Java – Aula 02

GUI - Interfaces gráficas

15/08/2012

Celso Olivete Júnior

olivete@fct.unesp.br

Na aula passada

- Introdução
  - Entrada e saída de dados
    - System.out...
    - Scanner
    - JOptionPane
    - Estruturas de controle
    - Arrays
    - Strings
    - Classes
    - ...

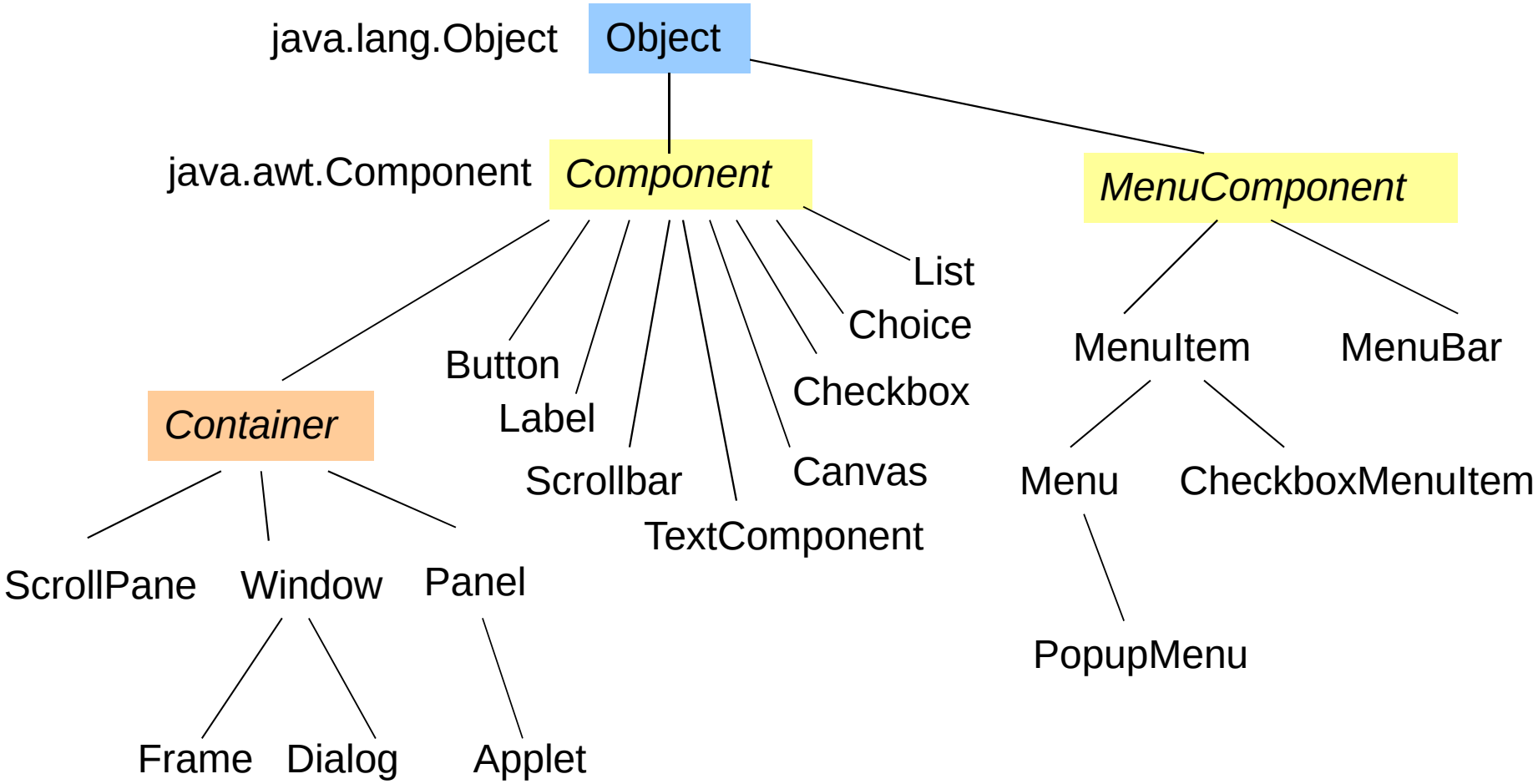
Na aula de hoje

- Interfaces gráficas
  - Hierarquias de classes para GUI
  - Usando componentes de interface
    - AWT = Abstract Window Toolkit
- Programação de aplicações interativas
  - Modelo de delegação de eventos
  - Tipos de eventos e listeners associados

AWT - Abstract Window Toolkit

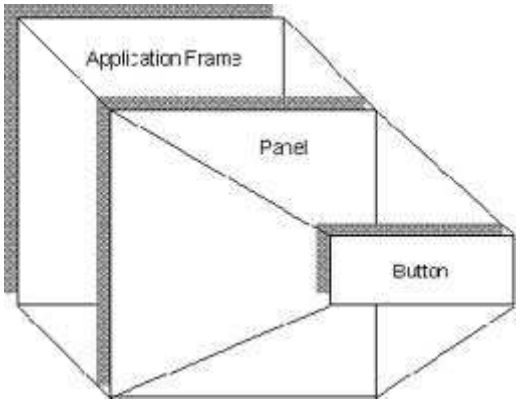
- java.awt.*
- Uma interface gráfica é um conjunto de *Components* inseridos num *Container*
  - Componentes possibilitam interação
  - A maioria gera eventos
- AWT leva a uma interface com usuário independente de plataformas

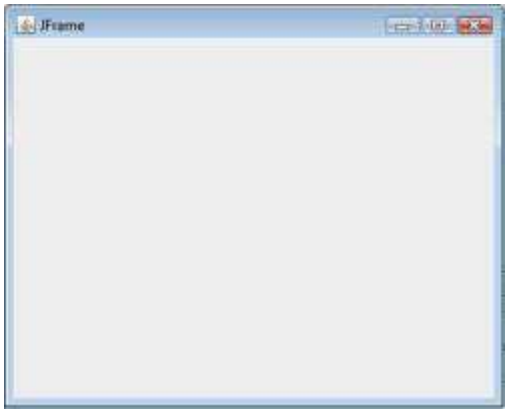
Hierarquia de Classes



Components e *containers*

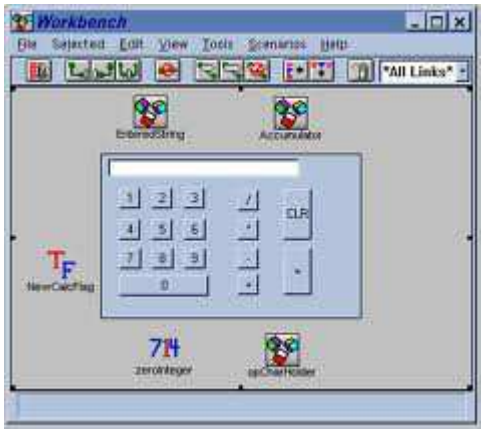
- *Containers* são componentes especiais utilizados para agrupar unidades de interface.





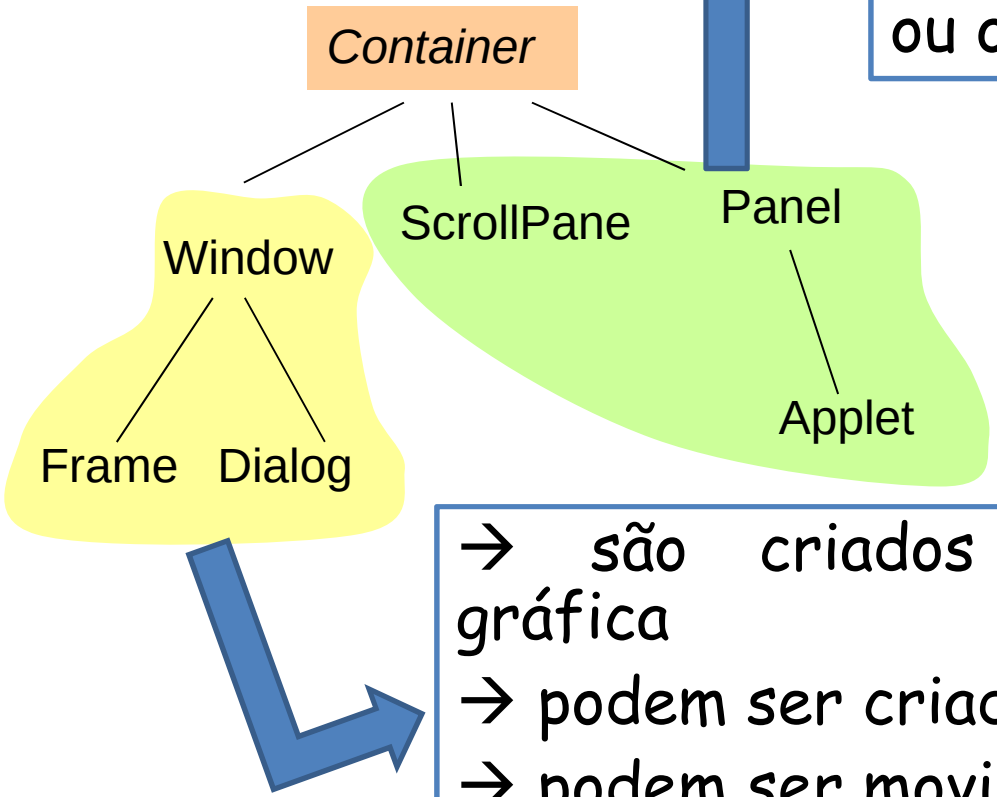
Components e *containers*

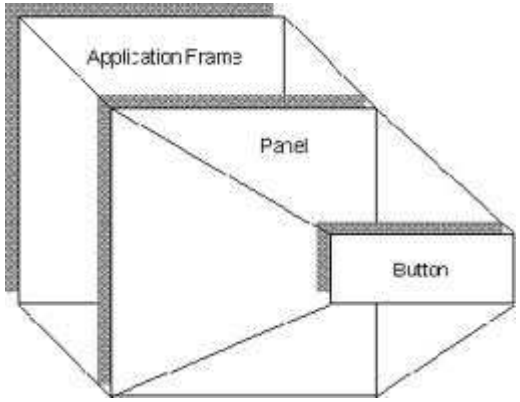
- *Components* são elementos de interface, tanto de agrupamento como de interação botões, labels, áreas de entrada de texto



