



BCC221

Programação Orientada a Objetos

Prof. Marco Antonio M. Carvalho

2013/1



Na aula passada

UFOP

- Polimorfismo
 - Exemplo
- Classes e métodos abstratos
 - Exemplo Polimorfismo e Downcast
 - Resolução Dinâmica
 - Operador *instanceof*
 - *Downcast*
 - Método *getclass*
- Métodos e Classes *final*
- Criando e Utilizando Interfaces
- Declarando Constantes em Interfaces
- Introdução ao Java *Reflection API*
- Interfaces Comuns da API Java



Na aula de hoje

UFOP

- Caixas de Diálogo
- Componentes Swing
- Swing vs. AWT
- Componentes Leves e Pesados
- Exibindo Textos e Imagens em uma Janela
- Especificando o Leiaute
- Criando Labels
- Campos de Texto e Introdução à Manipulação de Eventos
- Tipos Comuns de Eventos e Listener Interfaces
- Botões
 - Botões que Mantêm seu Estado
- Caixas de Marcação
- Botões de Opção
- Caixas de Combinação
- Listas
 - Listas de Seleções Múltiplas
- Manipulação de Eventos do Mouse
- Classes Adaptadoras
- Subclasse JPanel para Desenhar com o Mouse
- Manipulação de Eventos do Teclado
- Gerenciadores de Leiaute
- Áreas de Texto



Componentes da Interface Gráfica

UFOP

- Uma **interface gráfica** ou **GUI** (*Graphical User Interface*) fornece um mecanismo amigável para interagirmos com uma aplicação
 - *Look and Feel*;
 - Permite que os usuários se familiarizem com a aplicação e a utilizem de maneira mais produtiva.
- Uma GUI é formada por **componentes**
 - Barra de título (*title bar*);
 - Barra de menu (*menu bar*);
 - Menus;
 - Botões (*buttons*);
 - Caixas de Combinação (*Combo boxes*);
 - Barras de rolagem (*scroll bars*)
 - Etc.



Componentes da Interface Gráfica

UFOP

- Os componentes de interfaces gráficas são chamados às vezes de **controles** ou **widgets** (*window gadgets* – geringonça, dispositivo);
- Um componente de uma GUI é um objeto com o qual o usuário interage via mouse, teclado ou outra forma de entrada
 - Por exemplo, reconhecimento de voz.

Caixas de Diálogo



Caixas de Diálogo

UFOP

- A interface gráfica mais simples é a **caixa de diálogo**
 - Janelas que os programas utilizam para informações importantes ou para entrada de dados.
- A classe ***JOptionPane*** (pacote ***javax.swing***) fornece caixas de diálogo para entrada e saída.



Caixas de Diálogo

UFOP

```
import javax.swing.JOptionPane;

public class Addition {
    public static void main( String args[] )
    {
        String firstNumber; //primeira string digitada pelo usuário
        String secondNumber; //segunda string digitada pelo usuário
        int number1;         //primeiro número
        int number2;         //segundo número
        int sum;             //soma

        //lê o primeiro número como uma string
        firstNumber = JOptionPane.showInputDialog("Enter first integer");

        //lê o segundo número como uma string
        secondNumber = JOptionPane.showInputDialog("Enter second integer");
```



Caixas de Diálogo

UFOP

```
// converte os números de String para int
number1 = Integer.parseInt(firstNumber);
number2 = Integer.parseInt(secondNumber);

// adiciona os numeros
sum = number1 + number2;

// mostra o resultado
JOptionPane.showMessageDialog(null, "The sum is " + sum, "Results",
                           JOptionPane.PLAIN_MESSAGE);

System.exit( 0 );
}
```



Constantes *JOptionPane*

UFOP

JOptionPane Icons in Java Look and feel :



Error Message



Information Message



Question Message



Warning Message

ERROR_MESSAGE	INFORMATION_MESSAGE
QUESTION_MESSAGE	WARNING_MESSAGE
PLAIN_MESSAGE	

Componentes Swing



Componentes Swing

- Apesar de ser possível realizar operações de entrada e saída usando caixas de diálogo, geralmente as interfaces gráficas são mais elaboradas
 - Customizadas.
- Para criar interfaces gráficas em Java, podemos utilizar os componentes ***Swing***
 - Pacote ***javax.swing***.
- A maioria dos componentes ***Swing*** são componentes Java puros
 - Ou seja, escritos, manipulados e exibidos completamente em Java
 - Maior portabilidade.
 - Parte da ***Java Foundation Classes*** (JFC).



Componentes Swing

UFOP

Componente	Descrição
<i>JLabel</i>	Exibe texto ou ícones que não podem ser editados.
<i>JTextField</i>	Permite que o usuário entre com dados a partir do teclado. Também pode ser utilizado para exibir texto editável ou não.
<i>JButton</i>	Dispara um evento quando acionado.
<i>JCheckBox</i>	Especifica uma opção que pode ser selecionada ou não.
<i>JComboBox</i>	Fornece uma lista drop-down de elementos em que o usuário pode selecionar um elemento ou possivelmente digitar no campo adequado.
<i>JList</i>	Fornece uma lista de ítems em que o usuário pode selecionar um ítem clicando nele. Múltiplos ítems podem ser selecionados.
<i>JPanel</i>	Fornece uma área em que os componentes podem ser colocados e organizados. Pode também ser utilizado como uma área para desenhar gráficos.

Swing vs. AWT



Swing vs. AWT

- Existem basicamente dois conjuntos de componentes de interfaces gráficas em Java
 - AWT (*Abstract Window Toolkit*)
 - Pacote `java.awt`;
 - O sistema operacional renderiza os componentes;
 - Mesma aplicação com “cara” diferente em diferentes plataformas.
 - Swing (Java SE 1.2)
 - Renderiza por conta própria os componentes;
 - “Caras” iguais em plataformas diferentes.
- A aparência e a forma em que o usuário interage com a aplicação são chamados de *look and feel* da aplicação.

Componentes Leves e Pesados



Componentes Leves e Pesados

UFOP

- A maioria dos componentes Swing não são “amarrados” à plataforma em que a aplicação é executada
 - Tais componentes são chamados de **leves**.
- Os componentes AWT (muitos dos quais paralelos aos componentes Swing) são amarrados ao sistema de janelas da plataforma local
 - Componentes **pesados**;
 - Dependem da plataforma local para determinar a funcionalidade e o *look and feel*.



Componentes Leves e Pesados

UFOP

- Alguns dos componentes Swing também são componentes pesados
 - Também requerem interação com o sistema de janelas local;
 - Podem ter a funcionalidade e a aparência restritos pode isso;
 - Menos flexíveis que os componentes leves.



Componentes Leves e Pesados

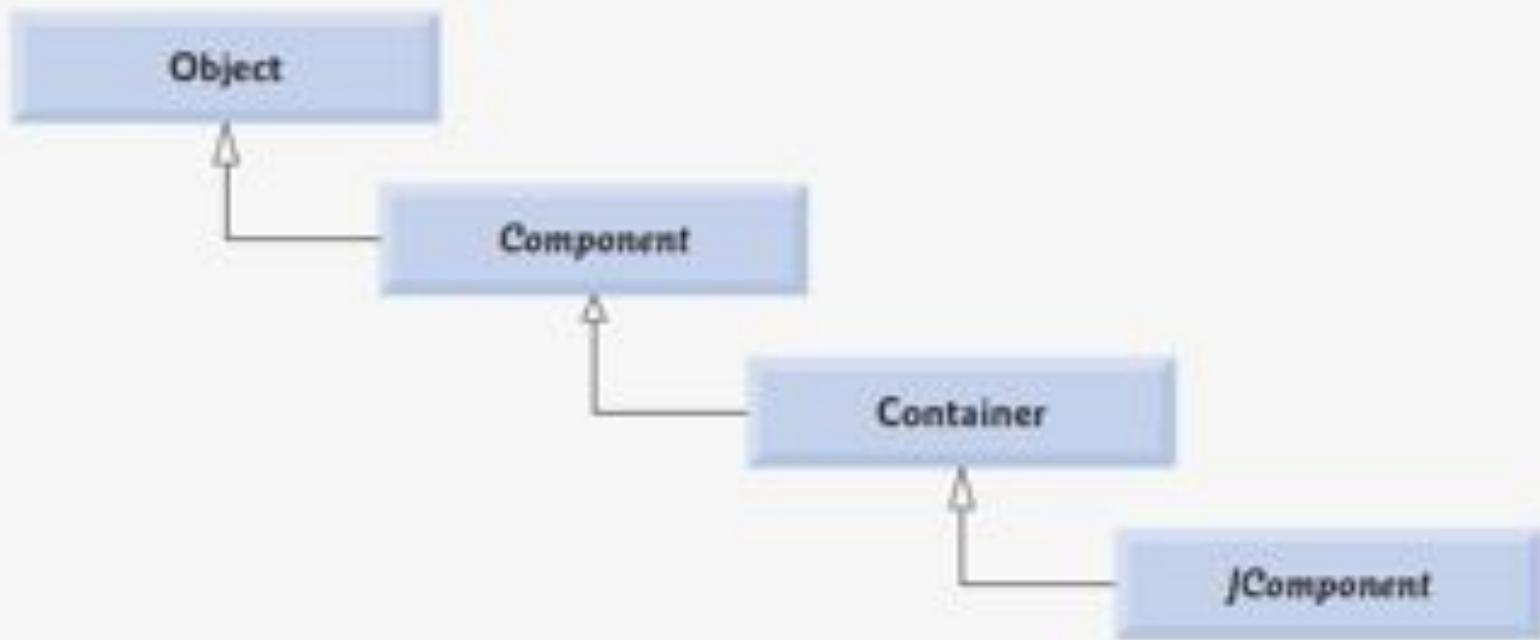
UFOP

- O diagrama UML a seguir apresenta a hierarquia de herança contendo as classes das quais os componentes leves Swing herdam atributos e comportamento
 - O topo da hierarquia é representado pela classe Object;
 - As classes ***Component***, ***Container*** e ***Jcomponent*** completam a hierarquia.



