24

java.awt.GridLayout

- Células do mesmo tamanho especificadas pelo número de linhas e colunas



25

Exemplo de GridLayout

```
Container contentPane = janela.getContentPane();

contentPane.setLayout(new GridLayout(0,2));

contentPane.add(new JButton("Button 1"));
contentPane.add(new JButton("2"));
contentPane.add(new JButton("Button 3"));
contentPane.add(new JButton("Long-Named Button 4"));
contentPane.add(new JButton("Button 5"));
```

26

Orientação por Eventos

27

Orientação por Eventos

Um modelo de programação que tornou-se bastante difundido com o uso de interfaces gráficas foi a *programação orientada por eventos*. Segundo esse modelo, o programa deixa de ter o controle do fluxo de execução, que passa a um sistema encarregado de gerenciar a interface. Assim, o programa passa a ser chamado pelo sistema quando algum *evento* é gerado na interface.

28

Mecanismos de *Callback*

Para que o programa possa ser chamado pelo sistema, ele deve registrar uma função para cada evento de interface que ele desejar tratar. Tais funções são chamadas de *callbacks* por serem 'chamadas de volta' pelo sistema.

29

Callbacks OO?

Esse modelo é ortogonal ao modelo de orientação por objetos. É perfeitamente possível projetar um sistema OO que use o modelo de orientação por eventos para tratar eventos de interface, comunicações, etc. Porém, temos um problema: uma linguagem puramente OO não possui o conceito de *função*. Como resolver então?

30

Objeto Callback

A solução é utilizar um *objeto* que faça o papel de *callback*. Ou seja, onde registraríamos uma função, passamos a registrar um objeto. Assim, quando o sistema precisar executar a *callback*, ele deverá executar um determinado método do objeto. Esse método, então, fará o tratamento do evento.

31

Callbacks em Java

- Como Java é uma linguagem OO na qual não existe o conceito de *função*, *callbacks* devem ser implementadas através de objetos
- Um objeto que implementa uma *callback* em Java é chamado de *listener*

32

Listeners & Eventos

Os *listeners* fazem o papel das *callbacks*. *Listeners* são definidos por interfaces e podem estar aptos a tratar mais de um tipo de evento. Quando um *listener* tem um de seus métodos chamados, ele recebe um parâmetro que descreve o evento ocorrido. Esse parâmetro é um objeto: existem classes para modelar diferentes grupos de eventos.

33

Listeners

- Definem interfaces que representam um grupo de *callbacks*
- São registrados junto aos componentes
- Exemplo: **java.awt.event.MouseListener**

```
public abstract void mouseClicked(MouseEvent e)
public abstract void mousePressed(MouseEvent e)
public abstract void mouseReleased(MouseEvent e)
public abstract void mouseEntered(MouseEvent e)
public abstract void mouseExited(MouseEvent e)
```

34

ActionListener

- Modela a *callback* de um evento do tipo **ActionEvent**.

```
public abstract void actionPerformed (ActionEvent e)
```

35

WindowListener

- Modela a *callback* de um evento do tipo **WindowEvent**. Essa interface declara um método para cada evento do grupo

```
public abstract void windowOpened(WindowEvent e)
public abstract void windowClosing(WindowEvent e)
public abstract void windowClosed(WindowEvent e)
public abstract void windowIconified(WindowEvent e)
public abstract void windowDeiconified(WindowEvent e)
public abstract void windowActivated(WindowEvent e)
public abstract void windowDeactivated(WindowEvent e)
```

36

Eventos

- Trazem informações sobre o evento ocorrido
- São passados aos *listeners* (*callbacks*)
- Exemplo: **java.awt.event.MouseEvent**

```
public int getX()
public int getY()
public int getClickCount()
```

37

ActionEvent

- Modela o evento que representa uma ação executada sobre um componente.
- O significado depende do componente.
- Exemplos:
  - "botão pressionado"
  - "entrada de texto finalizada"
  - …

38

WindowEvent

- Modela os eventos que podem ocorrer em uma janela. Essa classe declara constantes que identificam os diversos eventos

```
public static final int WINDOW_OPENED
public static final int WINDOW_CLOSING
public static final int WINDOW_CLOSED
public static final int WINDOW_ICONIFIED
public static final int WINDOW_DEICONIFIED
public static final int WINDOW_ACTIVATED
public static final int WINDOW_DEACTIVATED

public Window getWindow()
```

39

Implementando um Listener

- Define a classe que implementa a interface do Listener

```
class MeuListener implements ActionListener {

  public void actionPerformed(ActionEvent e) {
    System.out.println("Botão pressionado");
  }

}
```

40

Registrando um Listener

- Cria o objeto de callback e o registra no componente

```
button.addActionListener(new MeuListener());
```

41

Implementando um Listener de Janela

Para criarmos um *listener* para um evento de janela devemos criar uma classe que implemente a interface **WindowListener**. Nessa classe, codificaremos, então, o método correspondente ao evento que desejamos tratar. Porém, não podemos implementar uma interface e deixar de codificar algum método. Assim, precisaremos implementar todos os sete métodos definidos.

