Construção de Interfaces Gráficas

Mário Antonio Meireles Teixeira
DEINF – UFMA

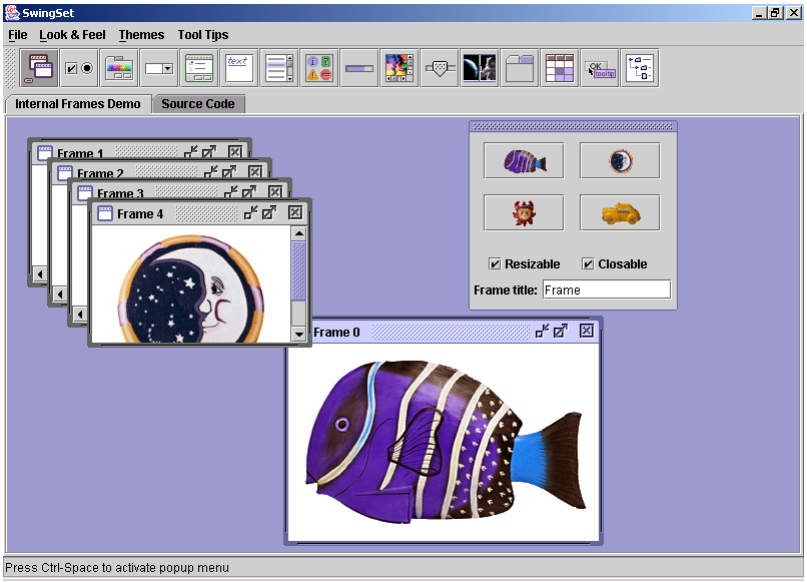
Baseado em material original de
João Carlos Pinheiro – CEFET/MA

1

Objetivos

- Apresentar
  - Os fundamentos da construção GUI em Java
  - Modelo de Eventos AWT
  - Biblioteca de Componentes Swing

2



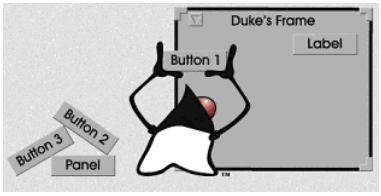
Introdução

- Interface gráfica é composta de:
  - **Componentes**
  - **Recipientes (Containers)**
  - **Layout**
  - **Eventos**

4

Recipientes e Componentes

- Componentes, controles ou widgets são os aspectos visíveis de uma GUI, como botões, menus, caixas de textos
  - São colocados dentro de **recipientes** (**containers**).
- Os recipientes podem conter:
  - Um ou mais componentes, assim como outros recipientes
    - Importante para construção de *layouts* de complexidade realista



5

AWT versus Swing

- A Abstract Window Toolkit (AWT) é o antigo conjunto de ferramentas para interfaces gráficas do Java
  - **Oferece** uma infra-estrutura mínima de interface gráfica
    - Componentes têm aparência **dependente** de plataforma
  - **Limitado** em recursos devido a depender de suporte de cada plataforma para os componentes oferecidos
  - **Bugs** e **incompatibilidades** entre plataformas

6

AWT versus Swing

- O swing não é um substituto completo do AWT, mas fornece um conjunto muito mais **rico** e **conveniente** de componentes para interface com usuário
- O Swing é menos sujeito a erros específicos de cada plataforma

7

Evolução das AWT

- **Java 1.0**
  - Modelo de eventos arcaico baseado em interfaces
- **Java 1.1**
  - **Melhora do modelo de eventos**: por delegação usando "design pattern" Observer
- **Java 1.2**
  - Swing substitui totalmente componentes AWT
  - **Mantém** e **estende** a interface de **eventos** e **layout** da versão 1.1