Hierarquia dos Componentes Leves

UFOP





Classe *Component*

UFOP

- Pacote `java.awt`;
- Declara muitos dos atributos e comportamentos comuns aos componentes GUI nos pacotes `java.awt` e `javax.swing`
 - A maioria dos componentes GUI herda direta ou indiretamente desta classe.
- Lista completa em

<http://download.oracle.com/javase/7/docs/api/java/awt/Component.html>



Classe *Container*

- Pacote `java.awt`;
- Como veremos a seguir, os componentes (*Components*) são associados a contêineres (*Container*) para poderem ser organizados e exibidos na tela;
- Qualquer objeto que é **um *Container*** pode ser utilizado para organizar outros *Components*
 - Como um *Container* é **um *Component***, podemos associar um *Container* a outro.
- Maiores informações

<http://download.oracle.com/javase/7/docs/api/java.awt/Container.html>



Classe *JComponent*

- Pacote `java.swing`;
- É a superclasse de todos os componentes leves e declara os atributos e comportamentos comuns a eles;
- Por ser uma subclasse de *Container*, todos os componentes leves são também contêineres;
- Mais informações em

<http://download.oracle.com/javase/7/docs/api/javax/swing/JComponent.html>



Classe *JComponent*

- Algumas das características dos componentes leves suportados pela classe *Jcomponent* incluem:
 - Plugabilidade do *look and feel*
 - Customização de componentes para uso em plataformas particulares.
 - Teclas de atalho para acesso aos componentes;
 - Capacidade de manipular eventos em casos em que existem ações padronizadas;
 - *Tooltips* (dicas);
 - Suporte a tecnologias de acessibilidade;
 - Suporte a regionalização
 - Língua e outras convenções culturais.

Exibindo Textos e Imagens em uma Janela

Exibindo Textos e Imagens em uma Janela



UFOP

- Uma interface consiste de vários componentes
 - Em uma interface grande, pode ser difícil identificar o propósito de cada um dos componentes.
- Pode ser fornecido um texto com instruções sobre o componente
 - Tal texto é conhecido como *label* (ou rótulo)
 - Criado pela classe **JLabel**, uma subclasse da *Jcomponent*;
 - Um *JLabel* exibe uma linha de texto não editável, uma imagem ou ambos.

Exibindo Textos e Imagens em uma Janela



UFOP

- O exemplo a seguir mostra diferentes *labels*
 - Também introduz uma espécie de *framework* usado para criar as interfaces dos próximos exemplos.



LabelFrame.java

UFOP

```
import java.awt.FlowLayout; // especifica como os componentes são organizados
import javax.swing.JFrame; // fornece as funcionalidade básicas de janelas
import javax.swing.JLabel; // exibe texto e imagem
import javax.swing.SwingConstants; // constantes comuns utilizadas no Swing
import javax.swing.Icon; // interface utilizada para manipular imagens
import javax.swing.ImageIcon; // carrega as imagens

public class LabelFrame extends JFrame
{
    private JLabel label1; // JLabel só com texto
    private JLabel label2; // JLabel contruí-do com texto e ícone
    private JLabel label3; // JLabel com texto e ícone adicionados
```



LabelFrame.java

UFOP

```
// o construtor adiciona JLabels ao JFrame
public LabelFrame()
{
    //construtor só com string
    super( "Testing JLabel" );
    setLayout( new FlowLayout() ); // ajusta o layout do frame

    label1 = new JLabel( "Label with text" );
    label1.setToolTipText( "This is label1" );
    add( label1 ); // adiciona o label1 ao JFrame

    //construtor com string, icone e alinhamento
    Icon bug = new ImageIcon( getClass().getResource( "bug1.gif" ) );
    label2 = new JLabel( "Label with text and icon", bug,
        SwingConstants.LEFT );
    label2.setToolTipText( "This is label2" );
    add( label2 ); // adiciona o label1 ao JFrame
```



LabelFrame.java

UFOP

```
label3 = new JLabel(); //construtor sem argumentos
label3.setText( "Label with icon and text at bottom" );
label3.setIcon( bug ); //adiciona um ícone ao JLabel
label3.setHorizontalTextPosition( SwingConstants.CENTER );
label3.setVerticalTextPosition( SwingConstants.BOTTOM );
label3.setToolTipText( "This is label3" );
add( label3 ); //adiciona um label ao JFrame
}
}
```

Exibindo Textos e Imagens em uma Janela



UFOP

- No exemplo, a classe *LabelFrame* herda as características de janela da classe **JFrame**
 - Normalmente, os construtores das subclasses da JFrame criam a interface que estará na janela;
 - Um dos construtores da JFrame define o conteúdo da barra de título da janela.



LabelTest.java

UFOP

```
import javax.swing.JFrame;

public class LabelTest
{
    public static void main( String args[] )
    {
        LabelFrame labelFrame = new LabelFrame(); // cria um LabelFrame
        labelFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        labelFrame.setSize( 275, 180 ); // ajusta o tamanho do frame
        labelFrame.setVisible( true ); // exibe o frame
    }
}
```

Exibindo Textos e Imagens em uma Janela



UFOP

- A classe *LabelTest* cria um objeto da classe *LabelFrame*
 - Especifica a ação a ser tomada na operação de fechamento da janela
 - Método *setDefaultCloseOperation()*
 - Constante *JFrame.EXIT_ON_CLOSE*;
 - O programa termina quando a janela é fechada;
 - O método *setSize* determina o tamanho da janela;
 - O método *setVisible*, com argumento *true* faz com que a janela seja exibida.

Exibindo Textos e Imagens em uma Janela



UFOP

Arquivo LabelTest.java

Especificando o Leiaute



Especificando o Leiaute

- Quando criamos uma GUI, cada componente deve ser anexado a um contêiner
 - Como a janela criada com o JFrame;
- Além disso, deve ser especificado onde posicionar cada um dos componentes
 - Existem vários gerenciadores de leiaute (*layout managers*) que nos ajudam nesta tarefa.
- Muitas IDEs fornecem ferramentas de criação de interfaces que permitem especificar visualmente o tamanho e a localização de cada componente
 - Usando o mouse;
 - O código é gerado pela IDE.



Especificando o Leiaute

- Diferentes IDEs geram diferentes códigos
 - Podem ser utilizados gerenciadores de leiaute como alternativa;
 - Por exemplo, o *FlowLayout*.
- Em um *FlowLayout*, os componentes são inseridos da esquerda para a direita
 - Na ordem em que o programa os anexa ao contêiner;
 - Se não houver mais espaço horizontal, passa-se para a linha debaixo;
 - Se o contêiner for redimensionado, os componentes são rearranjados
 - Possivelmente, usando menos linhas.

Criando Labels



Criando Labels

UFOP

- Uma vez definido o layout de uma janela, podemos anexar componentes
 - Se não anexarmos um componente explicitamente, ele não será exibido quando o contêiner for exibido;
 - No nosso exemplo, os componentes foram labels.
- Depois de criarmos um novo JLabel, para anexá-lo utilizamos o método ***add***
 - Da classe Container.
- Podemos utilizar o método ***setToolTipText*** para que seja exibida uma dica quando o mouse for posicionado em cima do label
 - O uso de ***tooltips*** é uma boa prática.



Criando Labels

UFOP

- Um objeto *ImageIcon* representa um ícone
 - No exemplo, associamos o figura bug1.gif ao ícone;
 - *Icon* é uma interface implementada pela classe *ImageIcon*;
 - Utilizamos os métodos *getClass* e *getResource* para obtermos a localização da imagem como uma URL;
 - O construtor *ImageIcon* utiliza esta URL para carregar a imagem para a memória.



Criando Labels

- Um *JLabel* pode ser utilizado para exibir o ícone criado previamente
 - Há uma sobrecarga do construtor que recebe como argumentos um texto, um ícone, e uma constante de alinhamento;
- A interface ***SwingConstants*** (pacote `javax.swing`) define diferentes constantes para o alinhamento de componentes
 - Por padrão, o texto aparece à direita de uma imagem em um label;
- O alinhamento horizontal e vertical de um *label* podem ser definidos através dos métodos ***setHorizontalAlignment*** e ***setVerticalAlignment***.



SwingConstants

UFOP

Constante	Descrição
<i>Alinhamento Horizontal</i>	
SwingConstants.LEFT	Posiciona o conteúdo à esquerda.
SwingConstants.CENTER	Posiciona o conteúdo ao centro.
SwingConstants.RIGHT	Posiciona o conteúdo à direita.
<i>Alinhamento Vertical</i>	
SwingConstants.TOP	Posiciona o conteúdo ao topo.
SwingConstants.CENTER	Posiciona o conteúdo ao centro.
SwingConstants.BOTTOM	Posiciona o conteúdo ao fundo.



Criando Labels

- Outra forma de criar labels é utilizar o construtor sem parâmetros e invocar explicitamente os métodos para inserir/retornar conteúdo
 - *setText*: define o texto;
 - *getText*: retorna o texto;
 - *setIcon*: define o ícone;
 - *getIcon*: retorna o ícone
 - *setHorizontalTextPosition*: define a posição horizontal;
 - *setVerticalTextPosition*: define a posição vertical;

Campos de Texto e Introdução à Manipulação de Eventos

Introdução à Manipulação de Eventos



UFOP

- Interfaces gráficas são baseadas em eventos
 - Quando um usuário interage com a aplicação, a interação (conhecida como **evento**) leva a aplicação a realizar uma tarefa;
- Eventos comuns incluem:
 - Clicar em um botão;
 - Escrever em um campo de texto;
 - Selecionar um item de um menu;
 - Fechar uma janela;
 - Mover o mouse.
- O código que realiza uma tarefa em resposta a um evento é chamado de **manipulador de evento**.



Campos de Texto

- Dois componentes que representam campos de texto são os *JTextFields* (pacote *javax.swing*) e os *JPasswordFields* (pacote *javax.swing.text*)
 - A classe *JTextField* herda da classe *JTextComponent*, que fornece as características de componentes baseados em texto da Swing;
 - Por sua vez, a classe *JPasswordField* herda da classe *JTextField* e adiciona diversos métodos específicos para processar senhas.



Campos de Texto

- Quando o usuário digita *enter* em um dos dois componentes introduzidos, um evento ocorre;
- Nos próximos exemplos, será demonstrado como um programa pode realizar uma tarefa em resposta a este evento
 - As técnicas apresentadas são aplicáveis a todos os componentes que geram eventos.