42

Implementando um Listener de Janela

```
class MeuListener implements WindowListener {
  public void windowOpened(WindowEvent e) { }
  public void windowClosing(WindowEvent e) {
    System.out.println("Janela sendo fechada");
  }
  public void windowClosed(WindowEvent e) { }
  public void windowIconified(WindowEvent e) { }
  public void windowDeiconified(WindowEvent e){ }
  public void windowActivated(WindowEvent e){ }
  public void windowDeactivated(WindowEvent e) { }
}
```

43

Adaptadores

No caso anterior, seis implementações seriam vazias pois só desejávamos responder a um único evento. Como essa é uma situação comum, o pacote **event** define *adaptadores* para todas as interfaces de *listeners* que têm mais de um método. Um adaptador é uma classe que implementa o *listener* e dá implementações vazias para todos os métodos.

44

Adaptadores

```
class MeuListener extends WindowAdapter {
  public void windowClosing(WindowEvent e) {
    System.out.println("Janela sendo fechada");
  }
}
```

45

Uso de *local inner classes*

- Classe que implementa a interface pode ser:
  - aninhada dinamicamente
  - local ao método que faz o registro do listener
  - anônima

```
button.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    System.out.println("Botão pressionado");
  }
});
```

46

Exemplo I

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

public class Janela1 {
  JButton botaoLimpa;
  JTextField campoTexto;
  JFrame janela;
  public Janela1() {
    botaoLimpa = new JButton("Limpa");
    campoTexto = new JTextField(10);
    janela = new JFrame ("Exemplo 1 de Listener");
    janela.setSize(300,100);
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);;
    JPanel painel = new JPanel();;
    painel.add (botaoLimpa); painel.add (campoTexto);
    janela.getContentPane().add(painel);
    botaoLimpa.addActionListener (new LimpaCampoListener(campoTexto));
    janela.setVisible(true);
  }
  public static void main(String[] args) {
    new Janela1();
  }
}
class LimpaCampoListener implements ActionListener {
  JTextField campo;
  LimpaCampoListener(JTextField campo) {
    this.campo = campo;
  }
  public void actionPerformed(ActionEvent e) {
    campo.setText("");
  }
}
```

47

Exemplo II

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

public class Janela2 {
  JButton botaoLimpa;
  JTextField campoTexto;
  JFrame janela;
  private class LimpaCampoListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      campoTexto.setText("");
    }
  }
  public Janela2() {
    botaoLimpa = new JButton("Limpa");
    campoTexto = new JTextField(10);
    janela = new JFrame ("Exemplo 2 de Listener");
    janela.setSize(300,100);
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);;
    JPanel painel = new JPanel();;
    painel.add (botaoLimpa); painel.add (campoTexto);
    janela.getContentPane().add(painel);
    botaoLimpa.addActionListener (new LimpaCampoListener());
    janela.setVisible(true);
  }
  public static void main(String[] args) {
    new Janela2();
  }
}
```

48

Exemplo III

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

public class Janela3 {
  JButton botaoLimpa;
  JTextField campoTexto;
  JFrame janela;
  public Janela3() {
    botaoLimpa = new JButton("Limpa");
    campoTexto = new JTextField(10);
    janela = new JFrame ("Exemplo 3 de Listener");
    janela.setSize(300,100);
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);;
    JPanel painel = new JPanel();;
    painel.add (botaoLimpa); painel.add (campoTexto);
    janela.getContentPane().add(painel);
    botaoLimpa.addActionListener (new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        campoTexto.setText("");
      }
    });
    janela.setVisible(true);
  }
  public static void main(String[] args) {
    new Janela3();
  }
}
```

49

Exemplo IV

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

public class Janela4 {
  public Janela4() {
    JButton botaoLimpa = new JButton("Limpa");
    final JTextField campoTexto = new JTextField(10);
    JFrame janela = new JFrame ("Exemplo 4 de Listener");
    janela.setSize(300,100);
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);;
    JPanel painel = new JPanel();;
    painel.add (botaoLimpa); painel.add (campoTexto);
    janela.getContentPane().add(painel);
    botaoLimpa.addActionListener (new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        campoTexto.setText("");
      }
    });
    janela.setVisible(true);
  }

  public static void main(String[] args) {
    new Janela4();
  }
}
```

50