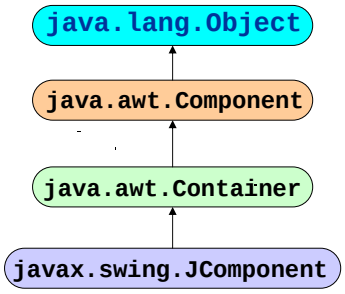
8

Tipos de Aplicativos Graficos

- Há dois tipos de aplicações gráficas em Java
  - ***Applets*** *executam dentro de um navegador Web*
  - **Aplicações *standalone*** iniciadas via sistema operacional
- Ambas capturam eventos do sistema e desenham-se sobre um **contexto gráfico** fornecido pelo sistema
- **Applets** são aplicações especiais que executam a partir de um browser
  - Browser é quem controla seu ciclo de vida (início, fim, etc.)
  - Geralmente ocupam parte da janela do browser, mas podem abrir janelas extras
  - Possuem restrições de segurança

9

Principais classes de GUI



10

Classe `java.awt.Component`

- **Raiz** da hierarquia de componentes gráficos
- Existe um Component por trás de tudo que pode ser pintado na tela
- Principais métodos (chamados pelo sistema)
  - `void paint(java.awt.Graphics g)`
  - `void repaint()`
  - `void update(java.awt.Graphics g)`
  - `Dimension getSize()` – retorna o tamanho de um componente. O tipo de retorno é um objeto da classe `Dimension`, que possui dois atributos públicos `height` e `width`
  - `void setSize(int, int)` – Altera o tamanho de um componente

11

Class `java.awt.Component`

- `getLocation()` e `setLocation(int x, int y)` – estes métodos **alteram a localização** dos componentes (em pixels) relativa ao canto superior esquerdo do componente
  - O **tipo de retorno** de `getLocation()` é Dimension
- `setForeground()` e `setBackground()` – altera as cores dos componentes
  - Os argumentos são uma instância de `java.awt.Color`

12

Class `java.awt.Component`

- **`setFont()`** – define a fonte que será usada para renderizar o texto que será apresentado no componente
  - Este método recebe três argumentos: fonte, estilo e o tamanho
- **`setEnable(boolean)`** – habilita ou desabilita um componente
  - Este método é útil para criar interfaces que reflitam o estado atual da aplicação do usuário

13

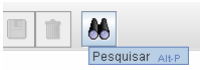
`java.awt.Container`

- **Subclasse** de **`java.awt.Component`**
- São "recipientes". Podem conter outros componentes
- Descendentes da classe Container:
  - **`Frame`**, **`Panel`**, **`Applet`** e
  - **`JComponent`** (raiz da hierarquia dos componentes Swing)
- Para adicionar um componente em um Container (**`Panel`**, **`Frame`**, **`Applet`**) utiliza-se o método **`add(Component c)`**

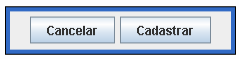
14

javax.swing.JComponent

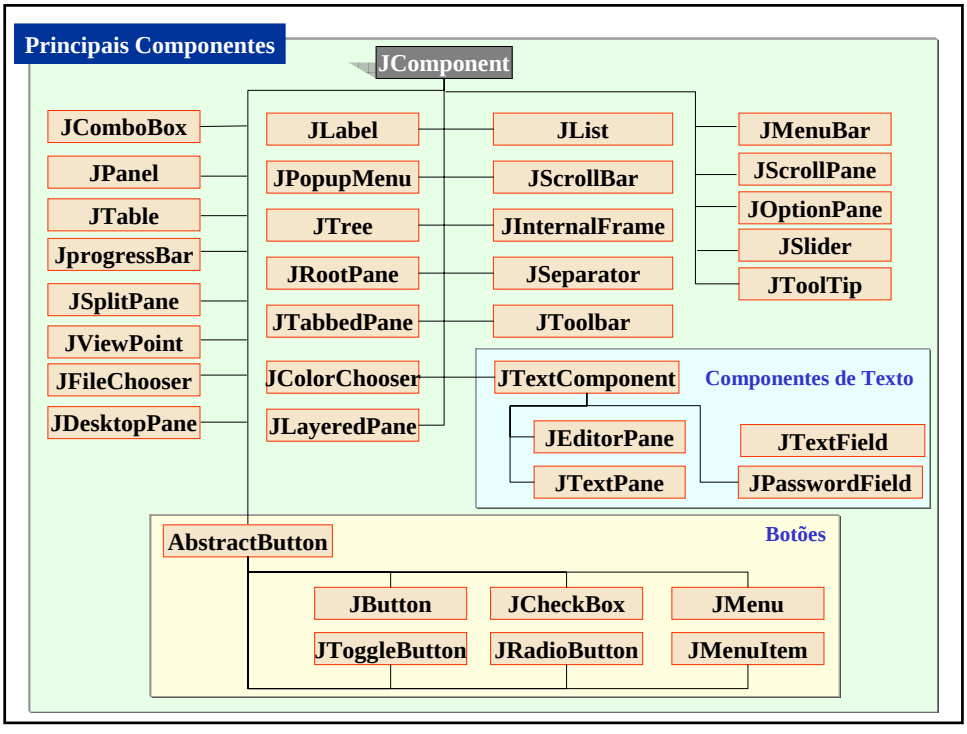
- Fornece recursos de alto nível a todos os componentes swing
  - Tamanho preferencial **`setPreferredSize()`**
  - Suporte a acessibilidade e internacionalização
  - Suporte a dicas quando o mouse permanece sobre um componente