TextFieldFrame.java

UFOP

```
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JPasswordField;
import javax.swing.JOptionPane;

public class TextFieldFrame extends JFrame
{
    private JTextField textField1; // campo de texto com tamanho definido
    private JTextField textField2; // campo de texto construído com texto
    private JTextField textField3; // campo de texto com texto e tamanho
    private JPasswordField passwordField; // campo de senha com texto
```



TextFieldFrame.java

```
//O construtor adiciona os campos de texto ao frame
public TextFieldFrame()
{
    super( "Testing JTextField and JPasswordField" );
    setLayout( new FlowLayout() ); //define o layout do frame
    //constroi um campo de texto com 10 colunas
    textField1 = new JTextField( 10 );
    add( textField1 ); //adiciona o campo de texto ao JFrame
    //constroi um campo de texto com texto padrão
    textField2 = new JTextField( "Enter text here" );
    add( textField2 ); //adiciona o campo de texto ao JFrame
    // constroi um campo de texto com texto padrão e 21 colunas
    textField3 = new JTextField( "Uneditable text field", 21 );
    textField3.setEditable( false ); //desabilita a edição
    add( textField3 ); //adiciona o campo de texto ao JFrame
    //constroi um campo de senha com texto padrão
    passwordField = new JPasswordField( "Hidden text" );
    add( passwordField ); //adiciona o campo de senha ao JFrame
    TextFieldHandler handler = new TextFieldHandler();
    textField1.addActionListener( handler );
    textField2.addActionListener( handler );
    textField3.addActionListener( handler );
    passwordField.addActionListener( handler );
}
```



TextFieldFrame.java

```
//classe interna privada para manipulação de eventos
private class TextFieldHandler implements ActionListener
{
    //processa os eventos dos campos de texto
    public void actionPerformed( ActionEvent event )
    {
        String string = ""; //declara a string a ser exibida

        //o usuário pressionou enter no JTextField textField1
        if( event.getSource() == textField1 )
            string = String.format( "textField1: %s", event.getActionCommand() );
        //o usuário pressionou enter no JTextField textField2
        else if(event.getSource() == textField2)
            string = String.format( "textField2: %s", event.getActionCommand() );
```



TextFieldFrame.java

UFOP

```
//o usuário pressionou enter no JTextField textField3
else if(event.getSource() == textField3)
    string = String.format( "textField3: %s", event.getActionCommand());
    // o usuário pressionou enter no JTextField passwordField
else if(event.getSource() == passwordField)
    string = String.format( "passwordField: %s", new
        String(passwordField.getPassword()) );

//exibe o conteúdo do campo de texto
JOptionPane.showMessageDialog( null, string );
}

}
```



Campos de Texto

- No exemplo, são mostrados 3 construtores para criação de um *JTextField*
 - Somente texto padrão;
 - Somente tamanho padrão;
 - Ambos texto e tamanho padrão.
- Todos os componentes são adicionados ao frame pelo método *add*.



Manipulação de Eventos

- Antes que uma aplicação possa responder a um evento é necessário seguir os seguintes passos:
 - Criar uma classe que represente o manipulador de eventos;
 - Implementar uma interface apropriada, chamada *event-listener interface* (interface *ouvinte* de eventos), na classe do passo anterior;
 - Indicar que os objetos desta mesma classe devem ser notificados quando o evento ocorrer
 - Conhecido como registro do manipulador de eventos.



Manipulação de Eventos

- Java nos permite declarar classes dentro de classes
 - Classes aninhadas, que podem ser *static* ou não;
 - Classes aninhadas não static são chamadas de **classes internas**;
 - Geralmente, utilizam-se classes internas para manipulação de eventos.
- Objetos de classes internas podem acessar todos os métodos e atributos da classe externa
 - Sem a necessidade de uma referência a um objeto da classe externa.
- No nosso exemplo, criamos uma classe interna privada
 - Só será utilizada para criar manipuladores de evento.



Manipulação de Eventos

- Cada tipo de evento é representado por uma classe e só pode ser processado pelo tipo apropriado de manipulador
 - Pressionar o *enter* gera um **ActionEvent**;
 - Este evento é processado por um objeto cuja classe implemente a interface **ActionListener**.
- No nosso exemplo, a classe interna implementa o único método da interface
 - ***actionPerformed***;
 - Especifica a ação a ser tomada.



Manipulação de Eventos

UFOP

- Para registrar um manipulador de eventos em um componente, utilizamos o método ***addActionListener***
 - Recebe como argumento um objeto ***ActionListener***, que pode ser de qualquer classe que implemente esta interface.
- Depois de registrado, o manipulador *ouve* os eventos
 - O manipulador é chamado para processar o evento.
- Se o manipulador não for registrado, o campo de texto é simplesmente ignorado pela aplicação.



Manipulação de Eventos

- O método ***getSource*** da classe *ActionEvent* retorna uma referência ao objeto que é a origem do evento
 - Desta forma, é possível determinar qual componente é a origem do evento;
 - Outra forma mais eficiente é declarar um manipulador por componente
 - Não há a necessidade da estrutura *if-else*.
- Se o usuário interagiu com um *JTextField*, o método ***getActionCommand*** retorna o conteúdo do componente;
- Se o usuário interagiu com um *JPasswordField*, o método ***getPassword*** retorna o conteúdo do componente.



TextFieldTest.java

UFOP

```
import javax.swing.JFrame;

public class TextFieldTest
{
    public static void main( String args[] )
    {
        TextFieldFrame textFieldFrame = new TextFieldFrame();
        textFieldFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        textFieldFrame.setSize( 350, 100 ); //define o tamanho do frame
        textFieldFrame.setVisible( true ); // exibe frame
    }
}
```



Campos de Texto

UFOP

Arquivo TextFieldTest.java

Tipos Comuns de Eventos e *Listener Interfaces*

Tipos Comuns de Eventos e *Listener Interfaces*



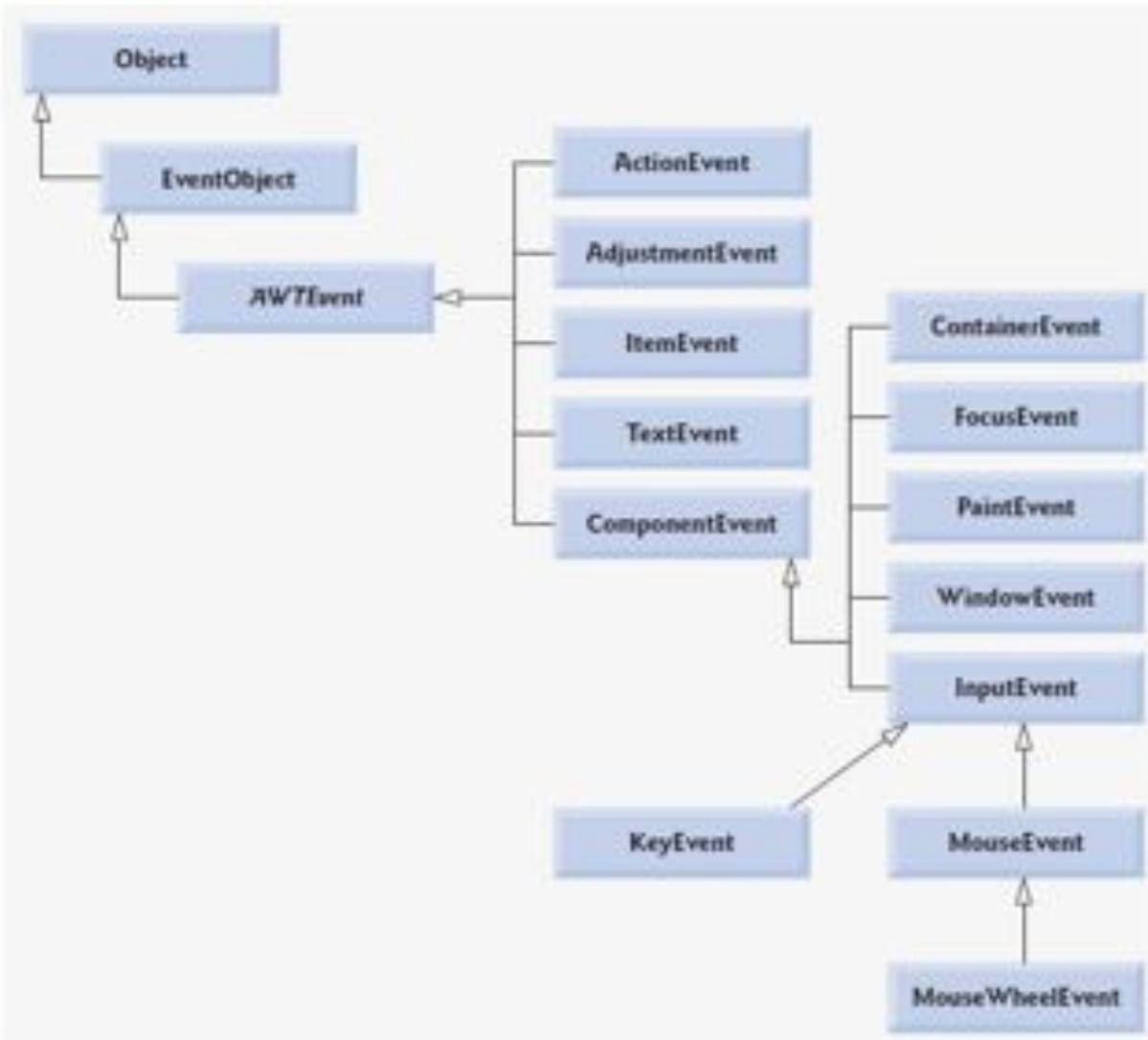
UFOP

- A informação sobre qualquer tipo de evento que ocorra é armazenada em um objeto de uma classe que estenda a classe ***AWTEvent***
 - O diagrama a seguir ilustra algumas das classes de eventos do pacote ***java.awt.event***
 - Alguns eventos específicos dos componentes Swing são declarados no pacote ***javax.swing.event***.



Tipos Comuns de Eventos e *Listener Interfaces*

UFOP



Tipos Comuns de Eventos e *Listener Interfaces*



UFOP

- O mecanismo de manipulação de eventos pode ser summarizado em três partes:
 - A origem do evento
 - Componente com o qual o usuário interage.
 - O objeto do evento
 - Encapsula a informação sobre o evento ocorrido.
 - O ouvinte do evento
 - Objeto notificado pela origem quando ocorre um evento;
 - O ouvinte é notificado através da recepção do objeto do evento;
 - O mesmo objeto é utilizado para responder ao evento.

