

Java

Swing, Programação por Eventos

Programação III
José Luis Oliveira; Carlos Costa

Introdução

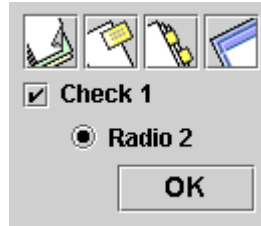
- A primeira interface gráfica de utilizador (GUI) incluída na versão 1.0 foi designada por AWT (Abstract Windows Toolkit).

`java.awt.*`

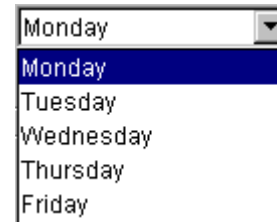
- A ideia do AWT era criar interfaces que fossem boas em qualquer plataforma.
 - O resultado foi interfaces fracas em todos os sistemas :(
- A versão 2.0 do JDK passou a incluir uma nova API designada por Java Foundation Classes (JFC)
 - Swing
 - `javax.swing.*`
 - AWT
 - Java2D
 - Drag-and-Drop

Elementos de uma aplicação gráfica (1)

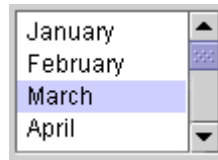
- Buttons



- Combo boxes



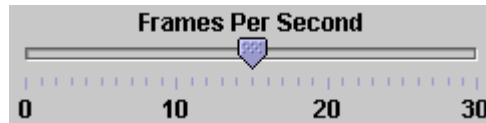
- Lists



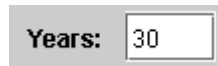
- Menus



- Sliders



- Text Fields



- Labels

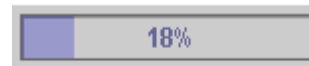


Elementos de uma aplicação gráfica (2)