  - Definição de bordas



15

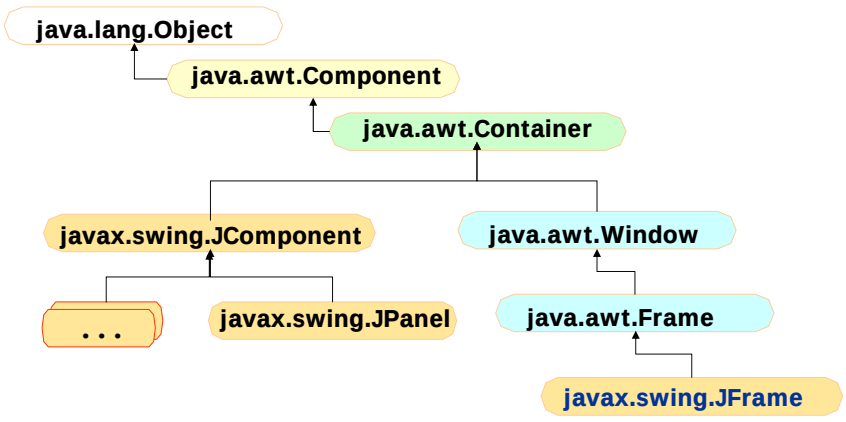


Frames (Molduras)

- É uma janela independente que pode ser movimentada na tela independente de quaisquer outras janelas de GUI
- Java fornece duas classes para este fim:
  - `java.awt.Frame`, subclasse da classe `java.awt.Window` e
  - `javax.swing.JFrame`, subclasse de `java.awt.Frame`

17

Hierarquia de Herança da Classe JFrame



18

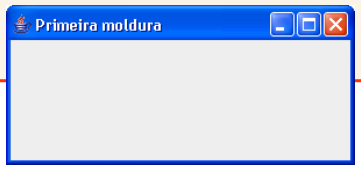
Frames (JFrame)

- O seguinte código exibe um JFrame vazio:

```
import javax.swing.JFrame;

public class Moldura {

  public static void main(String[] args) {
       JFrame f = new JFrame("Primeira moldura");
       f.setSize(300, 250);
       f.setVisible(true);

  }
}
```

Aparece no título da moldura

É necessário, pois um frame recém construído não está visível na tela



19

Adicionando componentes ao JFrame

- Até o JDK 1.5 era necessário obter um Container para adicionar os componentes

```
JFrame frame = new JFrame("Teste");
Container cont = frame.getContentPane();
JPanel pan = new JPanel();
pan.add(new JButton("Button"));
cont.add(pan);
```