Containers

- devem ser criados ligados a outra superfície gráfica
- não podem ser movidos (move-se a janela do browser ou o frame)





→ são criados como superfície gráfica
→ podem ser criados sozinhos
→ podem ser movimentados

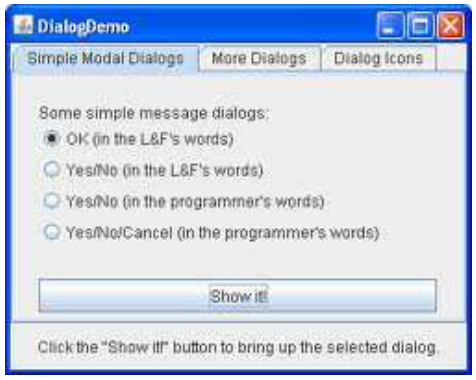
Containers

- *Container*
  - classe pai de todas que podem conter componentes. Não é instanciada.
- *Panel*
  - *container* usado para organizar e agrupar componentes. Pode ser incluído em outros
- *Applet*
  - panel destinado à exibição num browser

Containers

- *Window*
  - container reposicionável pelo usuário. Abstrata, não tem bordas, nem título, nem menu
- *Frame*
  - window com bordas e título. Pode ter menus, ser iconizado.

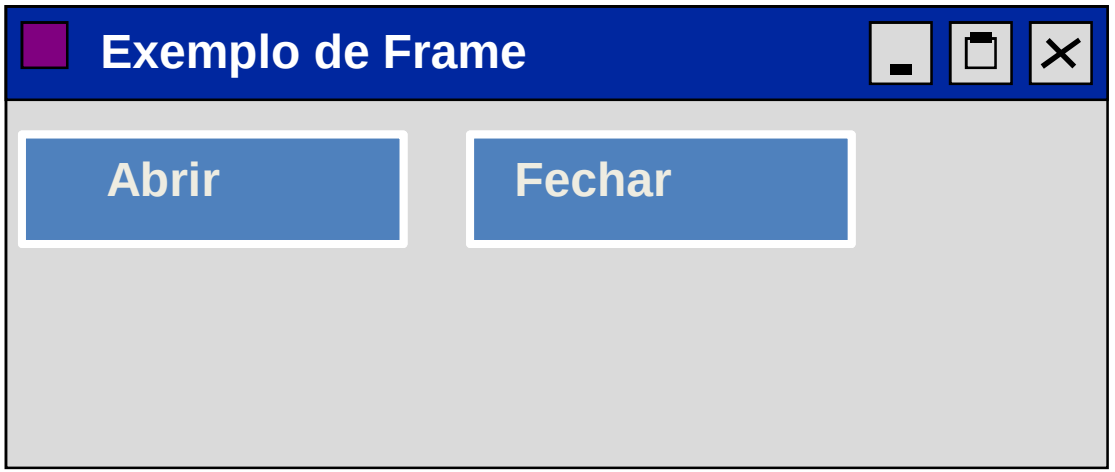
- *Dialog*
  - window modal; pode bloquear o programa até ser fechada pelo usuário



Panels


```
1. Button botaoAB=new Button(“Abrir”);
2. Button botaoFE=new Button(“Fechar”);
3. Panel painel=new Panel();   // Instancia o Panel
4. painel.add(botaoAB);   // Coloca botão Abrir no Panel
5. painel.add(botaoFE);   // Coloca botão Fechar no Panel

...
```

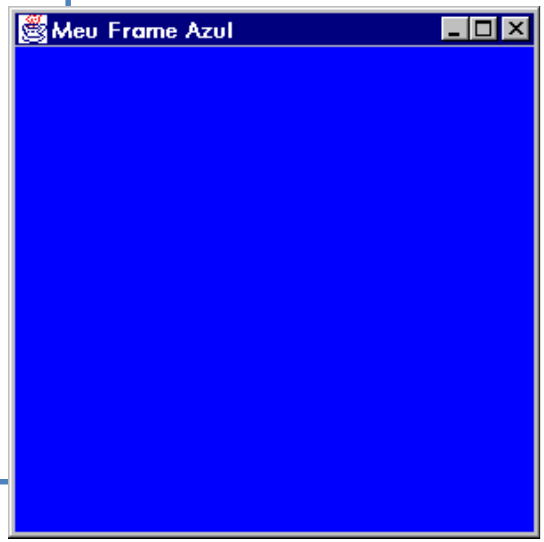
Frame




```
1. Button botaoAB=new Button(“Abrir”);
2. Button botaoFE=new Button(“Fechar”);
3. Frame frame=new Frame(“Exemplo de Frame”);        // Instancia
4.                                                   // Frame com título
5. frame.add(botaoAB);                  // Coloca botão Abrir no frame
6. frame.add(botaoFE);                  // Coloca botão Fechar no frame
7. frame.show();                  // Exibe o frame
8. ...
9. frame.hide();                  // Oculta o frame
```

Criando e exibindo um *container*

```
1. import java.awt.*;
2. class UmFrameTeste extends Frame
3. // a aplicação é uma classe que estende Frame
4. {
5.  public static void main(String args[] )   {
6.       //instancio a classe
7.       UmFrameTeste p1 = new UmFrameTeste();
8.       p1.setBackground(Color.blue);
9.          p1.setTitle("Meu Frame Azul");
10.          // na posicao (0,0)
11.          p1.setSize(300,300);
12.          p1.show();
13.      // intervenção externa para encerrar
14.       }
15.}
```



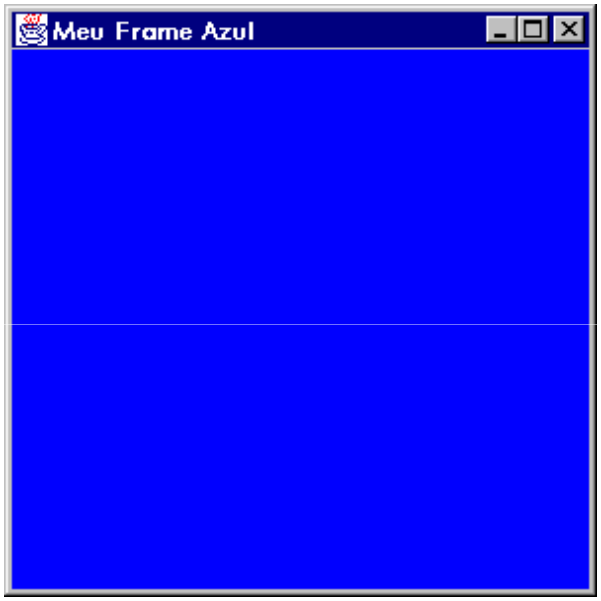
Exibindo mais de um *container*

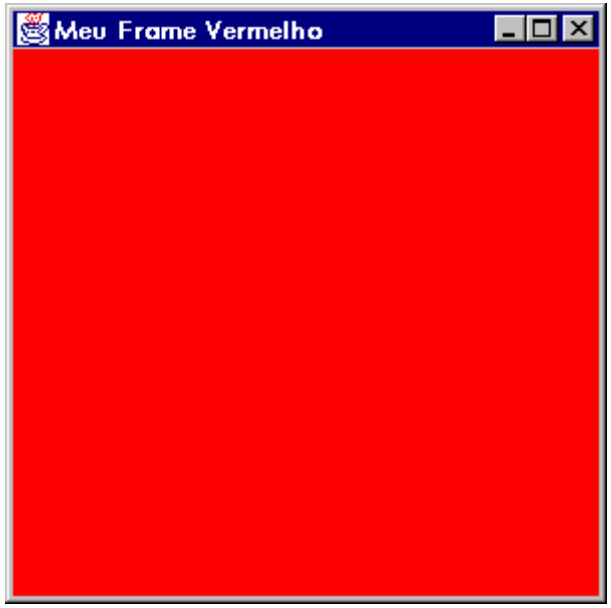
```
import java.awt.*;
class UmFrameTeste extends Frame
// a aplicação é uma classe que estende Frame
{
 public static void main(String args[] )   {
      UmFrameTeste p1 = new UmFrameTeste();
      p1.setBackground(Color.blue);
      p1.setTitle("Meu Frame Azul");
      // na posicao (0,0)
      p1.setSize(300,300);
      p1.show();
       // continua ...
```

... mais de um *container*

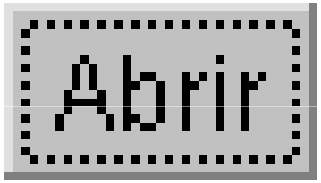
```
   // cria mais uma janela (instancia novamente a mesma
   classe)
1.    UmFrameTeste p2 = new UmFrameTeste();
2.    p2.setBackground (Color.red);
3.    p2.setTitle("Meu Frame Vermelho");
4.    // na posicao (400,400)
5.    p2.setLocation (300, 300);
6.    p2.setSize(300,300);

7.    p1.show();
8.   p2.show();
  // so encerra o programa com intervencao externa
 }
}
```

Resultado





Botões




```
Button botao1=new Button() ;   // Botão sem rótulo
Button botao2=new Button("Fechar");
...
botao1.setLabel("Abrir");    // Altera rótulo do botão
```

Rótulos

```
Label rotulo=new Label() ;  // Sem rótulo
Label  rotulo1=new Label(“Texto”);   // Rótulo Texto
...
rotulo1.setText(“Novo texto”);// Altera texto do rótulo
```