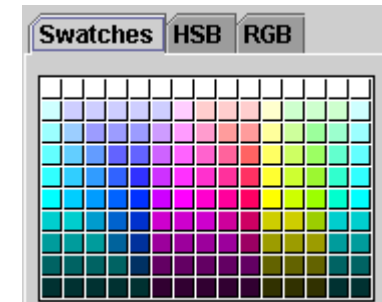
- Tool tips



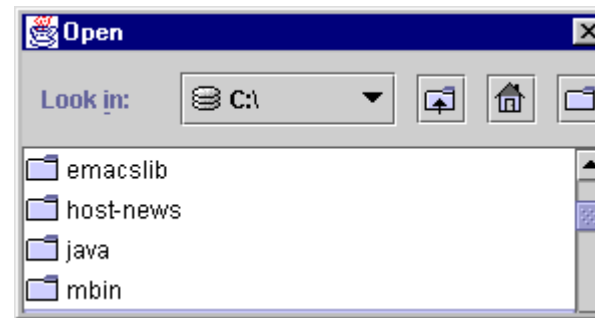
- Progress bars



- Colour choosers



- File choosers




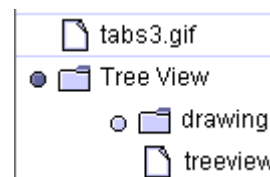
- Tables

- Text

- Trees



First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

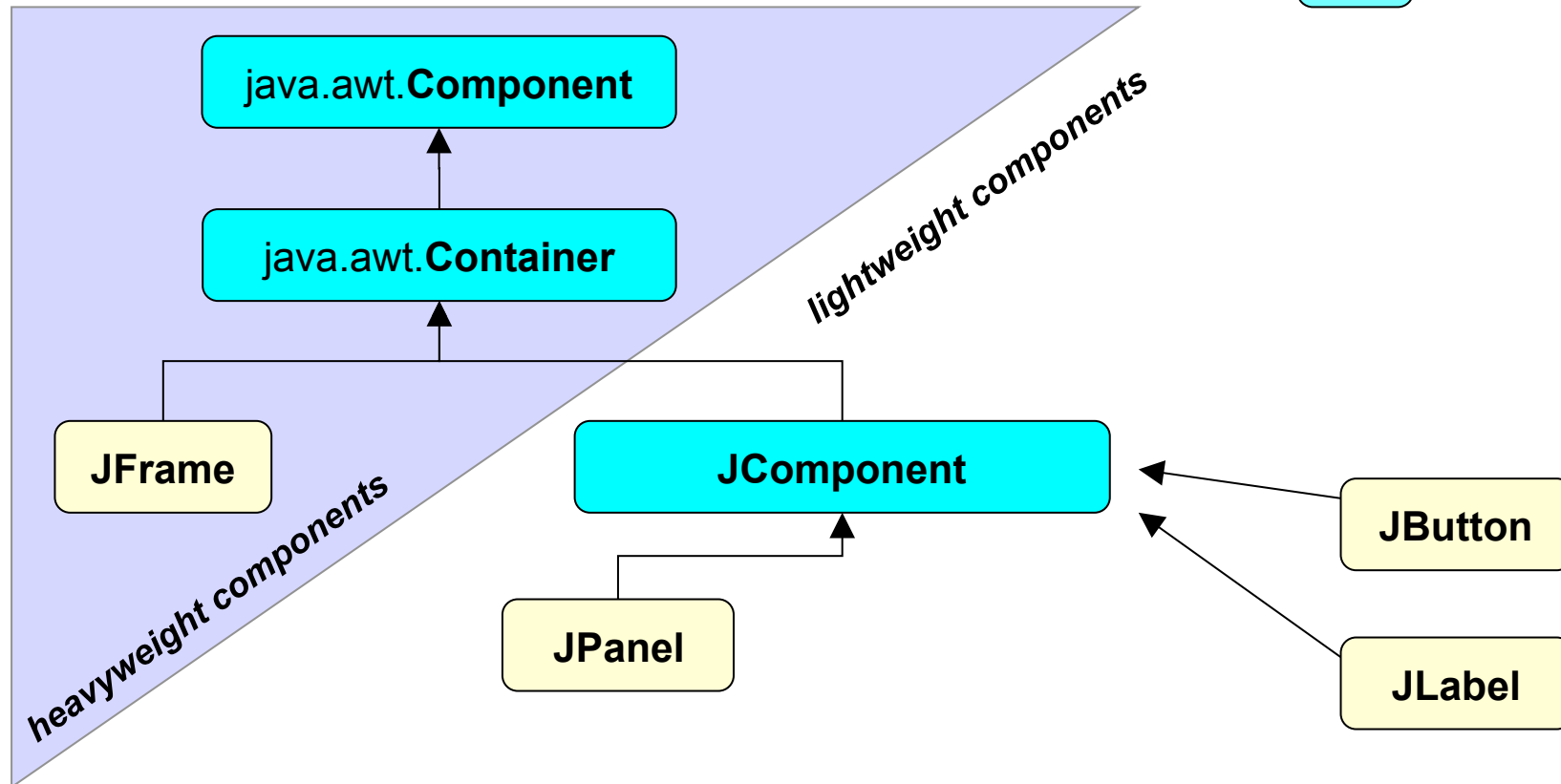


Elementos de uma Aplicação Swing

- Componentes gráficos
 - Exemplos: windows, buttons, labels, text fields, menus, ...
 - Herdam de `javax.swing.JComponent` (a um nível mais elevado de `java.awt.Component`)
- Contentores (Containers)
 - Um Container é um componente onde podem ser colocados outros componentes.
 - Exemplos : `JFrame` (contentor principal), `JPanel` (contentor secundário)
- Eventos
 - Permite lidar com acções do utilizador sobre a interface gráfica
 - `java.awt.event.*`

Swing GUI

- Os componentes Swing são herdados da árvore



- As classes base definem muitas das funcionalidades encontradas nos diversos componentes Swing.