É necessário para manter a independência de plataforma

- A SUN corrigiu este inconveniente possibilitando a verificação especial deste recurso em tempo de execução

```
JFrame frame = new JFrame("Teste");
JPanel pan = new JPanel();
pan.add(new JButton("Button"));
frame.add(pan);
```

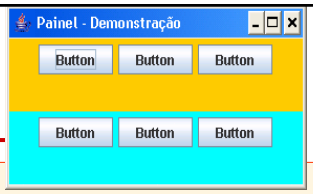
20

Panel e JPanel

- É um recipiente de uso geral. Ele não pode ser usado isoladamente, como frames e caixas de diálogo
- Painéis podem administrar eventos

21

Painel - Exemplo




```
public class PanelDemo {
    public static void main(String[] args) {
        JFrame f = new JFrame("Content Pane Demo");
        f.setSize(350, 250);
        Container cont = f.getContentPane();
        cont.setLayout(new GridLayout(2,1));

        for (int i=0; i<2; i++) {
           JPanel pan = new JPanel();
           pan.setBackground(i==0 ? Color.ORANGE:Color.CYAN);

           for (int j=0; j<3; j++)
               pan.add(new JButton("Button"));
           cont.add(pan);
        }
        f.setVisible(true);
    }
}
```

Tratamento de Eventos

Tratamento de Eventos

- Esta seção apresenta **como lidar** com eventos ou "ações" que ocorrem dentro de ambientes GUI
- Interfaces gráficas são **guiadas por eventos**
  - Exemplo: movimento do mouse, pressionar um botão, entrada de dados em um textfield, etc.
- Em java **eventos são objetos** que descrevem o que aconteceu
- Eventos em Java são uma instância de uma subclasse de **`java.util.EventObject`**

Fornece o método **`getSource()`** que retorna a fonte do evento

Tratamento de Eventos

- O modelo de eventos se torna fácil de usar, pois é baseado em uma nomenclatura simples:
  - Para cada tipo de evento **XXXEvent** existe uma interface **XXXListener** (interface "ouvinte"):
    - Os métodos desta interfaces são todos **public void** e recebem um único argumento do tipo **XXXEvent** (um objeto)

```
ActionEvent: ActionListener
ItemEvent  : ItemListener
TextEvent  : TextListener
WindowEvent: WindowListener
KeyEvent   : KeyListener
        . . .
XXXEvento  : XXXListener
```

25

Tratamento de Eventos

- Nomenclatura de eventos:
  - Se um componente emite eventos de tipo **XXXEvent**, então ele tem um método público chamado **addXXXListener(XXXListener)**
    - Quando um componente é ativado de uma maneira que o leve a emitir eventos XXX, todos os receptores registrados recebem uma chamada para um dos métodos da interface receptora

Qualquer objeto que implementa esta interface pode ser **registrado** como um receptor de eventos XXX

26

Tratamento de Eventos

- Nomenclatura de eventos:
  - Além do método **addXXXListener** (**XXXListener**), um componente que emite eventos, também tem um método público chamado **`removeXXXListener(XXXListener)`**
    - Possibilita que os receptores que não estejam mais interessados em eventos **XXX** desfaçam o registro

27

Modelo de Tratamento de Eventos

- Pode ser dividido em três partes
  1. **Origem do evento (fonte)**
     - É o resultado de alguma ação do usuário em um componente GUI
     - Exemplo: o clique de mouse em um componente irá gerar (source) um **`ActionEvent`**

28

Modelo de Tratamento de Eventos

29

Modelo de Tratamento de Eventos

2. **Objeto que contem informações sobre evento (`ActionEvent`)**
   - Encapsula informações sobre o evento que ocorreu
     - **Exemplo**: um objeto da Classe `ActionEvent`
   - Todo **evento** tem um objeto que é sua **fonte**
     - `Object fonte = evento.getSource();`

30

Modelo de Tratamento de Eventos

3. Listener (Ouvinte)

- Recebe eventos como argumento

```
public class NomeClasse implements ActionListener {
 public void actionPerformed(ActionEvent evt) {
    Object fonte = evt.getSource();
    System.out.println(evt + "em" + fonte);
 }
}
```