Tipos Comuns de Eventos e *Listener Interfaces*



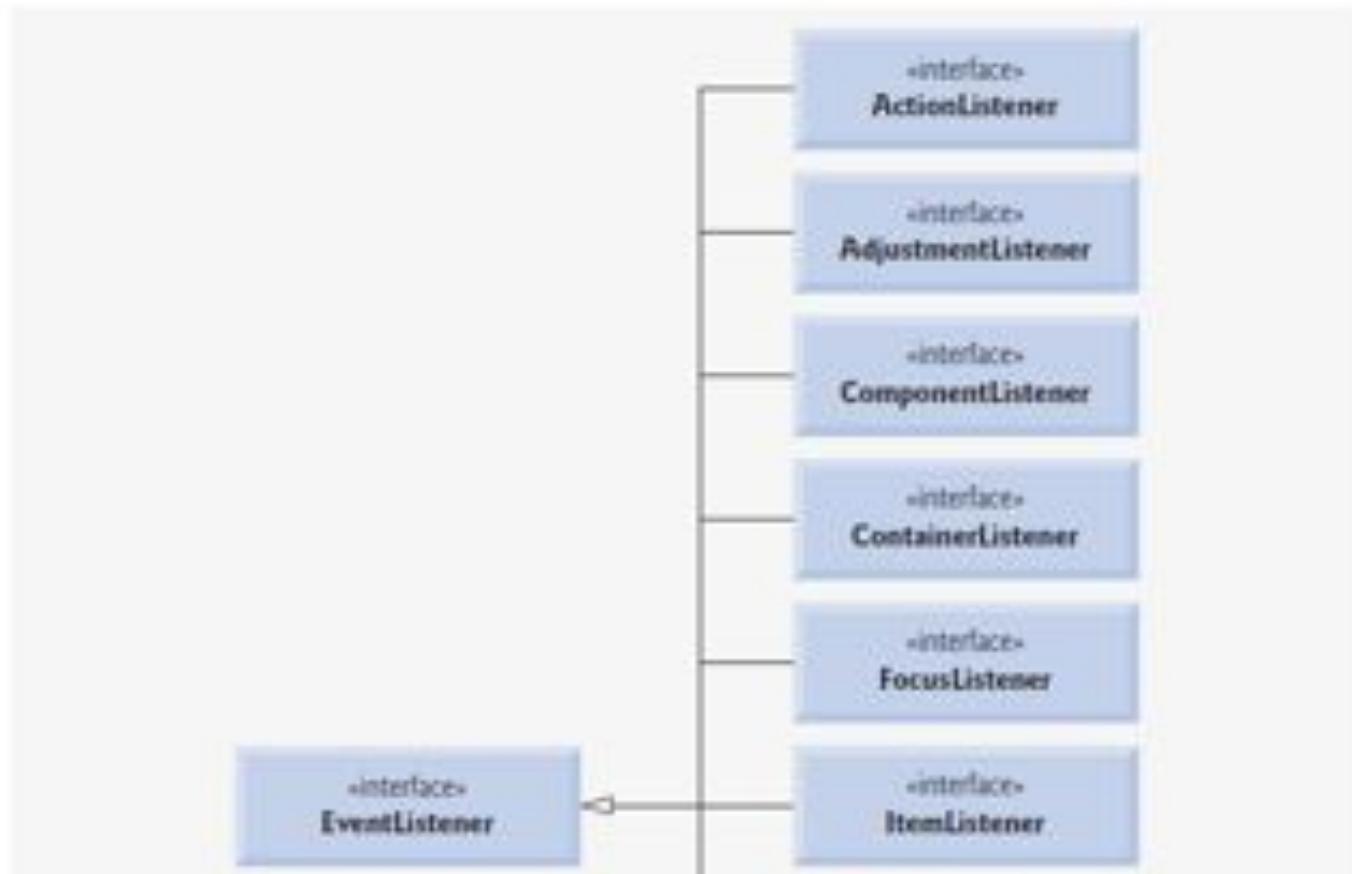
UFOP

- Para cada tipo de objeto de evento, há uma interface de ouvinte correspondente
 - Um ouvinte de eventos é um objeto de uma classe que implementa uma ou mais *listeners interfaces* dos pacotes java.awt.event e javax.swing.event.
- Boa parte dos ouvintes são comuns à AWT e ao Swing
 - Alguns deles exibidos no diagrama a seguir.



Tipos Comuns de Eventos e *Listener Interfaces*

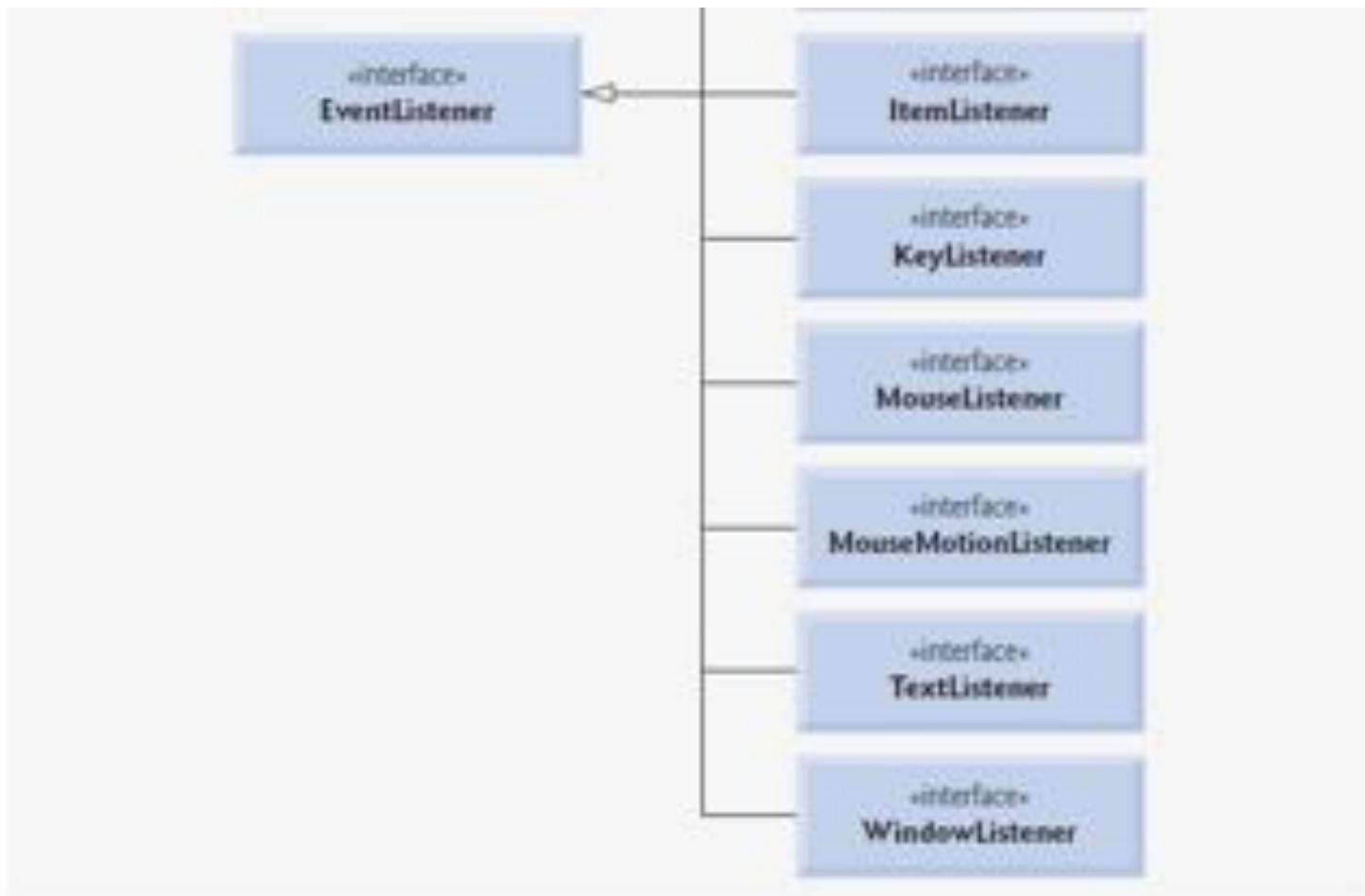
UFOP





Tipos Comuns de Eventos e *Listener Interfaces*

UFOP



Botões



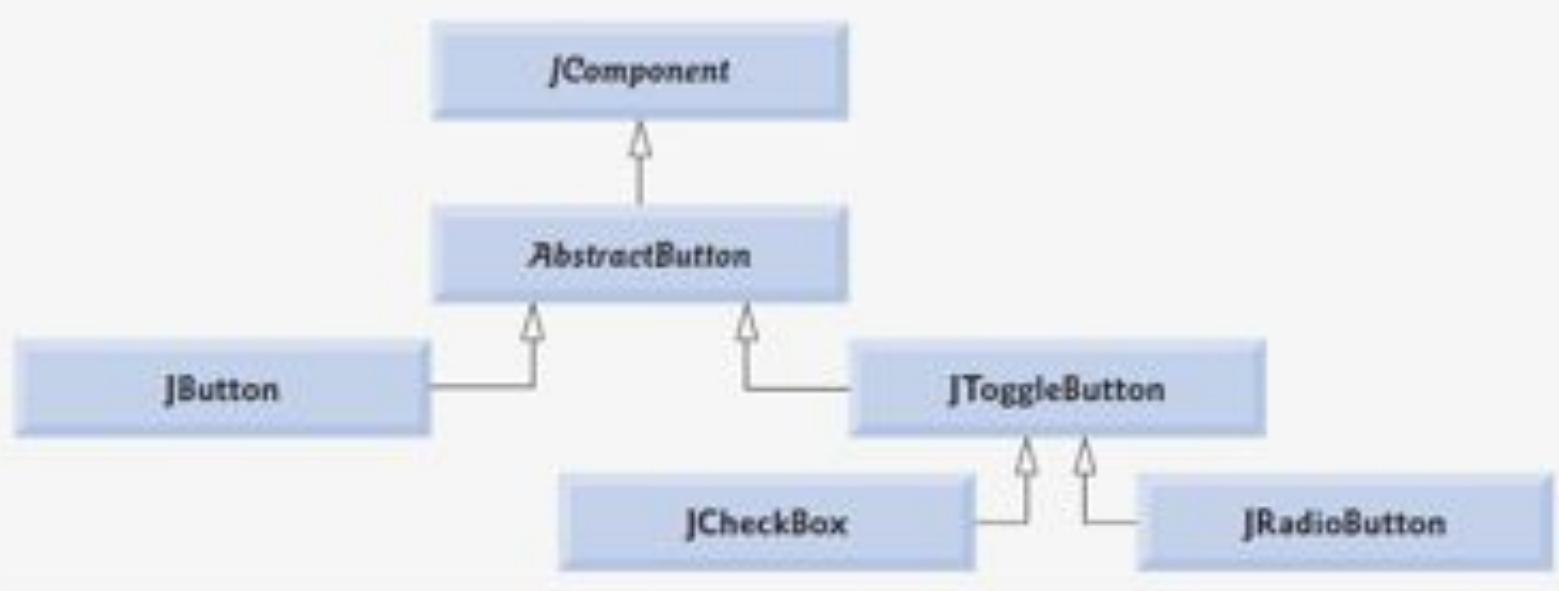
Botões

- Um botão é componente utilizado para disparar uma ação específica;
- Aplicações Java podem ter vários tipos de botões:
 - Botões de comando (*command buttons*);
 - Caixas de marcação (*check boxes*);
 - Botões de alternância (*toggle buttons*);
 - Botões de opção (*radio buttons*).
- Todos os tipos de botões são subclasses de ***AbstractButton***
 - Declara as características comuns dos botões *Swing*.



Botões

UFOP





Botões

UFOP

- Um botão de comando gera um *ActionEvent* quando o usuário clica no botão
 - Criado utilizando a classe ***JButton***;
 - O texto de um botão é chamado de **rótulo do botão**.