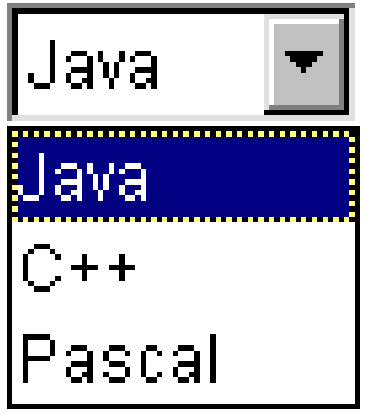
Caixas de seleção


```
Checkbox c1=new Checkbox(); // Caixa sem texto
Checkbox c2=new Checkbox("VISA");
Checkbox c3=new Checkbox("MASTERCARD");
```

Radio Buttons

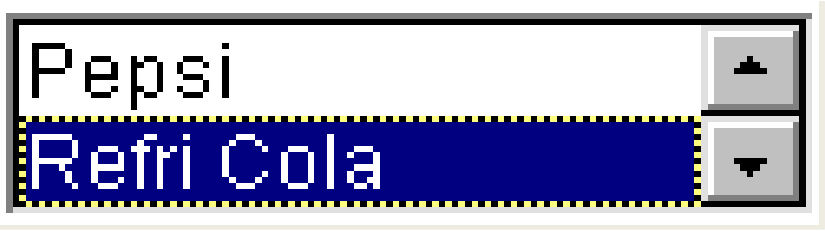

```
CheckboxGroup caixas=new CheckboxGroup();//Grupo de  Botões
...
add(new Checkbox("rapido",caixas,false));
add(new Checkbox("moderado",caixas,false));
add(new Checkbox("lento",caixas,true));
```

Opções (drop-down)




```
1. Choice escolha=new Choice();
2. escolha.addItem("Java");           // Item 0 da l
3. escolha.addItem("C++");    // Item 1 da lista
4. escolha.addItem("Pascal");         // Item 2 da lista
5. ...
6. escolha.select("C++");             // Seleciona item 1
7. escolha.select(2);                 // Seleciona item Pascal
8. int item=escolha.getSelectedIndex(); // Número do item selecionado
9. String sitem=escolha.getSelectedItem();
                              // Retorna String do item selecionado
```

Caixas de seleção

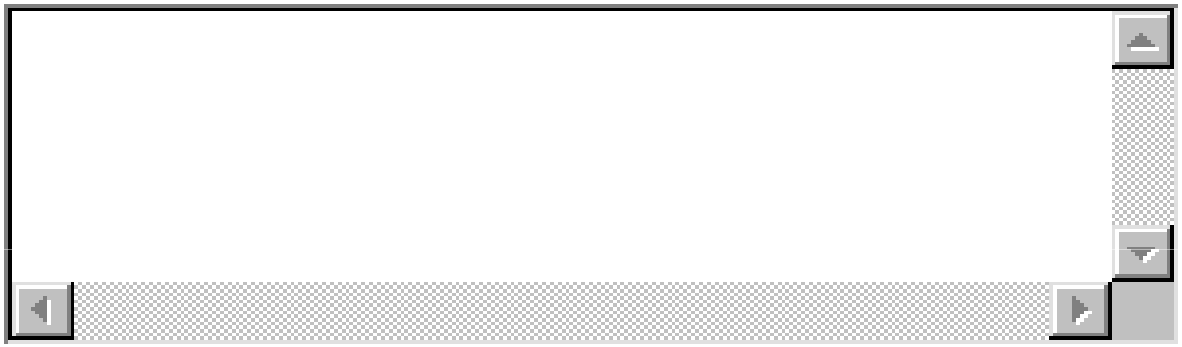



```
1. List lista=new  List(2,true);        // Lista com 2 elementos visíveis
2.                                              // e escolha múltipla
3. lista.addItem("Coca Cola");          // Item 0 da lista
4. lista.addItem("Pepsi");              // Item 1 da lista
5. lista.addItem("Refri Cola");         // Item 2 da lista
6. ...
7. lista.replaceItem("Zero",0);  // Substitui o item 0 da lista por Zero
8. lista.deleteItem(0);                 // Retira o item 0 da lista
9. int selecoes[]=lista.getSelectedIndexes();//Lista de códigos das escolhas
10.String selecoes2[]=lista.getSelectedItems(); // Elementos selecionados
```

Campos de texto

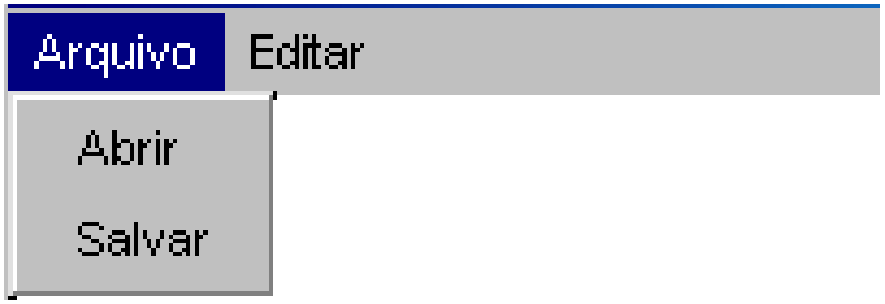
```
TextField texto1=new TextField();// Campo com número de
                                 // colunas não especificado
TextField texto2=new TextField(40);// Campo com 40 colunas
```

Caixas de texto



```
TextArea area=new TextArea(); // Área sem um número especificado
                                 // de linhas e colunas
TextArea area2=new TextArea(5,40); // 4 linhas e 40 colunas
```

Menus




```
1. MenuBar barra_menus=new MenuBar();  // Barra de Menus
2. Menu arquivo=new Menu("Arquivo");  // Grupo de opções de arquivo
3. Menu editar=new Menu("Editar");   // Grupo de opções de edição
4. arquivo.add("Abrir");   // Adiciona a opção Abrir ao grupo Arquivo
5. arquivo.add("Salvar");  // Adiciona a opção Salvar ao grupo Arquivo
6. editar.add("Copiar");   // Adiciona a opção Copiar ao grupo Editar
7. editar.add("Colar");    // Adiciona a opção Colar ao grupo Editar

8. barra_menus.add(arquivo); // Adiciona grupo Arquivo à barra de menus
9.  barra_menus.add(editar); // Adiciona grupo Editar à barra de menus

10. setMenuBar(barra_menus);
```

Incluindo componente em *container*

```
1. import java.awt.*;
2. class UmFrameComComp extends Frame    {
3.   public static void main(String args[ ] )  {
4.        UmFrameComComp p1 = new UmFrameComComp();
5.        p1.setBackground(Color.blue);
6.        p1.setTitle("Meu Frame Azul");
7.        p1.setSize(300,300);
8.        Label mensagem = new Label ("Isto é um Label");
9.        p1.add (mensagem);
10.        p1.setVisible(true);
11. // só encerra o programa com intervenção externa
12. }
13.}
```

Resultado

Incluindo componente (campo de texto) na segunda janela

```
1.    p1.setVisible(true);
2.    // define segunda janela
3.    UmFrameComComp p2 = new UmFrameComComp();
4.    p2.setBackground (Color.red);
5.    p2.setTitle("Meu Frame Vermelho");
6.    p2.setLocation (300,300);
7.    p2.setSize(300,300);
8.    TextField texto = new TextField ("campo de
   texto",20);
9.    texto.setBackground (Color.black);
10.   texto.setForeground (Color.yellow);
11.   p2.add (texto);
12.   p2.setVisible(true);    // mostra a segunda janela
```

Resultado


Meu Frame Azul
Isto é um Label


Meu Frame Vermelho
campo de texto

Mais de um componente no mesmo *container*

```
import java.awt.*;
class UmFrameComLayout extends Frame  {
  public static void main(String args[ ] )   {
1.       UmFrameComLayout p1 = new   UmFrameComLayout();
2.       p1.setBackground(Color.blue);
3.       p1.setTitle("Meu Frame Azul");
4.       p1.setSize(300,300);
5.       Label mensagem = new Label ("Isto é um Label");
6.       p1.add (mensagem);
7.       TextField texto = new TextField ("campo de texto",20);
8.       texto.setBackground (Color.black);
9.       texto.setForeground (Color.yellow);
10.      p1.add (texto);
11.      p1.setVisible(true);   } }
```

Mais de um componente no mesmo *container*

Mais de um componente no mesmo *container*

- Em *frames*, é preciso especificar o arranjo dos componentes na superfície gráfica
  - em *panels* e *applets* existe uma distribuição *default*
- Em Java, isso é feito através de objetos
  - ***LayoutManagers***

Arranjo dos componentes

- O arranjo de vários componentes num *Container* é gerenciado por um objeto *LayoutManager*
  - vantagem: a apresentação dos componentes se adapta quando do redimensionamento da janela
  - desvantagem: pouco domínio que o programador tem da posição dos componentes com alguns *LayoutManagers*