31

Modelo de Tratamento de Eventos

- Programador deve realizar duas tarefas
  1. **Registrar** um ouvinte para cada evento que se deseja tratar
  2. **Implementar** o(s) método(s) que manipulam o evento

32

Como ligar a fonte ao Ouvinte

- Na ocorrência de um evento, em uma fonte, todos os "ouvintes" registrados serão notificados.
  - `fonte.add<Listener>` (referência_para_Listener);
- O mesmo objeto que é fonte às vezes também é listener, se implementar as interfaces:
  - Ainda assim, é necessário registrar a fonte ao listener
  - `this.addWindowListener(this);`

Fonte

Ouvinte

Confira: `ButtonTest.java`

33

Descrição dos Eventos

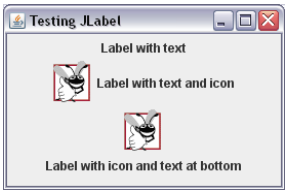
- Descendentes de **`java.awt.event.AWTEvent`**
- Divididos em categorias (**`java.awt.event`**)
  - **ActionEvent** (fonte: componentes de ação (**`JButton`**, **`JTextField`**, ...))
  - **MouseEvent** (fonte: componentes afetados pelo mouse)
  - **ItemEvent** (fonte: componentes de seleção (**`JCheckBox`**, **`JRadioButton`**, **`JComboBox`**, ...))
  - **AdjustmentEvent** (fonte: scrollbars)
  - **TextEvent** (fonte: componentes de texto)
  - **WindowEvent** (fonte: janelas)
  - **FocusEvent** (fonte: componentes em geral)
  - **KeyEvent** (fonte: componentes afetados pelo teclado)

34

Alguns componentes
(subclasse de JComponent)

JLabel

- São strings de texto usadas para identificar outros elementos. Normalmente são inseridas a esquerda ou acima do elemento que estar sendo rotulado
- Pode apresentar:
  - Somente Texto
  - Imagens
  - Texto e imagens



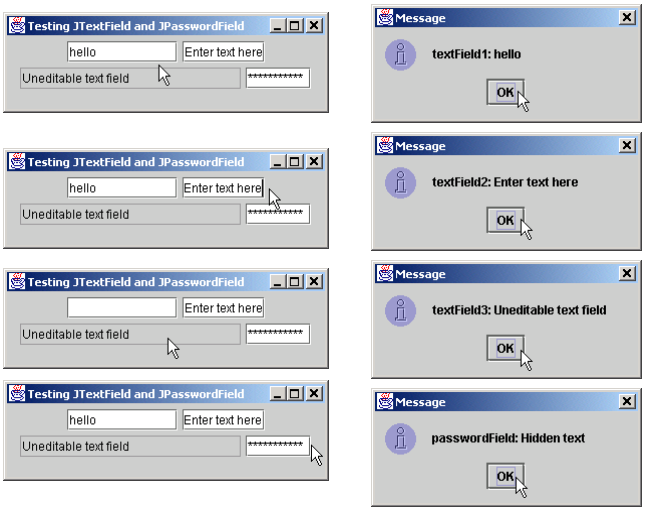
Confira: `LabelTest.java`

JTextField e JPasswordField

- É um dispositivo de linha única de texto
  - Em virtude de ser possível apenas uma linha, um `ActionListener` pode ser informado – através de `actionPerformed()` – quando a **tecla enter for pressionada**
  - Este componente **não exibe barra de rolagem**, mas é possível rolar sobre o texto longo da esquerda para a direita, se for necessário
- A classe `JPasswordField` herda de `JTextField` e adiciona vários métodos que são específicos ao processamento de senhas