ButtonFrame.java

UFOP

```
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

public class ButtonFrame extends JFrame
{
    private JButton plainJButton; // botão só com texto
    private JButton fancyJButton; // botão com ícones
```



ButtonFrame.java

```
// adiciona JButtons ao JFrame
public ButtonFrame()
{
    super( "Testing Buttons" );
    setLayout( new FlowLayout() ); // define o leiaute do frame

    plainJButton = new JButton( "Plain Button" ); // botão com texto
    add( plainJButton ); // adicionar o plainJButton ao JFrame

    Icon bug1 = new ImageIcon( getClass().getResource( "bug1.gif" ) );
    Icon bug2 = new ImageIcon( getClass().getResource( "bug2.gif" ) );
    fancyJButton = new JButton( "Fancy Button", bug1 ); // define a imagem
    fancyJButton.setRolloverIcon( bug2 ); // define a imagem a ser mostrada
                                         //quando o mouse for posicionado sobre o botão
    add( fancyJButton ); // adiciona o fancyJButton ao JFrame

    //cria um novo ButtonHandler para a manipulação de eventos do botão
    ButtonHandler handler = new ButtonHandler();
    fancyJButton.addActionListener( handler );
    plainJButton.addActionListener( handler );
}
```



ButtonFrame.java

UFOP

```
//classe interna para manipulação de eventos
private class ButtonHandler implements ActionListener
{
    //manipula o evento
    public void actionPerformed( ActionEvent event )
    {
        JOptionPane.showMessageDialog( ButtonFrame.this,
            String.format("You pressed: %s", event.getActionCommand()));
    }
}
```



Botões

- O exemplo utiliza um ícone *rollover*
 - Um ícone que é exibido quando o mouse é posicionado sobre o botão;
 - Método *setRolloverIcon*.
- Assim como *JTextFields*, os *JButtons* geram *ActionEvents* que podem ser processados por qualquer objeto *ActionListener*
- A classe *ButtonHandler* implementa o método *actionPerformed*, que manipula os eventos.

A Referência *this* em Classes Internas



UFOP

- Quando utilizado em uma classe interna, o *this* se refere ao objeto desta classe interna que está sendo manipulado
 - Um método da classe interna pode utilizar o *this* dos objetos externos precedendo-o com o nome da classe externa e um ponto;
 - Como em **ButtonFrame.this**.



ButtonTest.java

UFOP

```
import javax.swing.JFrame;

public class ButtonTest
{
    public static void main( String args[] )
    {
        JFrame buttonFrame = new JFrame(); //cria o JFrame
        buttonFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        buttonFrame.setSize( 275, 110 ); // define o tamanho do frame
        buttonFrame.setVisible( true ); // exibe o frame
    }
}
```



Botões

UFOP

Arquivo ButtonTest.java

Botões Que Mantêm Seu Estado



Botões Que Mantêm Seu Estado

UFOP

- Os componentes GUI contêm três tipos de **botões de estado**
 - Caixas de marcação (*check boxes*);
 - Botões de alternância (*toggle buttons*);
 - Botões de opção (*radio buttons*).
- Todos eles possuem valores on/off ou true/false.

Caixas de Marcação



CheckBoxFrame.java

UFOP

```
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JCheckBox;

public class CheckBoxFrame extends JFrame
{
    private JTextField textField; // exibe o texto em fonte formatável
    private JCheckBox boldJCheckBox; // seleciona negrito ou não
    private JCheckBox italicJCheckBox; // seleciona itálico ou não
```



CheckBoxFrame.java

UFOP

```
// construtor que adiciona JCheckboxes ao JFrame
public CheckBoxFrame()
{
    super( "JCheckBox Test" );
    setLayout( new FlowLayout() ); // define o leiaute do frame

    // cria o JTextField e define a fonte
    textField = new JTextField( "Watch the font style change", 20 );
    textField.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
    add( textField ); // adiciona o textField ao JFrame

    boldJCheckBox = new JCheckBox( "Bold" ); // cria o checkbox para o negrito
    italicJCheckBox = new JCheckBox( "Italic" ); // cria o checkbox para o itálico
    add( boldJCheckBox ); // adiciona o checkbox para o negrito ao frame
    add( italicJCheckBox ); // o checkbox para o italicico ao frame

    // registra os ouvintes dos JCheckboxes
    CheckBoxHandler handler = new CheckBoxHandler();
    boldJCheckBox.addItemListener( handler );
    italicJCheckBox.addItemListener( handler );
}
```



CheckBoxFrame.java

UFOP

```
// classe interna privada para a manipulação de eventos
private class CheckBoxHandler implements ItemListener
{
    private int valBold = Font.PLAIN; // controla o estilo negrito
    private int valItalic = Font.PLAIN; // controla o estilo itálico

    // responde aos eventos do checkbox
    public void itemStateChanged( ItemEvent event )
    {
        // processa os eventos do checkbox para o negrito
        if ( event.getSource() == boldJCheckBox )
            valBold =
                boldJCheckBox.isSelected() ? Font.BOLD : Font.PLAIN;

        // processa os eventos do checkbox para o itálico
        if ( event.getSource() == italicJCheckBox )
            valItalic =
                italicJCheckBox.isSelected() ? Font.ITALIC : Font.PLAIN;

        // define a fonte
        textField.setFont( new Font( "Serif", valBold + valItalic, 14 ) );
    }
}
```



Caixas de Marcação

- Um *JTextField* pode utilizar o método ***setFont*** para definir a fonte de seu texto como um objeto da classe ***Font*** (pacote *java.awt*);
- Quando um *JCheckBox* é clicado, um evento ***ItemEvent*** ocorre
 - O método ***itemStateChanged*** deve ser implementado para manipular o evento.
- O método *getSource* pode ser utilizado para determinar qual *checkbox* foi clicado;
- O método ***isSelected*** determina se o *JCheckBox* foi selecionado ou não.



Caixas de Marcação

UFOP

- No exemplo, a classe interna acessa variáveis declaradas na classe externa
 - Classes interna são autorizadas a acessar atributos e métodos das classes externas;
 - Não é necessária nenhuma referência a objetos da classe externa.
- Utilizamos esta propriedade para determinar a origem do evento, o estado dos *checkboxes* e para definir a fonte dos *JTextFields*.



CheckBoxTest.java

UFOP

```
import javax.swing.JFrame;

public class CheckBoxTest
{
    public static void main( String args[] )
    {
        CheckBoxFrame checkBoxFrame = new CheckBoxFrame();
        checkBoxFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        checkBoxFrame.setSize( 275, 100 ); // define o tamanho do frame
        checkBoxFrame.setVisible( true ); // exibe o frame
    }
}
```



Caixas de Marcação

UFOP

Arquivo CheckBoxTest.java

Botões de Opção



Botões de Opção

- **Botões de Opção (*Radio Buttons*)** são similares às caixas de marcação
 - Possuem dois estados: selecionado e não selecionado;
 - Porém, normalmente aparecem em grupos, em que apenas um dos botões pode ser selecionado por vez;
 - Selecionar um botão de opção implica em remover a seleção dos demais;
 - Usamos para representar opções mutuamente exclusivas;
 - O relacionamento entre botões de opção em um grupo é mantido por um objeto ***ButtonGroup***.



RadioButtonFrame.java

UFOP

```
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;

public class RadioButtonFrame extends JFrame
{
    private JTextField textField; // usado para exibir as mudanças na fonte
    private Font plainFont; // fonte para texto simples
    private Font boldFont; // fonte para o texto em negrito
    private Font italicFont; // fonte para o texto em itálico
    private Font boldItalicFont; // fonte para o texto em negrito e itálico
    private JRadioButton plainRadioButton; // seleciona texto simples
    private JRadioButton boldRadioButton; // seleciona texto em negrito
    private JRadioButton italicRadioButton; // seleciona texto em itálico
    private JRadioButton boldItalicRadioButton; // negrito e itálico
    private ButtonGroup radioGroup; // buttongroup para armazenar os radio buttons
```



RadioButtonFrame.java

```
// construtor que adiciona JRadioButtons ao JFrame
public RadioButtonFrame()
{
    super( "RadioButton Test" );
    setLayout( new FlowLayout() ); // define o leiaute do frame

    textField = new JTextField( "Watch the font style change", 25 );
    add( textField ); // adiciona o textField ao JFrame

    // cria os radio buttons
    plainJRadioButton = new JRadioButton( "Plain", true );
    boldJRadioButton = new JRadioButton( "Bold", false );
    italicJRadioButton = new JRadioButton( "Italic", false );
    boldItalicJRadioButton = new JRadioButton( "Bold/Italic", false );

    //adiciona os botões ao frame
    add( plainJRadioButton );
    add( boldJRadioButton );
    add( italicJRadioButton );
    add( boldItalicJRadioButton );

    // cria o relacionamento lógico entre os JRadioButtons
    radioGroup = new ButtonGroup(); // crieia o ButtonGroup
```



RadioButtonFrame.java

```
// adiciona os botões ao grupo
radioGroup.add( plainJRadioButton ); // add plain to group
radioGroup.add( boldJRadioButton ); // add bold to group
radioGroup.add( italicJRadioButton ); // add italic to group
radioGroup.add( boldItalicJRadioButton ); // add bold and italic

// cria os objetos das fontes
plainFont = new Font( "Serif", Font.PLAIN, 14 );
boldFont = new Font( "Serif", Font.BOLD, 14 );
italicFont = new Font( "Serif", Font.ITALIC, 14 );
boldItalicFont = new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
textField.setFont( plainFont ); // set initial font to plain

// registra os eventos para os JRadioButtons
plainJRadioButton.addItemListener( new RadioButtonHandler( plainFont ) );
boldJRadioButton.addItemListener( new RadioButtonHandler( boldFont ) );
italicJRadioButton.addItemListener( new RadioButtonHandler( italicFont ) );
boldItalicJRadioButton.addItemListener( new RadioButtonHandler(
                                boldItalicFont ) );
}

}
```



RadioButtonFrame.java

UFOP

```
// classe interna privada para manipular os eventos dos radio buttons
private class RadioButtonHandler implements ItemListener
{
    private Font font; // fonte associada ao ouvinte

    public RadioButtonHandler( Font f )
    {
        font = f; // define a fonte deste ouvinte
    }

    // manipula os eventos do radio button
    public void itemStateChanged( ItemEvent event )
    {
        textField.setFont( font ); // define a fonte do textField
    }
}
```



Caixas de Opção

- O construtor *JRadioButton* determina o label que acompanha a caixa de opção e também se ela estará selecionada por padrão;
- O objeto da classe **ButtonGroup** é a “cola” que cria a relação lógica entre as diferentes caixas de opção
 - As caixas devem ser adicionadas ao *ButtonGroup*;
 - Duas caixas não podem ser selecionadas simultaneamente.
- Assim como os *JCheckboxes*, o *JRadioButtons* geram ItemEvents quando são clicados
 - Note que cada manipulador de evento é construído utilizando um objeto **Font**.