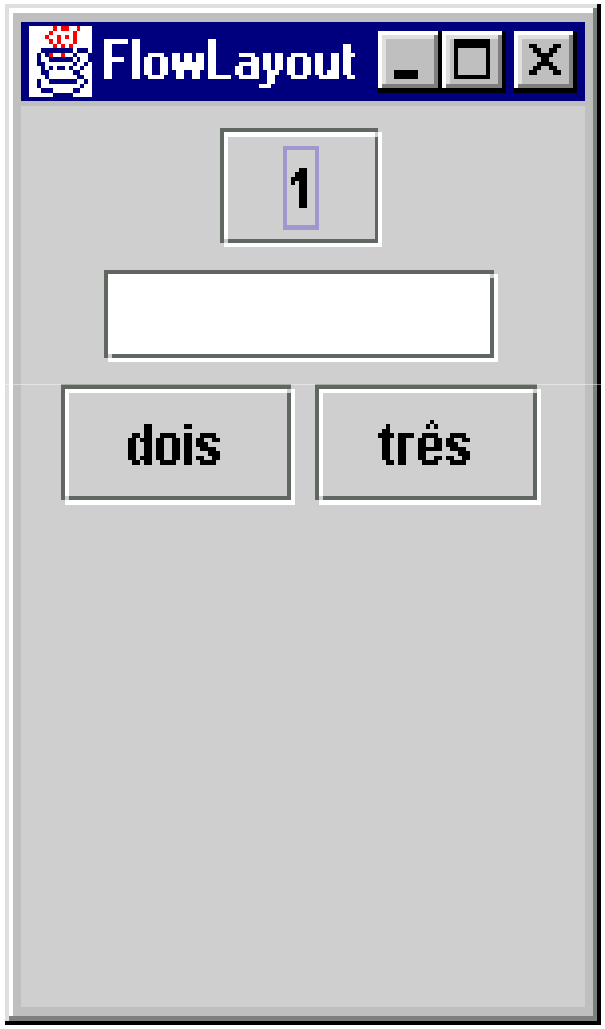
Usando um *gerente de layout*

```
1.  UmFrameComLayout p1 = new UmFrameComLayout();
2.  p1.setLayout (new FlowLayout ());
3.  p1.setBackground(Color.blue);
4.  p1.setForeground (Color.white);
5.  p1.setTitle("Meu Frame Azul");
6.  p1.setSize(300,300);
7.  Label mensagem = new Label ("Nome:");
8.  p1.add (mensagem);
9.  TextField texto = new TextField ("Fulano de Tal",20);
10. texto.setBackground (Color.black);
11. texto.setForeground (Color.yellow);
12. p1.add (texto);
13. p1.setVisible(true);
```

Usando *FlowLayout*

FlowLayout

```
1. setLayout(new FlowLayout());
2. add(new Button("1"));
3. add(new TextField(9));
4. add(new Button("dois"));
5. add(new Button("três"));
```



Respeita o tamanho "preferido" dos componentes mesmo quando não houver espaço suficiente no *container*

É default para *panels*

Agrupando componentes

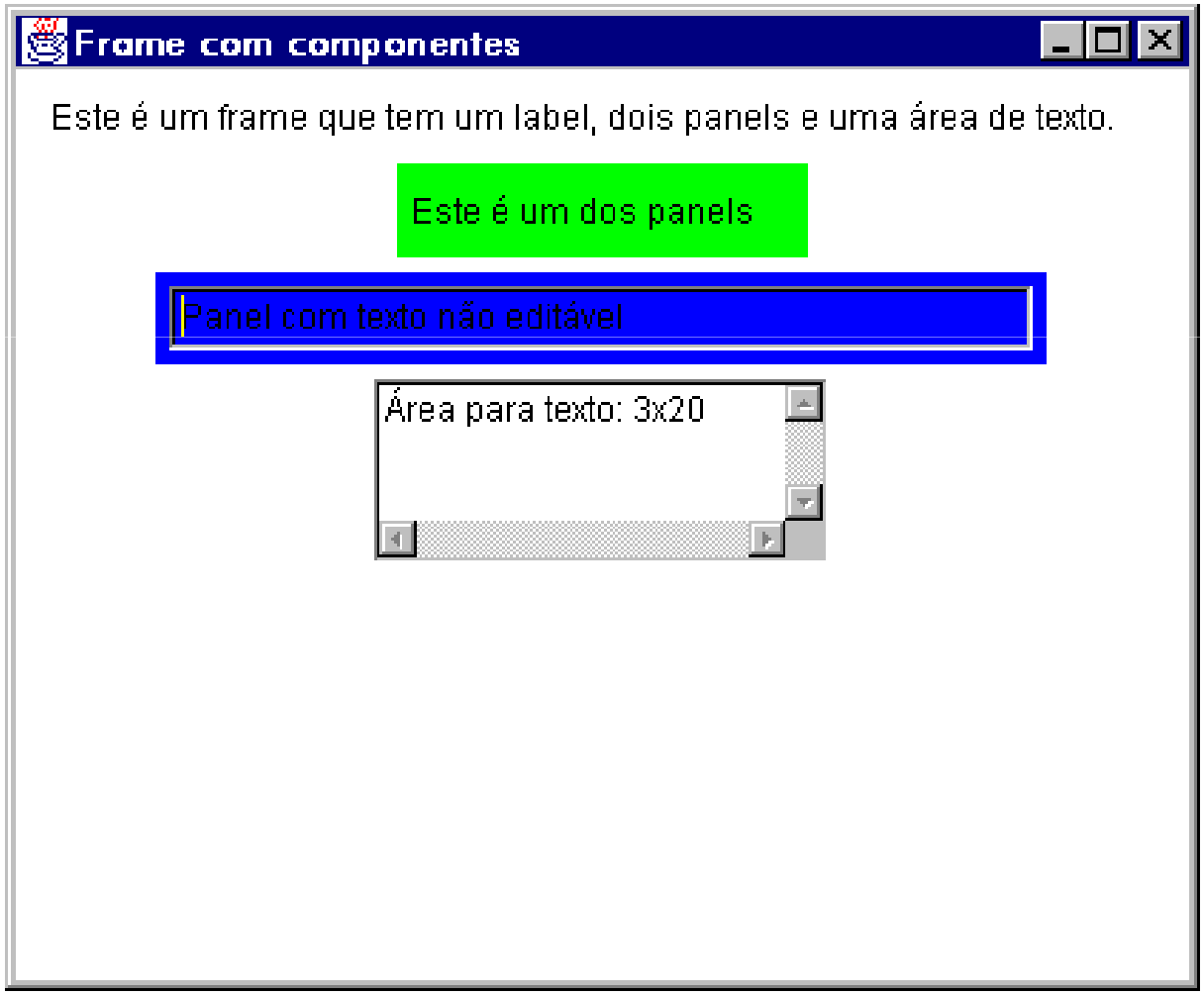
```
import java.awt.*;
class UmFrameComPanels extends Frame {
  public static void main(String args[ ] ) {
1.       UmFrameComPanels ff = new UmFrameComPanels();
2.       ff.setLayout (new FlowLayout());
3.       ff.setBackground(Color.white);
4.       ff.setTitle("Frame com componentes");
5.       ff.setSize(300,300);
6.       ff.add (new Label ("Este é um frame que tem um label,
   dois panels e uma área de texto."));
```

...continua

Agrupando componentes

```
1.  Panel painelMsg = new Panel();
2.              // cria primeiro panel
3.  painelMsg.setBackground(Color.green);
4.  Label msg = new Label ("Este é um dos panels");
5.  painelMsg.add (msg);
6.  ff.add(painelMsg);
7.     //adiciona panel no frame Panel
8.  painelTexto = new Panel();
9.     // cria segundo panel
10. painelTexto.setBackground(Color.blue);
11. TextField texto = new TextField (40);
12. texto.setText("Panel com texto não editável");
13. texto.setEditable(false);
14. painelTexto.add (texto);
15. ff.add(painelTexto);
16.    // adiciona panel no frame
17. ff.add(new TextArea("Área para texto:3x20",3,20));
18. ff.show(); } }
```

Frame com *panels*



FlowLayout x BorderLayout

BorderLayout

- A área é dividida em 5 partes
  - norte, sul, leste, oeste e centro
- Apenas um componente pode ser inserido em cada área
- Na adição do componente, especificar a sua localização no container usando:
  - "North", "South", "East", "West", "Center"
- Partes não utilizadas ficam vazias.

Usando *BorderLayout*

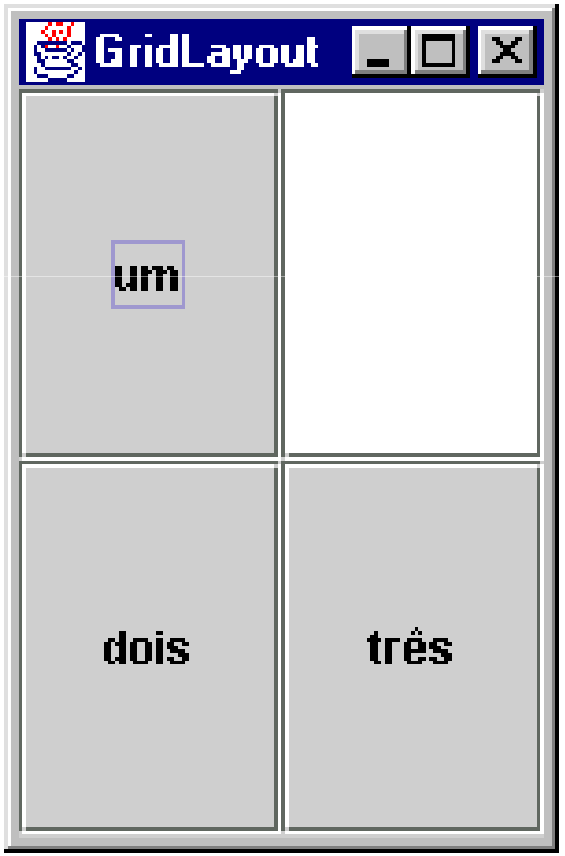
```
ff.setLayout (new BorderLayout());
ff.setBackground(Color.white);
ff.setTitle("Frame com componentes");
// na posicao (0,0)
ff.setSize(600,300);
ff.add ("North", new Label ("Este é um frame
                    que tem um label, dois
                    panels e uma área de
                    texto."));
Panel painelMsg = new Panel();
painelMsg.setBackground(Color.green);
Label msg = new Label ("Este é um dos panels");
painelMsg.add (msg);
ff.add("West",painelMsg);
                    // continua
```

Usando *BorderLayout*

```
1.      Panel painelTexto = new Panel();
2.      painelTexto.setBackground (Color.blue);
3.      TextField texto = new TextField(mens);
4.      texto.setEditable(false);   // o textField não é editável
5.      painelTexto.add (texto);
6.      ff.add("East", painelTexto);
7.      ff.add("South", new TextArea("Área para texto: 3x20“,3,20));
```