37

JTextField e JPasswordField



Confira: `TextFieldTest.java`

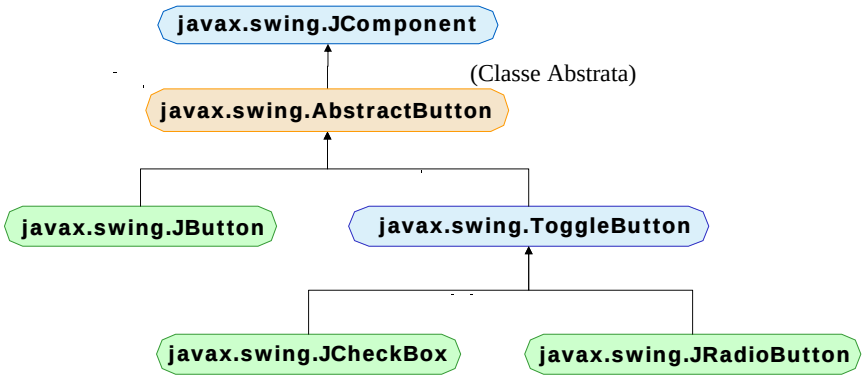
38

Botões

- São elementos de forma retangular que o usuário utiliza para iniciar ações
- O pacote swing define vários tipos de botões
  - **botões de comando**
  - **Check boxes (caixa de seleção)**
  - **Radio buttons**
  - **Todos estes botões são subclasse de `javax.swing.AbstractButton`**

39

JButton



40

JButton

- São criados com a classe **JButton**
  - Quando o usuário clica no botão é gerado um **ActionEvent**

  - O método **actionPerformed(ActionEvent evt)** de qualquer "ouvinte" registrado é chamado quando o botão for pressionado

Confira novamente: **ButtonTest.java**

41

JCheckBox (Caixa de Seleção)

- Fornece um dispositivo simples de entrada "liga/desliga" com um rótulo de texto ao lado
- A seleção ou o cancelamento de um checkbox é notificado pela interface **ItemListener** que define o método:
  - **public void itemStateChange(ItemEvent e)**
  - Um objeto da classe **ItemEvent** possui um método **getStateChange()**, que retorna uma das duas constantes:
    - **ItemEvent.SELECTED**
    - **ItemEvent.DESELECTED**

Confira:
CheckBoxTest.java

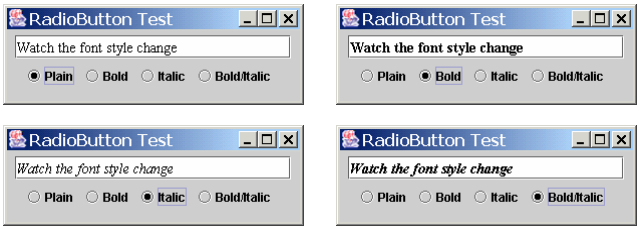
42

JRadioButton e ButtonGroup

- Os botões de opção normalmente aparecem como um grupo em que apenas um botão de opção pode ser selecionado
- O relacionamento lógico entre botões de um grupo é mantido por um objeto **`ButtonGroup`**
  - O objeto **`ButtonGroup`** em si não é um componente GUI, pois não é exibido em uma interface com o usuário
  - O mesmo **pode ser utilizado** como **parâmetro no construtor** de um **`JRadioButton`**

43

JRadioButton (Botões de Opção)

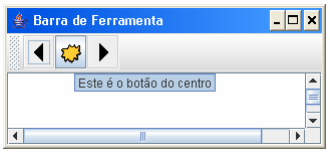


Confira: RadioButtonTest.java

44

JToolTip (Dica)

- É uma string de texto que fornece uma dica. Aparece automaticamente quando o mouse permanece sobre o componente
  - Pode ser configurado para qualquer **JComponent**
  - **É necessário apenas chamar o método setToolTipText() no componente**
    - **Ex: `comp.setToolTipText("dica");`**



45

JComboBox (Escolhas)

- Fornece uma lista de itens entre os quais o usuário pode escolher
- **JComboBox** gera **ItemEvent**, assim como, **JCheckBox** e **JRadioButton**
- Uso de **classe interna anônima** para o tratamento de eventos