Caixas de Opção

UFOP

- Os objetos da classe *Font* são inicializados com o tipo da fonte, a formatação (negrito, itálico, limpo) e o tamanho
 - É possível combinar formatações diferentes para uma mesma fonte.
- Quando uma caixa de opção for selecionada, o ***ButtonGroup*** desmarca a caixa selecionada anteriormente e a fonte da caixa de texto é alterada de acordo com o objeto armazenado no manipulador de evento.



RadioButtonTest.java

UFOP

```
import javax.swing.JFrame;

public class RadioButtonTest
{
    public static void main( String args[] )
    {
        RadioButtonFrame radioButtonFrame = new RadioButtonFrame();
        radioButtonFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        radioButtonFrame.setSize( 300, 100 ); // define o tamanho do frame
        radioButtonFrame.setVisible( true ); // exibe o frame
    }
}
```



Caixas de Opção

UFOP

Arquivo RadioButtonTest.java

Caixas de Combinação



Caixas de Combinação

- Uma Caixa de Combinação (*Combo Box* ou *Drop Down List*) fornece uma lista de itens a partir da qual o usuário pode fazer uma seleção
 - Implementada com a classe *JComboBox*;
 - Geram eventos *ItemEvent* assim como as caixas vistas anteriormente.
- No exemplo a seguir, utiliza-se uma caixa de combinação para que o usuário selecione uma imagem a ser exibida.



ComboBoxFrame.java

UFOP

```
import java.awt.FlowLayout;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JComboBox;
import javax.swing.Icon;
import javax.swing.ImageIcon;

public class ComboBoxFrame extends JFrame
{
    private JComboBox imagesJComboBox; // combobox para manter o nome dos ícones
    private JLabel label; // label para exibir o ícone selecionado

    private String names[] = { "bug1.gif", "bug2.gif", "travelbug.gif",
    "buganim.gif" };
    private Icon icons[] = {
        new ImageIcon( getClass().getResource( names[ 0 ] ) ),
        new ImageIcon( getClass().getResource( names[ 1 ] ) ),
        new ImageIcon( getClass().getResource( names[ 2 ] ) ),
        new ImageIcon( getClass().getResource( names[ 3 ] ) );
    };
}
```



ComboBoxFrame.java

```
// construtor que adiciona o JComboBox ao JFrame
public ComboBoxFrame()
{
    super( "Testing JComboBox" );
    setLayout( new FlowLayout() ); // define o layout do frame

    imagesJComboBox = new JComboBox( names ); // cria o JComboBox
    imagesJComboBox.setMaximumRowCount( 3 ); // exibe 3 linhas

    imagesJComboBox.addItemListener(
        new ItemListener() // classe interna anônima
    {
        // manipula os eventos do JComboBox
        public void itemStateChanged( ItemEvent event )
        {
            // determina se a check box foi selecionado
            if ( event.getStateChange() == ItemEvent.SELECTED )
                label.setIcon( icons[ imagesJComboBox.getSelectedIndex() ] );
        }
    });
}
```



Caixas de Combinação

- O objeto ***JComboBox*** é construído utilizando-se o vetor com os endereços das imagens
 - Cada elemento da lista possui um **índice**
 - Primeiro elemento no índice **zero**.
 - O primeiro elemento adicionado aparece como selecionado quando a lista é exibida.
- O método ***setMaximumRowCount*** determina a quantidade máxima de linhas a serem exibidas na lista
 - Se houver mais elementos, uma barra de rolagem é adicionada automaticamente.



Caixas de Combinação

UFOP

- Note que o objeto manipulador de eventos deste exemplo é criado implicitamente pela declaração da própria classe
 - Uma **classe interna anônima**
 - Declarada sem nome, tipicamente aparecendo dentro de um método;
 - Funciona como uma classe interna comum;
 - Como não possui nome, sua declaração já é a instanciação de um objeto.
 - A declaração **ItemListener()** depois de um new começa a declaração de uma classe anônima que implementa a interface **ItemListener**;
 - O uso de parênteses indica a chamada ao construtor padrão da classe anônima.



Caixas de Combinação

UFOP

- O método ***getSelectedIndex*** determina o índice do elemento selecionado na caixa de combinação
 - Quando um elemento é selecionado, o elemento que estava selecionado anteriormente deixa de ser;
 - O que gera dois eventos ***ItemEvent***;
 - Por isso é necessário comparar o retorno do método ***getStateChange*** com a constante **SELECTED**.



ComboBoxTest.java

UFOP

```
import javax.swing.JFrame;

public class ComboBoxTest
{
    public static void main( String args[] )
    {
        ComboBoxFrame comboBoxFrame = new ComboBoxFrame();
        comboBoxFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        comboBoxFrame.setSize( 350, 150 ); // define o tamanho do frame
        comboBoxFrame.setVisible( true ); // exibe o frame
    }
}
```



Caixas de Combinação

UFOP

Arquivo ComboBoxTest.java

Listas



Listas

- Uma lista exibe uma série de itens a partir da qual o usuário pode selecionar um (lista de seleção única) ou mais (lista de seleção múltipla) itens
 - Criadas com a classe *JList*;
- O exemplo a seguir cria uma lista contendo o nome de 13 cores
 - Quando o nome de uma cor é clicada na lista, ocorre um evento *ListSelectionEvent*, causando a mudança da cor de fundo do programa.



ListFrame.java

UFOP

```
import java.awt.FlowLayout;
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.ListSelectionModel;

public class ListFrame extends JFrame
{
    private JList colorJList; // lista para exibir as cores
    private final String colorNames[] = { "Black", "Blue", "Cyan", "Dark Gray",
                                         "Gray", "Green", "Light Gray", "Magenta", "Orange", "Pink",
                                         "Red", "White", "Yellow" };
    private final Color colors[] = { Color.BLACK, Color.BLUE, Color.CYAN,
                                   Color.DARK_GRAY, Color.GRAY, Color.GREEN,
                                   Color.LIGHT_GRAY, Color.MAGENTA, Color.ORANGE, Color.PINK,
                                   Color.RED, Color.WHITE, Color.YELLOW };
}
```



ListFrame.java

```
// construtor que adiciona o JScrollPane que contém o JList ao JFrame
public ListFrame()
{
    super( "List Test" );
    setLayout( new FlowLayout() ); // define o leiaute do frame

    colorJList = new JList( colorNames ); // cria a lista com o vetor colorNames
    colorJList.setVisibleRowCount( 5 ); // exibe 5 linhas por vez

    // não permite múltiplas seleções
    colorJList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );

    //adiciona o JScrollPane que contém o JList ao JFrame
    add( new JScrollPane( colorJList ) );

    colorJList.addListSelectionListener(
        new ListSelectionListener() // classe interna anônima
    {
        // manipula os eventos de seleção na lista
        public void valueChanged( ListSelectionEvent event )
        {
            getContentPane().setBackground(colors[ colorJList.getSelectedIndex() ] );
        }
    });
}
```



Listas

UFOP

- A lista é criada a partir do vetor de *strings* constantes com o nome das cores
 - O método ***setVisibleRowCount*** determina a quantidade de itens visíveis na lista;
 - O método ***setSelectionMode*** especifica se a lista possui múltiplas seleções, seleção única ou seleção de intervalo (que também pode ser único ou múltiplo);
 - Ao contrário da caixa de combinação, uma lista não fornece uma barra de rolagem se houver mais itens que o tamanho determinado da lista
 - Um objeto ***JScrollPane*** deve ser utilizado nestes casos.
 - Para registrar um ouvinte, é utilizado o método ***addListSelectionListener***.



Listas

UFOP

- Quando ocorre uma mudança de seleção na lista, o método ***valueChanged*** é executado e altera a cor de fundo da janela
 - Os métodos ***getContentPane*** e ***setBackground*** são utilizados para isto;
 - Listas também possuem o método ***getSelectedIndex*** para determinar qual item foi selecionado
 - A indexação começa do zero.



ListTest.java

UFOP

```
import javax.swing.JFrame;

public class ListTest
{
    public static void main( String args[] )
    {
        ListFrame listFrame = new ListFrame(); // create ListFrame
        listFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        listFrame.setSize( 350, 150 ); // define o tamanho do frame
        listFrame.setVisible( true ); // exibe o frame
    }
}
```



Listas

UFOP

Arquivo ListTest.java

Manipulação de Eventos do Mouse



Manipulação de Eventos do Mouse

UFOP

- Para manipular os eventos gerados pelo mouse utilizamos as interfaces ***MouseListener*** e ***MouseMotionListener***;
 - Os métodos destas interfaces são invocados quando o mouse interage com algum dos componentes se o objeto ouvinte apropriado estiver registrado no componente;
 - De fato, os eventos de mouse podem ser capturados por qualquer componente que derive de *java.awt.component*.
- A tabela a seguir sumariza os métodos destas interfaces.

Métodos da Interface

MouseListener



UFOP

Método	Descrição
<i>mousePressed</i>	Invocado quando um botão do mouse é pressionado e o cursor está sobre um componente.
<i>mouseClicked</i>	Invocado quando um botão do mouse é pressionado e solto e o cursor está parado sobre um componente. Este evento é sempre precedido do evento <i>mousePressed</i> .
<i>mouseReleased</i>	Invocado quando um botão do mouse é solto e o cursor está sobre um componente. Este evento é sempre precedido de um ou mais eventos <i>mouseDragged</i> .
<i>mouseEntered</i>	Invocado quando o mouse adentra os limites de um componente.
<i>mouseExited</i>	Invocado quando o mouse deixa os limites de um componente.

Métodos da Interface *MouseMotionListener*



UFOP

Método	Descrição
<i>mouseDragged</i>	Invocado quando um botão do mouse é pressionado enquanto o cursor está sobre um componente e o mouse é movido com o botão pressionado. Este evento é sempre precedido por um evento <i>mousePressed</i> . Este evento é enviado para o componente em que a ação começou.
<i>mouseMoved</i>	Invocado quando o mouse é movido quando o cursor está parado sobre um componente. Este evento é enviado para o componente em que a ação começou.



Manipulação de Eventos do Mouse

UFOP

- Cada um dos métodos recebe como argumento um objeto ***MouseEvent***
 - Contém informações sobre o evento, tais como as coordenadas *x* e *y* do local do evento
 - Medidos a partir do canto superior esquerdo do componente em que ocorreu o evento.
- Java ainda fornece a interface ***MouseWheelListener*** que captura o evento de rolagem do botão central do mouse
 - Método ***mouseWheelMoved***.
- O exemplo a seguir implementa os sete métodos das duas primeiras interfaces
 - Cada evento do mouse é exibido na **barra de status** da janela.