GridLayout

```
1. Panel c =new Panel();
2. c.setLayout(new GridLayout(2,2));
3. c.add(new Button("um"));
4. c.add(new TextField(5));
5. c.add(new Button("dois"));
6. c.add(new Button("três"));
```



- Divide a área em uma grade
- Dispõe os elementos da esquerda para a direita e de cima para baixo
- Todos tem mesmo tamanho

CardLayout

- Os componentes são "empilhados" no *container*
- Ficam visíveis através de métodos
  - first (Container c), last (Container c)
  - next (Container c), previous (Container c)
  - show (Container c, String componente)
- Os componentes são adicionados especificando a sua ordem "na pilha"

Usando o *CardLayout*

```
public static void main(String args[])
 {     FrameComCards ff = new FrameComCards ();
1.         CardLayout leiaute = new CardLayout ();
2.         ff.setLayout (leiaute);
3.         ff.setBackground(Color.white);
4.         ff.setTitle("Frame com componentes");
5.         ff.setSize(600,300);
6.         ff.add (new Label ("Este é um frame que tem um
   label, dois panels e uma área de texto."),"One");
7.         Panel painelMsg = new Panel();
8.         painelMsg.setBackground(Color.green);
9.         Label msg = new Label ("Este é um dos panels");
10.         painelMsg.add (msg);
```

Usando o *CardLayout*

```
1.    ff.add (painelMsg, "Two");
2.    Panel painelTexto = new Panel();
3.    painelTexto.setBackground (Color.blue);
4.    TextField texto = new TextField(mens);
5.    texto.setEditable(false);
6.    painelTexto.add (texto);
7.    ff.add(painelTexto,"Three");
8.    ff.add(new TextArea("Área para texto: 3x20",
   3,20),"Four");
9.    ff.show(); // mostra o primeiro componente
10.    leiaute.last(ff);  // mostra o último
}
```

Usando o *CardLayout*

Outros *layout managers*

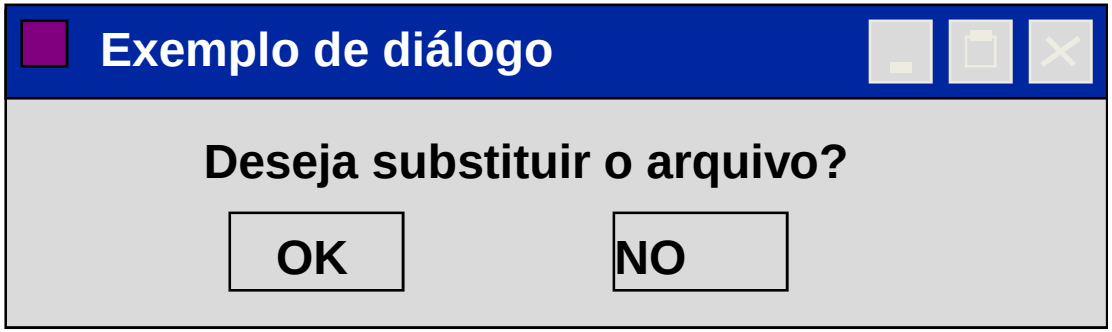
- *GridBagLayout*
  - a superfície é dividida numa grade
  - células podem ter tamanhos diferentes
  - componentes podem ocupar mais de uma célula
  - definido com uma série de parâmetros
- BoxLayout (com Swing)
- custom (com Swing)

Dialogs

Diálogos podem ser:

modais: impedem a entrada para outras janelas enquanto estão ativos.

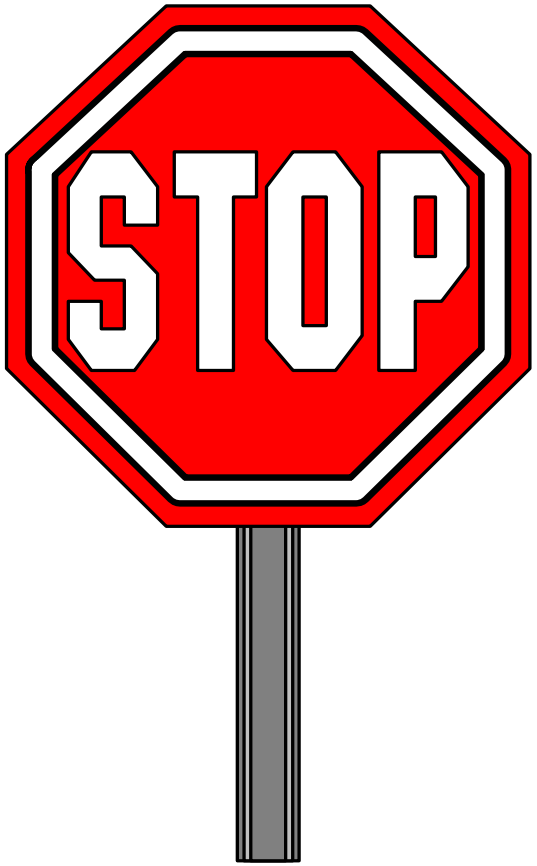
não-modais: não impedem a entrada para outras janelas.




```
Button botaoOK=new Button("OK");
Button botaoNO=new Button("NO");
Frame frame=new Frame("Exemplo de Frame");     // Instancia frame com título
Dialog dialogo;                                // Referência para diálogo
frame.add(new Label("Deseja substituir o arquivo? ");
frame.add(botaoOK);                        // Coloca botão Abrir no frame
frame.add(botaoNO);                        // Coloca botão Fechar no frame
dialogo=new Dialog(frame,"Exemplo de diálogo",true);    // Cria diálogo
                       |             |              |
              frame referência   título   modal ( false=não modal)
dialogo.show();                            // Mostra diálogo
```

Exercício 1

- exemplos de componentes
- abrindo mais de uma janela
- gerente de layout
- agrupando componentes



EVENTOS

Interação com elementos da interface

- Exemplo 1: Fechamento, minimização, iconização de janela
  - que parte do programa atende as requisições do usuário?
- Exemplo 2: Botões da interface
  - como capturar qual botão foi pressionado?
- Exemplo 3: Eventos do mouse

Programação orientada a eventos

- Interfaces gráficas de manipulação direta são dirigidas por eventos.
  - Eventos correspondem à ocorrência de algum fato, o qual deve (ou não) desencadear um procedimento específico
- Não há, em princípio, uma sequência para a execução de procedimentos.
  - Procedimentos são acionados dependendo dos eventos provocados pelo usuário.

Tratamento de eventos

- Cada vez que o usuário pressionar um botão, digitar um caractere ou movimentar o mouse, por exemplo, ocorre um evento.
  - O sistema operacional se encarrega de converter essas ações em eventos

Tratamento de eventos

- Para cada evento que o sistema operacional é capaz de gerar, a aplicação deve prever um tratamento correspondente ("o que deve acontecer em resposta").
- Em Java a ação resposta padrão para qualquer evento é "não fazer nada".

Tratamento de eventos em Java

- Java utiliza um método de delegação de eventos
  - objetos geradores de evento são fontes de eventos
  - objetos que recebem eventos são *listeners* (ouvintes)
  - deve ser determinado qual objeto receberá os eventos gerados por outro(s) objeto(s)
    - o programador deve registrar os objetos *listeners* junto aos objetos fontes de eventos

4 Componentes envolvidos no processo de tratamento de eventos

1. Origem do evento: é um componente onde o evento é gerado.

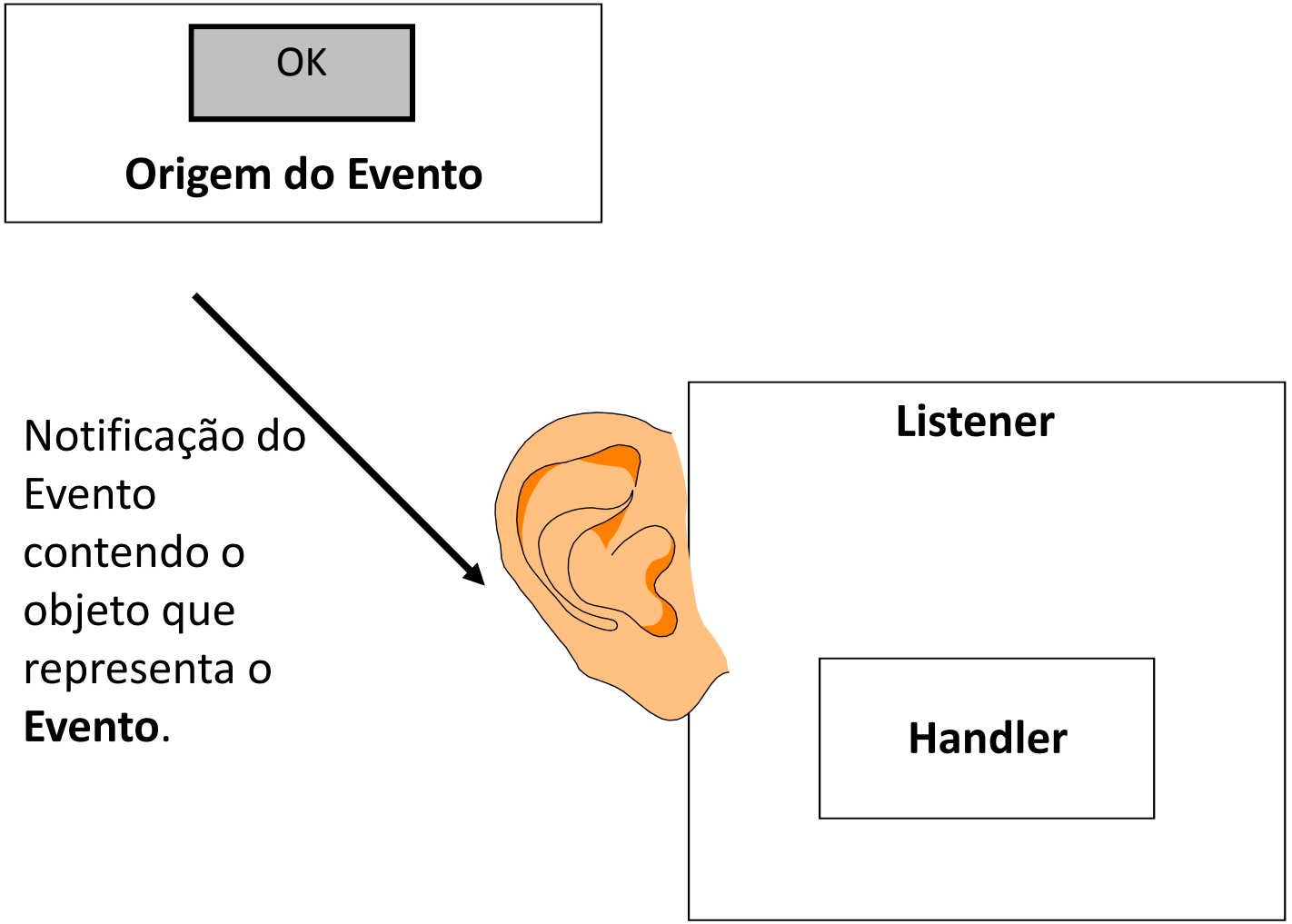
   Ex.: um evento de clique do mouse pode ser originado de um botão