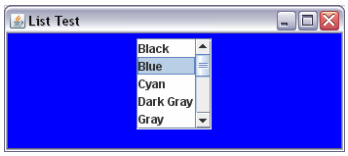
Confira: **ComboBoxTest.java**

46

JList

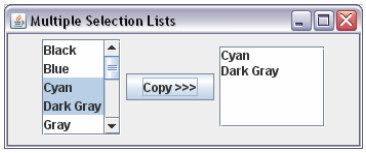
- Exibe uma série de itens em que o usuário pode selecionar um ou mais itens
- Este componente suporta
  - Listas de uma única seleção:

Confira: **`ListTest.java`**



  - Lista de seleção múltipla:

Confira: **`MultipleSelection.java`**



47

Tratamento de Eventos do Mouse

- Interfaces ouvintes de eventos do mouse
  - **`MouseListener`** e **`MouseMotionListener`**
- Os eventos do mouse podem ser capturados por qualquer componente GUI que herda de **`javax.swing.JComponent`**
- Todos os métodos "ouvintes" aceitam um objeto **`MouseEvent`** como seu argumento

48

Tratamento de Eventos do Mouse (MouseEvent)

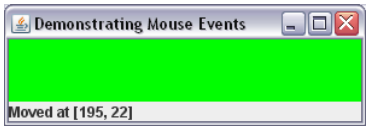
Métodos (Ouvintes) da interface **MouseListener**	
mouseClicked(MouseEvent e)	Quando o botão do mouse for pressionado e liberado
mousePressed(MouseEvent e)	Quando o botão do mouse for pressionado
mouseReleased(MouseEvent e)	Quando o botão do mouse for liberado (precedido por um evento mousePressed)
mouseEntered(MouseEvent e)	Quando o cursor do mouse entra nos limites de um componente
mouseExited(MouseEvent e)	Quando o cursor do mouse deixa os limites de um componente

49

Tratamento de Eventos do Mouse (MouseEvent)

Métodos (Ouvintes) da interface **MouseMotionListener**	
mouseDragged(MouseEvent)	Quando o botão do mouse for pressionado e movido (precedido por uma chamada mousePressed)
mouseMoved(MouseEvent)	Quando o cursor do mouse for movido nos limites de um componente

Confira: **`MouseTracker.java`**



50

Classes Adaptadoras

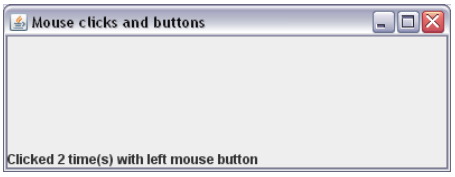
- Padrão de Projeto chamado Adapter
- Algumas interfaces possuem uma **classe adaptadora** que implementa todos os seus métodos, com um corpo vazio:
  - Só existem para interfaces que possuem mais de um método
  - Pode-se herdar de uma classe adaptadora e fornecer uma implementação (*sobrescrever*) somente o método necessário para o tratamento do evento

51

Classes Adaptadoras

- O nome da classe adaptadora é semelhante ao do Listener :
  - MouseListener: MouseAdapter
  - MouseMotionListener: MouseMotionAdapter
  - KeyListener: KeyAdapter
  - WindowListener: WindowAdapter ...

`MouseDetails.java`

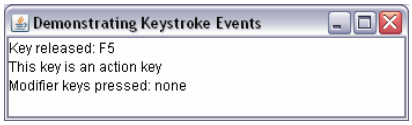


52

Tratamento de Eventos do Teclado

- Uma classe que implementa KeyListener deve fornecer as definições para os métodos:
  - KeyPressed
  - KeyReleased
  - KeyTyped
- Cada uma das classes recebe um KeyEvent como argumento
- A classe trata seus próprios eventos de teclado usando o método addKeyListener

Confira: **KeyDemo.java**



53

Referências

- Deitel & Deitel. Java como Programar, 6ª ed. Caps. 11 e 22;
- Materiais do site da SUN (http://java.sun.com)

54