MouseTrackerFrame.java

UFOP

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
```



MouseTrackerFrame.java

UFOP

```
public class MouseTrackerFrame extends JFrame
{
    private JPanel mousePanel; // painel em que os eventos do mouse ocorrerão
    private JLabel statusBar; // label que exibe a informação sobre o evento

    // construtor que cria a GUI e registra os
    // manipuladores de eventos do mouse
    public MouseTrackerFrame()
    {
        super( "Demonstrating Mouse Events" );

        mousePanel = new JPanel(); // cria o painel
        mousePanel.setBackground( Color.WHITE ); // define a cor de fundo
        add( mousePanel, BorderLayout.CENTER ); // adiciona o painel ao JFrame

        statusBar = new JLabel( "Mouse outside JPanel" );
        add( statusBar, BorderLayout.SOUTH ); // adiciona o label ao JFrame

        // cria e registra os ouvintes para eventos do mouse
        MouseHandler handler = new MouseHandler();
        mousePanel.addMouseListener( handler );
        mousePanel.addMouseMotionListener( handler );
    }
}
```



MouseTrackerFrame.java

```
private class MouseHandler implements MouseListener, MouseMotionListener
{
    // manipula o evento de quando o mouse é solto depois do clique
    public void mouseClicked( MouseEvent event )
    {
        statusBar.setText( String.format( "Clicked at [%d, %d]", event.getX(), event.getY() ) );
    }

    // manipula o evento de quando o mouse é pressionado
    public void mousePressed( MouseEvent event )
    {
        statusBar.setText( String.format( "Pressed at [%d, %d]", event.getX(),
event.getY() ) );
    }

    // manipula o evento de quando o mouse é solto depois de ter sido arrastado
    public void mouseReleased( MouseEvent event )
    {
        statusBar.setText( String.format( "Released at [%d, %d]", event.getX(),
event.getY() ) );
    }

    // manipula o evento de quando o mouse entra em uma área
    public void mouseEntered( MouseEvent event )
    {
        statusBar.setText( String.format( "Mouse entered at [%d, %d]", event.getX(),
event.getY() ) );
        mousePanel.setBackground( Color.GREEN );
    }
}
```



MouseTrackerFrame.java

UFOP

```
// manipula o evento de quando o mouse deixa a área
public void mouseExited( MouseEvent event )
{
    statusBar.setText( "Mouse outside JPanel" );
    mousePanel.setBackground( Color.WHITE );
}

// manipuladores de evento MouseMotionListener
// manipula o evento quando o mouse é movimento com o botão pressionado
public void mouseDragged( MouseEvent event )
{
    statusBar.setText( String.format( "Dragged at [%d, %d]", event.getX(),
                                    event.getY() ) );
}

// manipula o evento de quando o usuário move o mouse
public void mouseMoved( MouseEvent event )
{
    statusBar.setText( String.format( "Moved at [%d, %d]", event.getX(),
                                    event.getY() ) );
}
}
```



MouseTrackerFrame.java

- No início do exemplo, é criado um *JPanel*, que cobrirá a janela do programa e rastreará todos os eventos do mouse
 - Quando o mouse entra na área do *JPanel* ele se torna verde, e quando o mouse deixa a área, ele se torna branco;
- O gerenciador de layout utilizado é o *BorderLayout*
 - Arranja os componentes em cinco regiões
 - NORTH, SOUTH, EAST, WEST e CENTER.
 - O gerenciador automaticamente dimensiona o componente para ocupar todo o espaço disponível a partir da região especificada.



MouseTrackerFrame.java

- Os métodos *addMouseListener* e *addMouseMotionListener* são utilizados para registrar os respectivos ouvintes
 - No exemplo, um *MouseHandler* é tanto um *MouseListener* quanto um *mouseMotionListener*, pois a classe implementa as duas interfaces
 - Também seria possível implementar a interface *MouseInputListener*, que combina as duas interfaces anteriores.
- Quando qualquer um dos eventos ocorre, os manipuladores de evento utilizam os métodos *getX* e *getY* para determinar as coordenadas x e y do mouse quando o evento ocorreu.



Manipulação de Eventos do Mouse

UFOP

```
import javax.swing.JFrame;

public class MouseTracker
{
    public static void main( String args[] )
    {
        MouseTrackerFrame mouseTrackerFrame = new MouseTrackerFrame();
        mouseTrackerFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        mouseTrackerFrame.setSize(700, 400); // define o tamanho do frame
        mouseTrackerFrame.setVisible( true ); // exibe o frame
    }
}
```



Manipulação de Eventos do Mouse

UFOP

Arquivo MouseTracker.java

Classes Adaptadoras



Classes Adaptadoras

- Muitas interfaces *listeners* de eventos contêm múltiplos métodos
 - Nem sempre é desejável declarar todos os métodos de uma interface;
 - Por exemplo, uma aplicação pode precisar apenas do método *mouseClicked* da interface *MouseListener*, mas terá que declarar os outros quatro métodos.
- Para as interfaces *listeners* que possuem múltiplos métodos, os pacotes `java.awt.event` e `java.swing.event` fornecem **classes adaptadoras** de *listeners* de eventos.



Classes Adaptadoras

UFOP

- Uma **classe adaptadora** implementa uma interface e fornece uma implementação padrão para cada método da interface
 - Corpo vazio.
- É possível estender uma classe adaptadora para herdar a implementação padrão de todos os métodos
 - E sobrescrever apenas os métodos de interesse;
 - Se por acaso errarmos escrevermos o nome do método a ser sobrescrito, criaremos um método diferente, e não haverá um ouvinte de fato.



Classes Adaptadoras

UFOP

Classe adaptadora da AWT	Implementa a interface
<i>ComponentAdapter</i>	<i>ComponentListener</i>
<i>ContainerAdapter</i>	<i>ContainerListener</i>
<i>FocusAdapter</i>	<i>FocusListener</i>
<i>KeyAdapter</i>	<i>KeyListener</i>
<i>MouseAdapter</i>	<i>MouseListener</i>
<i>MouseMotionAdapter</i>	<i>MouseMotionListener</i>
<i>WindowAdapter</i>	<i>WindowListener</i>



Classes Adaptadoras

UFOP

- O exemplo a seguir demonstra como determinar o número de cliques do mouse e como diferenciar os botões do mouse
 - O *listener* de evento é o objeto de uma classe que estende a classe adaptadora *MouseAdapter*;
 - Desta forma, implementamos apenas o método *mouseClicked*.



MouseDetailsFrame.java

UFOP

```
import java.awt.BorderLayout;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class MouseDetailsFrame extends JFrame
{
    private String details; // representação em String
    private JLabel statusBar; // JLabel que aparece no fundo da janela

    // construtor que define a String da barra de título e registra o ouvinte do mouse
    public MouseDetailsFrame()
    {
        super( "Mouse clicks and buttons" );

        statusBar = new JLabel( "Click the mouse" );
        add( statusBar, BorderLayout.SOUTH );
        addMouseListener( new MouseClickHandler() ); // adiciona o manipulador
    }
}
```



MouseDetailsFrame.java

UFOP

```
//classe interna para manipular eventos
private class MouseClickHandler extends MouseAdapter
{
    // manipula o evento de clique do mouse e determina qual botão foi pressionado
    public void mouseClicked( MouseEvent event )
    {
        int xPos = event.getX(); // pega a posição x do mouse
        int yPos = event.getY(); // pega a posicao y do mouse

        details = String.format( "Clicked %d time(s)", event.getClickCount() );

        if ( event.isMetaDown() ) // botão direito do mouse
            details += " with right mouse button";
        else if ( event.isAltDown() ) // botão do meio do mouse
            details += " with center mouse button";
        else // botão esquerdo do mouse
            details += " with left mouse button";

        statusBar.setText( details ); //exibe a mensagem em statusBar
    }
}
```



Classes Adaptadoras

UFOP

- Novamente são utilizados os métodos ***getX*** e ***getY*** para determinar as coordenadas x e y do mouse quando o evento de clique ocorreu;
- O método ***getClickCount*** retorna a quantidade de cliques do mouse
 - Os métodos ***isMetaDown*** e ***isAltDown*** são utilizados para determinar qual botão do mouse foi clicado
 - Java assume que o botão do meio é clicado quando o ***alt*** é pressionado e o botão esquerdo do mouse é clicado;
 - Também assume que o botão da direita é clicado quando a tecla ***Meta*** é pressionada e o botão esquerdo do mouse é clicado.



MouseDetails.java

UFOP

```
import javax.swing.JFrame;

public class MouseDetails
{
    public static void main( String args[] )
    {
        MouseDetailsFrame mouseDetailsFrame = new MouseDetailsFrame();
        mouseDetailsFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        mouseDetailsFrame.setSize( 400, 150 ); // define o tamanho do frame
        mouseDetailsFrame.setVisible( true ); // exibe o frame
    }
}
```



Classes Adaptadoras

UFOP

Arquivo MouseDetails.java

Subclasse JPanel para Desenhar com o Mouse



Subclasse JPanel para Desenhar com o Mouse

UFOP

- Todos os componentes leves *Swing* possuem o método ***paintComponent***
 - Sobrescrevendo este método, é possível especificar como desenhar formas utilizando as facilidades gráficas do Java.
- Por exemplo, podemos utilizar um ***JPanel*** como área de desenho
 - O evento ***MouseDragged*** é utilizado
 - Por isso é possível estender apenas a classe adaptadora *MouseMotionAdapter*.
 - A classe ***Point*** também pode ser utilizada para armazenar as coordenadas de cada evento de arrasto do mouse;
 - A classe ***Graphics*** pode ser utilizada para desenhar.

Subclasse JPanel para Desenhar com o Mouse



UFOP

- O código a seguir implementa este exemplo
 - Utilizando um vetor de 10.000 objetos *Point* que determinam a trajetória do mouse
 - Através do tratamento do evento de arrasto do mouse.
 - O método *paintComponent* utiliza estes pontos para o desenho.



PaintPanel.java

UFOP

```
import java.awt.Point;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import javax.swing.JPanel;

public class PaintPanel extends JPanel
{
    private int pointCount = 0; // conta o número de pontos

    // vetor de 10000 referências java.awt.Point
    private Point points[] = new Point[ 10000 ];

    // criar a GUI e registra o manipulador de eventos do mouse
    public PaintPanel()
    {
        // manipula o evento de movimentação do mouse
        addMouseMotionListener(
```



PaintPanel.java

UFOP

```
new MouseMotionAdapter() // classe interna anonima
{
    // armazena as coordenadas de seleção e pinta novamente
    public void mouseDragged( MouseEvent event )
    {
        if ( pointCount < points.length )
        {
            points[ pointCount ] = event.getPoint(); // armazena o ponto
            pointCount++; // incrementa o número de pontos no vetor
            repaint(); // pinta novamente o JFrame
        }
    }
};
```



PaintPanel.java

UFOP

```
// exibe o desenho em uma caixa 4x4 no local especificado na janela
public void paintComponent( Graphics g )
{
    super.paintComponent( g ); // limpa a área de desenho