2. Evento: é um objeto que representa o evento gerado.

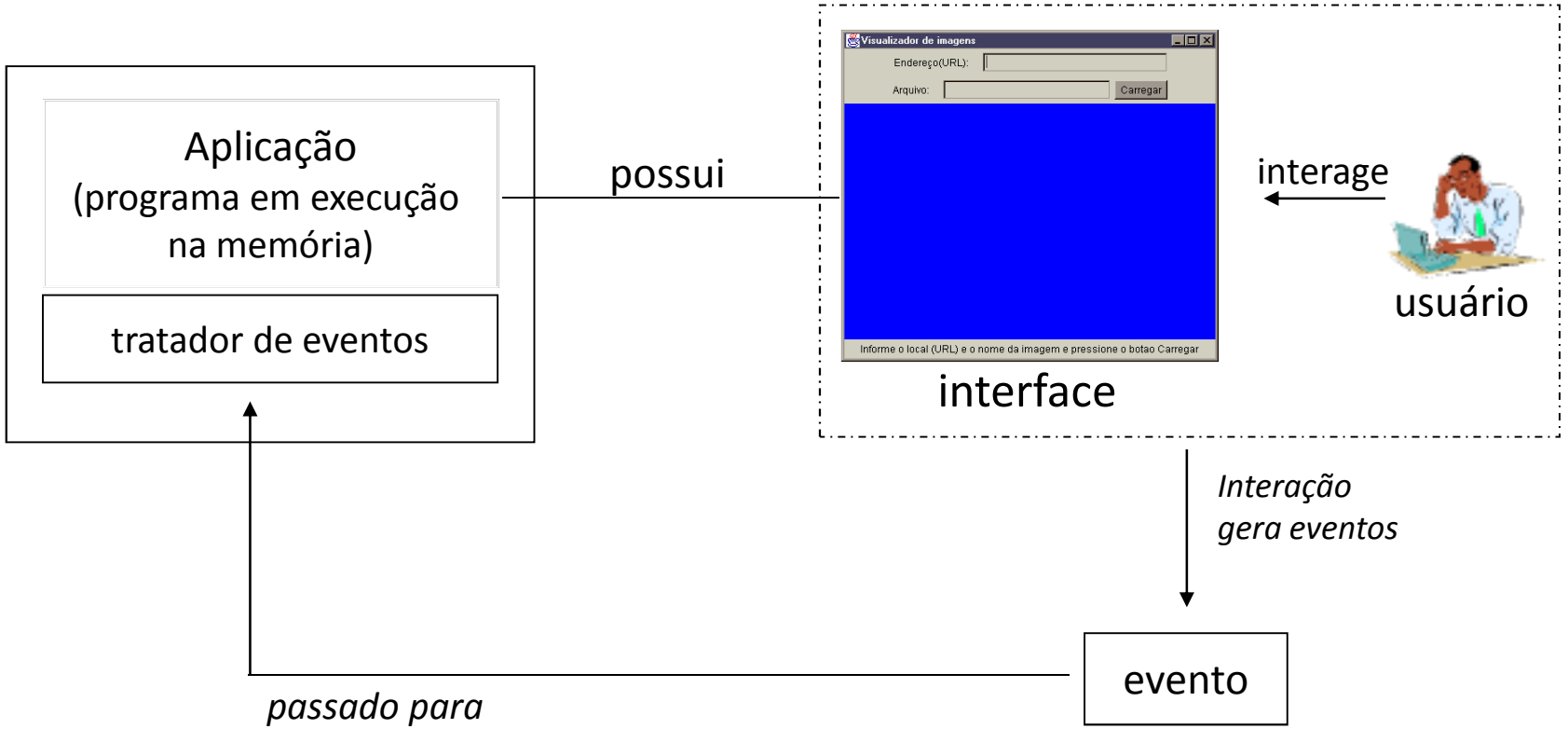
3. *Listener* ou Receptor de Evento: é um objeto responsável por "ouvir" ou "captar" a ocorrência do evento.

4. *Handler*: é um método do *Listener* responsável por executar as ações decorrentes do evento.