    // exibe todos os pontos no vetor
    for ( int i = 0; i < pointCount; i++ )
        g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
}
```

Subclasse JPanel para Desenhar com o Mouse



UFOP

- O método ***repaint*** (herdado diretamente da classe *Component*) indica que o componente deve ser recarregado na tela tão rápido quanto possível
 - Se o componente for **opaco**, seu fundo será “limpado”;
 - Se o componente for **transparente**, o fundo é mantido;
 - O método ***setOpaque*** recebe um argumento booleano para indicar opaco (*true*) ou transparente (*false*).
- O método ***paintComponent*** por sua vez recebe um objeto *Graphics*
 - É chamado automaticamente toda vez que o componente precisar ser exibido na tela.

Subclasse JPanel para Desenhar com o Mouse



UFOP

- O método *fillOval* (classe *Graphics*) desenha uma forma oval sólida
 - Os quatro parâmetros definem a área retangular (também conhecida como *bounding box*) na qual a forma será exibida;
 - Os primeiros dois parâmetros são as coordenadas x e y do canto superior esquerdo;
 - Os últimos dois parâmetros são a largura e a altura da área retangular.



Painter.java

UFOP

```
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class Painter
{
    public static void main( String args[] )
    {
        // create JFrame
        JFrame application = new JFrame( "A simple paint program" );

        PaintPanel paintPanel = new PaintPanel(); // cria o painel
        application.add( paintPanel, BorderLayout.CENTER ); // no centro

        // cria o label e o posiciona ao sul do BorderLayout
        application.add( new JLabel( "Drag the mouse to draw" ),BorderLayout.SOUTH );

        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        application.setSize( 400, 200 ); // define o tamanho do frame
        application.setVisible( true ); // exibe o frame
    }
}
```

Subclasse JPanel para Desenhar com o Mouse



UFOP

Arquivo Painter.java

Manipulação de Eventos do Teclado

Manipulação de Eventos do Teclado



UFOP

- Eventos do teclado são gerados quando as teclas são pressionadas ou soltas
 - A interface para manipulação destes eventos é a ***KeyListener***;
 - Uma classe que implemente esta interface deve fornecer declarações para os métodos:
 - ***keyPressed***: invocado pelo pressionamento de qualquer tecla;
 - ***keyTyped***: invocado pelo pressionamento de qualquer tecla que não seja uma tecla de ação;
 - ***keyReleased***: invocado quando uma tecla é liberada depois de um dos eventos acima.



Manipulação de Eventos do Teclado

UFOP



- Typewriter keys
- Function keys
- Enter keys
- System keys
- Application key
- Cursor control keys
- Other

Teclas de ação: *Home, End, Page Up, Page Down, setas, F+, Num Lock, Print Screen, Scroll Lock, Caps Lock e Pause.*



KeyDemoFrame.java

UFOP

```
import java.awt.Color;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
import javax.swing.JFrame;
import javax.swing.JTextArea;

public class KeyDemoFrame extends JFrame implements KeyListener
{
    private String line1 = ""; // primeira linha do textarea
    private String line2 = ""; // segunda linha do textarea
    private String line3 = ""; // terceira linha do textarea
    private JTextArea textArea; // textarea para exibir a saída
```



KeyDemoFrame.java

```
public KeyDemoFrame()
{
    super( "Demonstrating Keystroke Events" );

    textArea = new JTextArea( 10, 15 ); // cria o JTextArea
    textArea.setText( "Press any key on the keyboard..." );
    textArea.setEnabled( false ); // desabilita o textarea
    textArea.setDisabledTextColor( Color.BLACK ); // define a cor do texto
    add( textArea ); // adiciona o textarea ao JFrame

    addKeyListener( this ); // permite que o frame processe os eventos das teclas
}

// manipula uma tecla pressionada
public void keyPressed( KeyEvent event )
{
    line1 = String.format( "Key pressed: %s" , event.getKeyText(
                                event.getKeyCode() ) ); // exibe a tecla pressionada
    setLines2and3( event ); // define as linhas 2 e 3
}
```



KeyDemoFrame.java

UFOP

```
// manipula quando uma tecla é solta
public void keyReleased( KeyEvent event )
{
    line1 = String.format( "Key released: %s", event.getKeyText(
        event.getKeyCode() ) ); // exibe a tecla solta
    setLines2and3( event ); // define as linhas 2 e 3
}

// manipula o pressionamento de uma tecla de ação
public void keyTyped( KeyEvent event )
{
    line1 = String.format( "Key typed: %s", event.getKeyChar() );
    setLines2and3( event ); // define as linhas 2 e 3
}
```



KeyDemoFrame.java

UFOP

```
// define as linhas 2 e 3 da saída
private void setLines2and3( KeyEvent event )
{
    line2 = String.format( "This key is %s an action key", ( event.isActionKey() ?
                                                                "" : "not " ) );

    String temp = event.getKeyModifiersText( event.getModifiers() );

    line3 = String.format( "Modifier keys pressed: %s", ( temp.equals( "" ) ?
                                                                "none" : temp ) );

    textArea.setText( String.format( "%s\n%s\n%s\n", line1, line2, line3 ) );
    // exibe três linhas de texto
}
```

Manipulação de Eventos do Teclado



UFOP

- O método ***getKeyCode*** retorna o **código virtual** da tecla pressionada
 - A classe *KeyEvent* mantém um conjunto de constantes – as constantes de código virtual – que representam todas as teclas do teclado;
 - Para determinar a tecla pressionada, basta comparar o resultado do método *getKeyCode*;
- O método ***getKeyText*** retorna uma *string* contendo o nome da tecla pressionada;
- O valor Unicode da tecla pressionada pode ser obtido pelo método ***getKeyChar***;
- O método ***isActionKey*** determina se a tecla é de ação ou não.

Manipulação de Eventos do Teclado



UFOP

- O método ***getModifiers*** determina se alguma tecla de modificação (como *shift*, *alt* e *ctrl*) foi pressionada quando o evento ocorreu
 - Para obter uma *string* que contenha o nome das teclas utiliza-se o método ***getKeyModifiersText***.
- Para testar uma tecla especificamente, podemos também utilizar os métodos
 - ***isAltDown***;
 - ***isControlDown***;
 - ***isMetaDown***;
 - ***isShiftDown***.



KeyDemo.java

UFOP

```
import javax.swing.JFrame;

public class KeyDemo
{
    public static void main( String args[] )
    {
        KeyDemoFrame keyDemoFrame = new KeyDemoFrame();
        keyDemoFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        keyDemoFrame.setSize( 350, 100 ); // define o tamanho do frame
        keyDemoFrame.setVisible( true );// exibe o frame
    }
}
```

Manipulação de Eventos do Teclado



UFOP

Arquivo KeyDemo.java

Áreas de Texto



Áreas de Texto

UFOP

- Uma área de texto fornece uma área para manipular múltiplas linhas de texto
 - Implementada pela classe *JTextArea*;
 - Possui métodos em comum com os *JTextFields*.
- O exemplo a seguir possui duas áreas de texto
 - A primeira exibe um texto que o usuário pode selecionar;
 - A segunda é não editável e é utilizada para exibir o texto selecionado na primeira área.
- *JTextAreas* não possuem eventos de ação
 - Um botão será utilizado para disparar o evento de cópia do texto.



TextAreaFrame.java

UFOP

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.Box;
import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.JButton;
import javax.swing.JScrollPane;

public class TextAreaFrame extends JFrame
{
    private JTextArea textArea1; // exibe a string de demonstração
    private JTextArea textArea2; // o texto selecionado é copiado para cá
    private JButton copyJButton; // inicia a cópia do texto
```



TextAreaFrame.java

UFOP

```
public TextAreaFrame()
{
    super( "TextArea Demo" );
    Box box = Box.createHorizontalBox(); // cria o box
    String demo = "This is a demo string to\n" + "illustrate copying text\nfrom
        one textarea to \n" + "another textarea using an\ensexternal event\n";
    textArea1 = new JTextArea( demo, 10, 15 ); // cria o textarea1
    box.add( new JScrollPane( textArea1 ) ); // adiciona o scrollpane

    copyJButton = new JButton( "Copy >>>" ); // cria o botão de cópia
    box.add( copyJButton ); // adiciona o botão de cópia para o box
    copyJButton.addActionListener(
        new ActionListener() // classe interna anônima
    {
        // define o texto em textArea2 como o texto selecionado em textArea1
        public void actionPerformed( ActionEvent event )
        {
            textArea2.setText( textArea1.getSelectedText() );
        }
    });
};
```



TextAreaFrame.java

UFOP

```
textArea2 = new JTextArea( 10, 15 ); // cria o segundo textarea  
textArea2.setEditable( false ); // desabilita a edição  
box.add( new JScrollPane( textArea2 ) ); // adiciona o scrollpane  
  
add( box ); // adiciona o box ao frame  
}  
}
```



Áreas de Texto

UFOP

- **Box** é um contêiner utilizado para organizar os componentes de uma GUI
 - Utiliza o gerenciador de layout *BoxLayout* para organizar os componentes vertical ou horizontalmente;
 - O método ***createHorizontalBox*** organiza os componentes da esquerda para a direita, na ordem em que são adicionados.
- O construtor dos *JTextAreas* recebe três argumentos
 - O texto inicial, o número de linhas e o número de colunas.
- Não são adicionadas barras de rolagem automaticamente aos *JtextAreas*
 - É necessário criar um objeto *JScrollPane* e adicioná-lo.



Áreas de Texto

UFOP

- Quando o usuário clica no botão de cópia, o método ***getSelectedText*** retorna o texto selecionado na primeira área de texto
 - O método ***setText*** altera o texto da segunda área de texto.
- Para definir se uma área de texto é editável ou não, utilizamos o método ***setEditable***;
- Para evitar a “quebra” de palavras no final da linha o método ***setLineWrap*** pode ser utilizado.



Políticas para Barras de Rolagem

UFOP

- Métodos *setVerticalScrollbarPolicy* e *setHorizontalScrollbarPolicy*
- **Sempre**
 - JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
 - JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS
- **Se necessário (padrão)**
 - JScrollPane.VERTICAL_SCROLLBAR_NEEDED
 - JScrollPane.HORIZONTAL_SCROLLBAR_NEEDED
- **Nunca**
 - JScrollPane.VERTICAL_SCROLLBAR_NEVER
 - JScrollPane.HORIZONTAL_SCROLLBAR_NEVER



TextAreaDemo.java

UFOP

```
import javax.swing.JFrame;

public class TextAreaDemo
{
    public static void main( String args[] )
    {
        TextAreaFrame textAreaFrame = new TextAreaFrame();
        textAreaFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        textAreaFrame.setSize( 425, 200 ); // define o tamanho do frame
        textAreaFrame.setVisible( true ); // exibe o frame
    }
}
```



Áreas de Texto

UFOP

Arquivo TextAreaDemo.java



Na próxima aula

- Polimorfismo