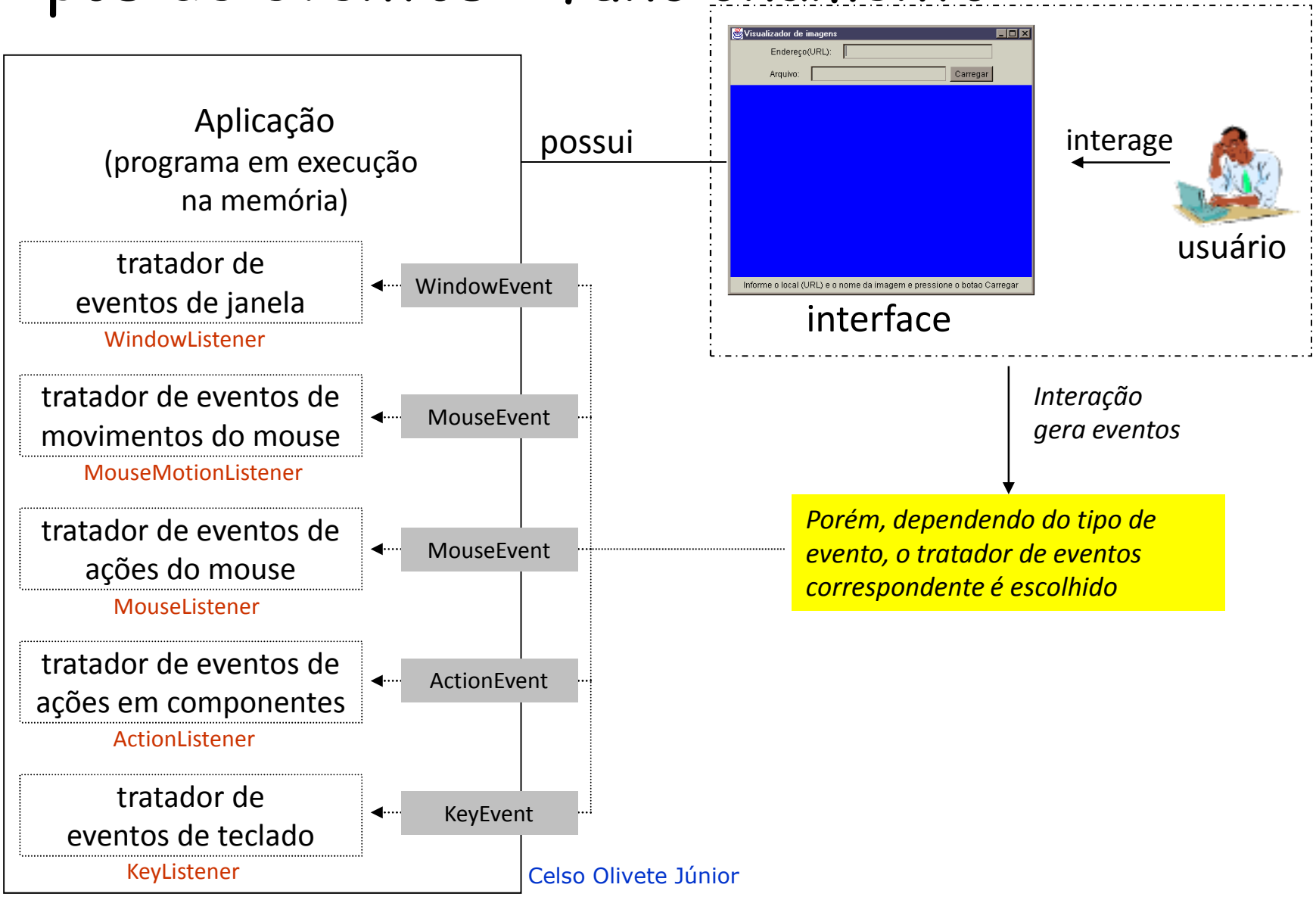
Tratamento de Eventos



Tipos de eventos - funcionamento



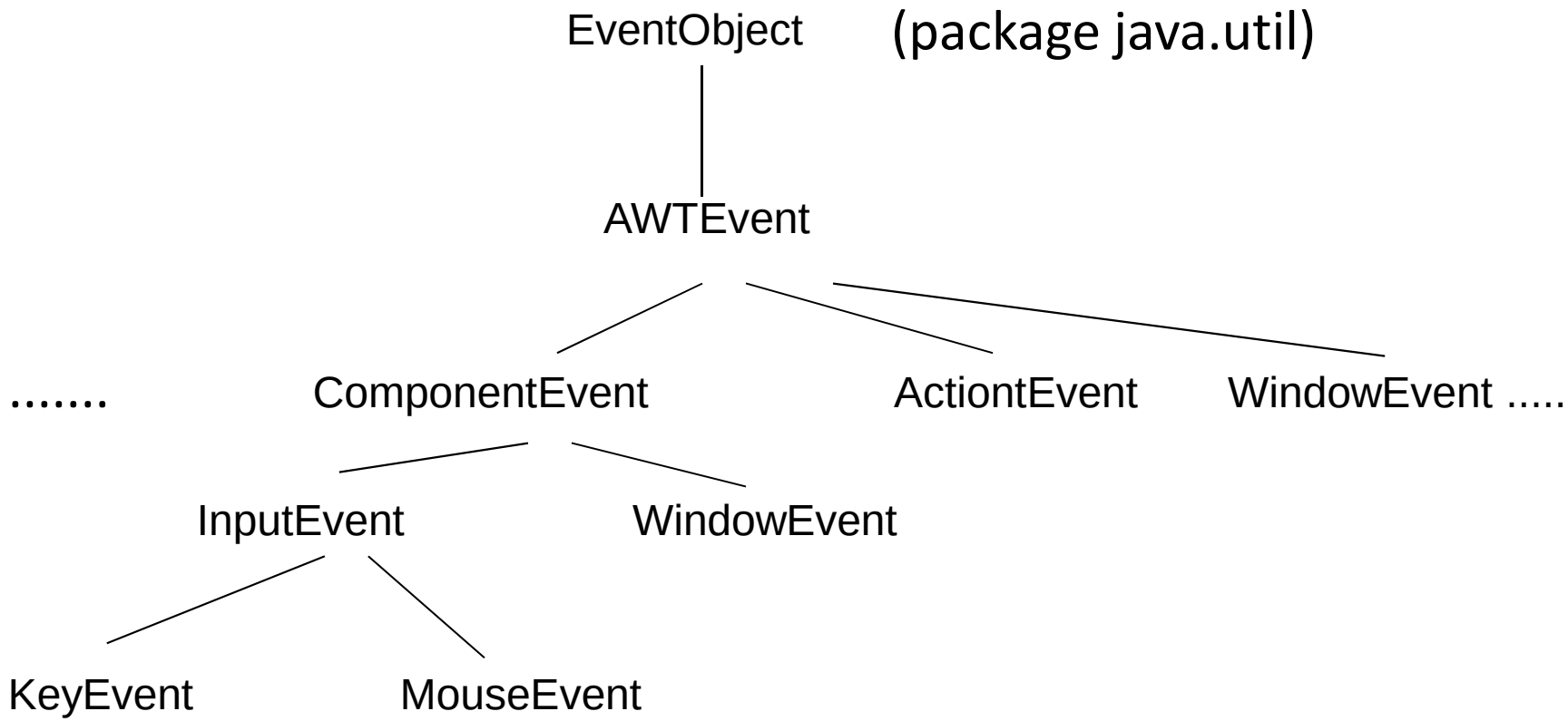
Tipos de eventos - funcionamento



Eventos em Java

- Eventos são objetos de subclasses de java.awt.AWTEvent
- Objetos AWTEvents têm métodos para
  - getSource () - devolver o objeto originador do evento
  - getID() - devolver o tipo do evento
  - outros, dependentes do evento
    - getX(), getY() - devolver a posição de clique do mouse

Classe de Eventos em Java



Exemplos de eventos

- Divididos em categorias (java.awt.event)
  - ActionEvent (componentes de ação)
  - MouseEvent (componentes afetados pelo mouse)
  - ItemEvent (checkboxes e similares)
  - AdjustmentEvent (scrollbars)
  - TextEvent (componentes de texto)
  - WindowEvent (janelas)
  - FocusEvent (componentes em geral)
  - KeyEvent (componentes afetados pelo teclado)
  - ...

Listeners

- *Listeners* são objetos que recebem eventos
- Devem implementar um conjunto de métodos que são invocados automaticamente quando da ocorrência do evento
  - tais métodos estão definidos em interfaces específicas para cada tipo de evento

Interfaces

- Classes que definem especificações e implementações de métodos
- Uma ***interface*** também define um tipo, a exemplo das classes, mas através de especificações (*sem implementação*)
  - Uma interface contém somente constantes e assinaturas de métodos
    - é implementada por uma classe

Interfaces

- Uma classe "implementa" uma interface fazendo a implementação de todos os métodos da interface
- Uma classe pode implementar mais de uma interface, além de estender uma classe

Tratando eventos de janela

- Janelas geram eventos do tipo *WindowEvent* (fechamento, iconização, ...)
- Eventos de janela devem ser tratados por classes que implementam a interface
  - WindowListener
- Métodos: windowClosing, windowClosed, windowOpened, ...

Mais interfaces...

• Cada modalidade de evento tem uma classe de interface "Listener" correspondente:

- ActionEvent: ActionListener
- MouseEvent: MouseListener e MouseMotionListener
- ItemEvent: ItemListener
- AdjustmentEvent: AdjustmentListener
- TextEvent: TextListener
- WindowEvent: WindowListener
- FocusEvent: FocusListener
- KeyEvent: KeyListener
- ...
- XXXEvent: XXXListener

• Cada "Listener" possui um ou mais métodos que devem ser implementados na sua utilização. (veja a tabela do próximo slide)

Tipos de eventos

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Button TextField MenuItem	ActionListener	.addActionListener()	actionPerformed (ActionEvent e)
List ComboBox	ItemListener	.addItemListener()	stateChanged (ChangeEvent e)
key on component	KeyListener	.addKeyListener()	keyPressed(), keyReleased(), keyTyped()
mouse on component	MouseListener	.addMouseListener()	mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()
mouse on component	MouseMotionListener	.addMouseMotionListener()	mouseMoved(), mouseDragged()
Frame	WindowListener	.addWindowListener()	windowClosing(WindowEvent e), ...

Implementando um *listener*

- **Crie uma nova janela onde deseja implementar o(s) listener(s) desejado(s)**

```
public class MyClass extends Frame implements ActionListener,ItemListener
{ ...}
```

- **Implemente cada um dos métodos da(s) interface(s)**

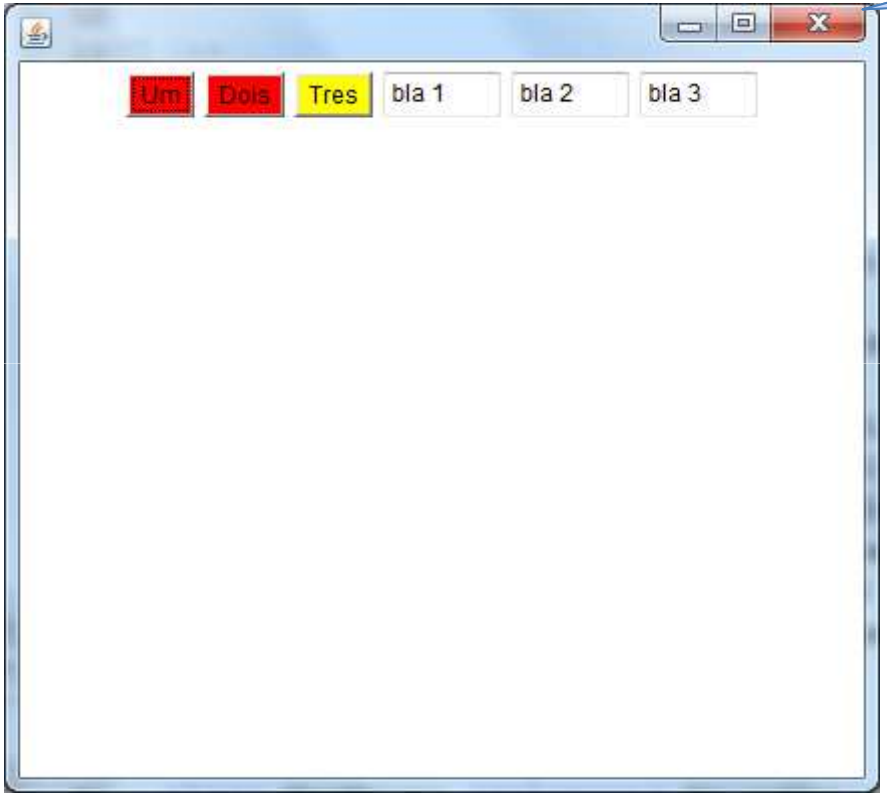
```
public void actionPerformed(ActionEvent e) { ...}
public void itemStateChanged(ItemEvent e) {...}
```

  - Veja a documentação sobre o listener usado e o evento correspondente (para saber que métodos usar para obter suas informações)
  - Todos os métodos são **public void**
  - Todos recebem o tipo de evento correspondente ao tipo do listener como argumento

- **Cadastre os listeners: vincule cada componente com o método a ser executado (evento):**
  - componente.addxxxListener (referência para aonde o listener foi implementado);
  - Ex: **button.addActionListener(this);**

- **Na ocorrência de um evento, todos os listeners registrados serão notificados**

Exemplo: tratando eventos da janela



Ao clicar em Minimizar, maximizar ou fechar – nada será feito. Necessário tratar esses eventos

- Janelas geram eventos do tipo *WindowEvent*
- Devem ser tratados por classes que implementam uma interface
  - WindowListener
  - Métodos: windowClosing, windowClosed, windowOpened, ...

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Frame	WindowListener	`.addWindowListener()`	`windowClosing(WindowEvent e), ...`

Definindo interface e implementando os métodos adequados

```
import java.awt.*;
import java.awt.event.*;      // importa
class Fechamento implements WindowListener
{
1.    public void windowClosing (WindowEvent e)
2.    {     System.exit (0);   }
3.    public void windowClosed (WindowEvent e){}
4.    public void windowIconified (WindowEvent e){}
5.    public void windowOpened (WindowEvent e){}
6.    public void windowDeiconified (WindowEvent e){}
7.    public void windowActivated (WindowEvent e){}
8.    public void windowDeactivated (WindowEvent e){}
}
```

Listener que manipula eventos do Frame

Implementando os métodos do listener - handler

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Frame	WindowListener	.addWindowListener()	windowClosing(WindowEvent e), ...

Tratando eventos de janela

```
public class UmFrameQueFecha extends Frame
 {
1.     public static void main(String args[])  {
2.           UmFrameQueFecha p1 = new UmFrameQueFecha();
3.           p1.setBackground(Color.blue);
4.           p1.setTitle("Meu Frame Azul");
5.           p1.setSize(300,300);
6.     // registra uma instância de Fechamento como listener
7.     // dos eventos de janela gerados por p1
8.           p1.addWindowListener (new Fechamento ());
9.           p1.show();
  }
}
```

Cadastra o *listener*: vincula o Frame com o método a ser executado (evento)

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Frame	WindowListener	.addWindowListener()	windowClosing(WindowEvent e), ...

Gerador e *listener* de eventos como um único objeto

Listener que manipula eventos do Frame

Implementando os métodos do listener - handler

```
import java.awt.*;
import java.awt.event.*;      // importa
public class UmFrameQueFecha extends Frame
                              implements WindowListener
 {
1.      public void windowClosed (WindowEvent e){}
2.      public void windowIconified (WindowEvent e){}
3.      public void windowOpened (WindowEvent e){}
4.      public void windowDeiconified (WindowEvent e){}
5.      public void windowActivated (WindowEvent e){}
6.      public void windowDeactivated (WindowEvent e){}
```

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Frame	WindowListener	.addWindowListener()	windowClosing(WindowEvent e), ...

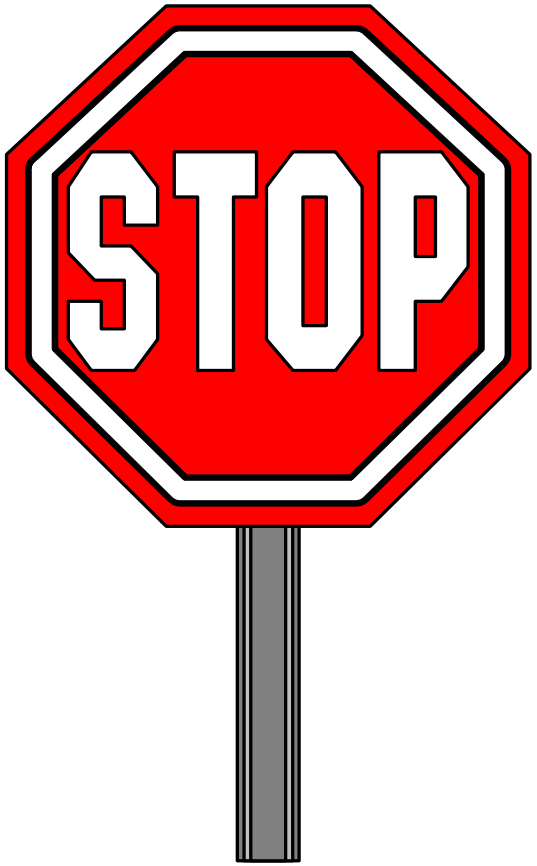
Gerador e *listener*

```
public void windowClosing (WindowEvent e)
       {     System.exit (0);   }
public static void main(String args[])  {
1.           UmFrameQueFecha p1 = new UmFrameQueFecha();
2.           p1.setBackground(Color.blue);
3.           p1.setTitle("Meu Frame Azul");
4.           p1.setSize(300,300);
5.          // registra p1 como listener
6.       // dos eventos de janela gerados pelo próprio  p1
7.           p1.addWindowListener (p1 );
8.           p1.show();
  }
} //fecha a classe UmFrameQueFecha
```

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Frame	WindowListener	.addWindowListener()	windowClosing(WindowEvent e), ...

Exercício 2

- implementar o "fechar frame"



- Exemplo 2: Botões
  - Como capturar o botão clicado?
    - necessário implementar o listener (ouvinte) ActionListener e o método actionPerformed(...) e vincular o botão ao listener - addActionListener(botão)

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Button TextField MenuItem	ActionListener	.addActionListener()	actionPerformed (ActionEvent e)

Exemplo 2: botões

```
1. public class JavaApplication1 extends Frame
2.              implements WindowListener, ActionListener


3. public void actionPerformed(ActionEvent e) {
4.         String botao_clicado = e.getActionCommand();
5.         System.out.print("clicou"+botao_clicado)
 }
```

Listeners para tratar eventos referentes aos eventos da Janela e dos botões

Implementa o método para o listener ActionListener

Neste exemplo, exibirá b1 ou b2, conforme o botão clicado

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Button TextField MenuItem	ActionListener	.addActionListener()	actionPerformed (ActionEvent e)

```
1.      public static Button b1,b2;
2.      public static JavaApplication1 frame;
3.      public static void main(String[] args) {
4.      frame = new JavaApplication1();
5.      frame.setSize(500,500);
6.      //associa o frame ao listener
7.      frame.addWindowListener(frame);
8.      frame.setLayout(new FlowLayout());
9.      b1 = new Button(); b2 = new Button();
10.     b1.setLabel("Botao 1"); b2.setLabel("Botao 2");
11.     b1.setActionCommand("b1"); b2.setActionCommand("b2");
12.     //instalar o listener - acoes
13.     b1.addActionListener(frame);
14.     b2.addActionListener(frame);
15.     frame.add(b1);
16.     frame.add(b2);
17.     frame.show();
18.   }
```

Agora, no main, é necessário criar o Frame, os botões e registrar os **listeners**

Registrar o listener responsável pelo fechamento

Configura as ações dos botões

Registra o listener que responde as ações dos botões

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
Button TextField MenuItem	ActionListener	.addActionListener()	actionPerformed (ActionEvent e)

Exemplo 3: Eventos do mouse

- interfaces

Componentes afetados	"Listener"	addxxxListener	Métodos do Listener
mouse on component	MouseListener	`.addMouseListener()`	`mouseClicked()`, `mouseEntered()`, `mouseExited()`, `mousePressed()`, `mouseReleased()`
mouse on component	MouseMotionListener	`.addMouseMotionListener()`	`mouseMoved()`, `mouseDragged()`

Exemplo 3: Recupera a posição X,Y do mouse sobre o Frame

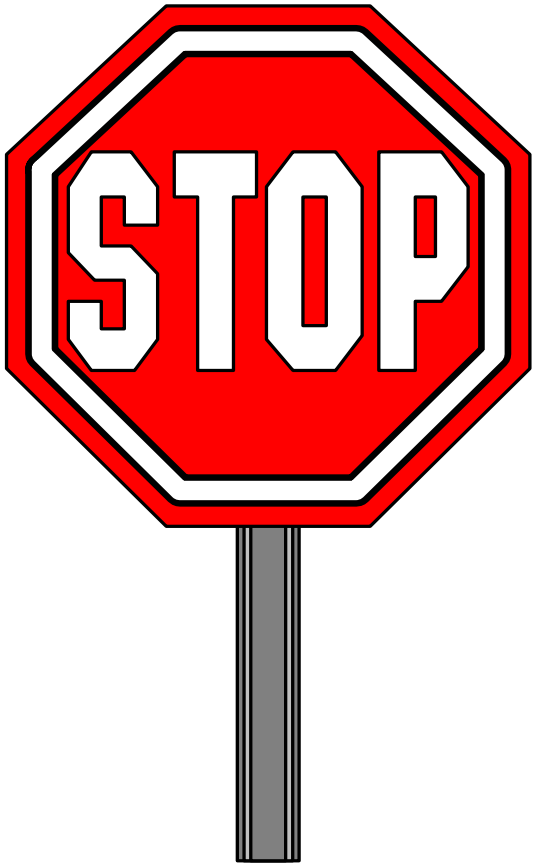
```
1. public class JavaApplication1 extends Frame
2.              implements MouseMotionListener {

3. public void mouseMoved(MouseEvent e) {
4.         int posX = e.getX();
5.         int posY = e.getY();
6.         System.out.print("X = " + posX + " Y = "+ posY);
7.     }
8. public static JavaApplication1 frame;
9. public static void main(String[] args) {
10.       frame = new JavaApplication1();
11.       frame.setSize(500,500);
12.       //associa o frame ao listener
13.       frame.addMouseMotionListener(frame);
14.         …
15.     }
16.}
```

Implementa o método que responde ao evento – movimento do mouse

Registra o listener que responde as ações de movimentos do mouse

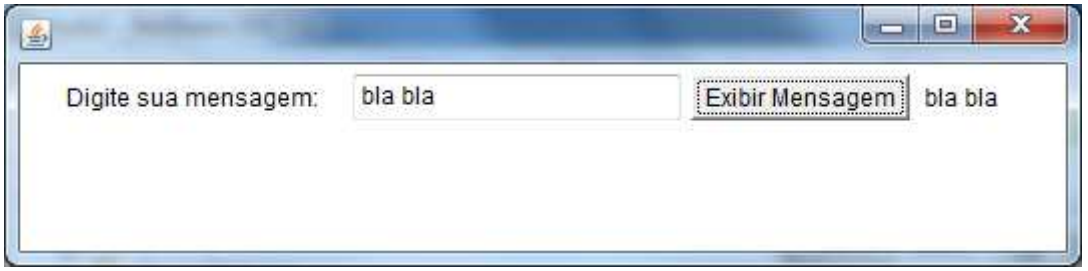
Exercícios

- Exemplos de componentes
- Tratando eventos de janela
- Tratando eventos de botão
- Abrindo mais de uma janela



Exercício

2. Faça um Frame com um componente Panel que contenha dois componentes Labels, um Button e um TextField para a entrada do texto. Após pressionar o Button, a mensagem deve ser exibida em um Label, conforme figura. Para isso, deve ser desenvolvido o **actionPerformed**



Exercício

3. Implemente uma calculadora simples (com operações de adição, subtração e multiplicação) para números inteiros.