AWT - Classes principais

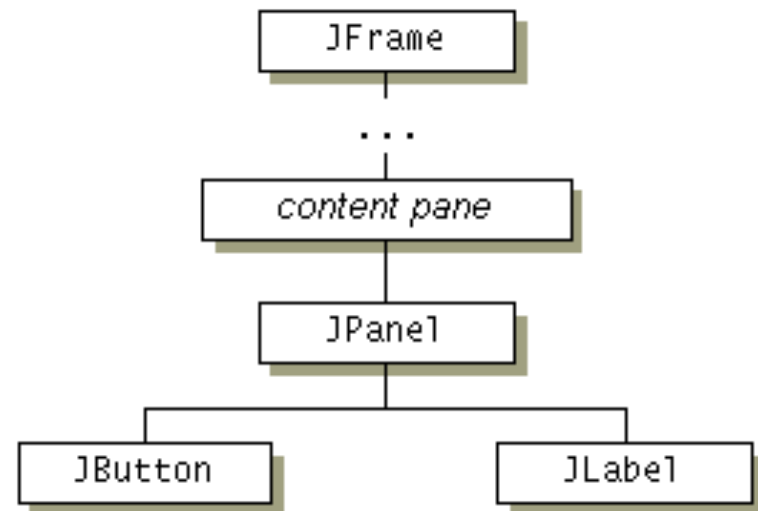
- java.awt.Component
 - Classe abstracta → Todos os elementos gráficos são Component
 - alguns métodos:

```
void setSize(int width, int height);  
void setBackground(Color c);  
void setFont(Font f);  
void setVisible();
```
- java.awt.Container
 - Classe abstracta
 - alguns métodos:

```
void setLayout(LayoutManager lman);  
void add(Component c);
```

Containers

- Os Containers servem para organizar e gerir todos os componentes de uma aplicações Swing
- Top-Level
 - JFrame
 - JDialog
 - JApplet
 - JWindow
- Qualquer aplicação swing deve usar um objecto JFrame (ou derivado) como container de topo



JFrame

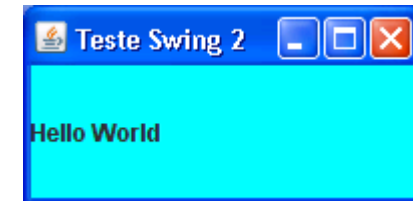
```
import javax.swing.*;
```



```
public class Win1 {  
    public static void main(String args[]) {  
        JFrame frame = new JFrame("Teste Swing");  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setVisible(true);  
    }  
}
```

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

```
public class Win2 {  
    public static void main(String args[]) {  
        JFrame frame = new JFrame("Teste Swing 2");  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setSize(200,100);  
        JLabel label = new JLabel("Hello World");  
        label.setBackground(Color.CYAN);  
        label.setOpaque(true);  
        frame.getContentPane().add(label);  
        frame.setVisible(true);  
    }  
}
```



Painéis - Containers de agregação

- Os Painéis são usados dentro de um container de topo para agrupar outros componentes

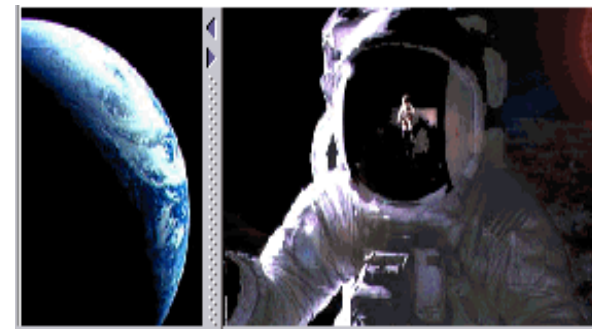
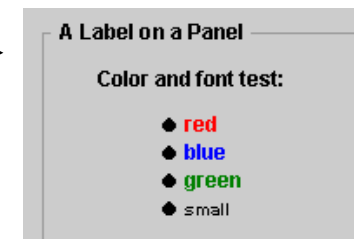
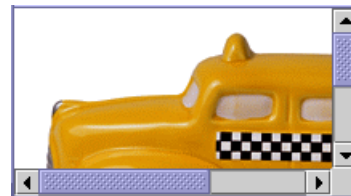
- JPanel

- JScrollPane

- JSplitPane

- JTabbedPane

- JToolBar



JPanel

- O JPanel é um componente:
 - onde se pode desenhar
 - que pode conter outros componentes
 - que permite criar sub-áreas dentro da janela principal

```
//Create a panel and add components to it.  
JPanel painel = new JPanel();  
painel.add(someComponent);  
painel.add(anotherComponent);  
  
//Make it the content pane.  
painel.setOpaque(true);  
topLevelContainer.setContentPane(painel);
```

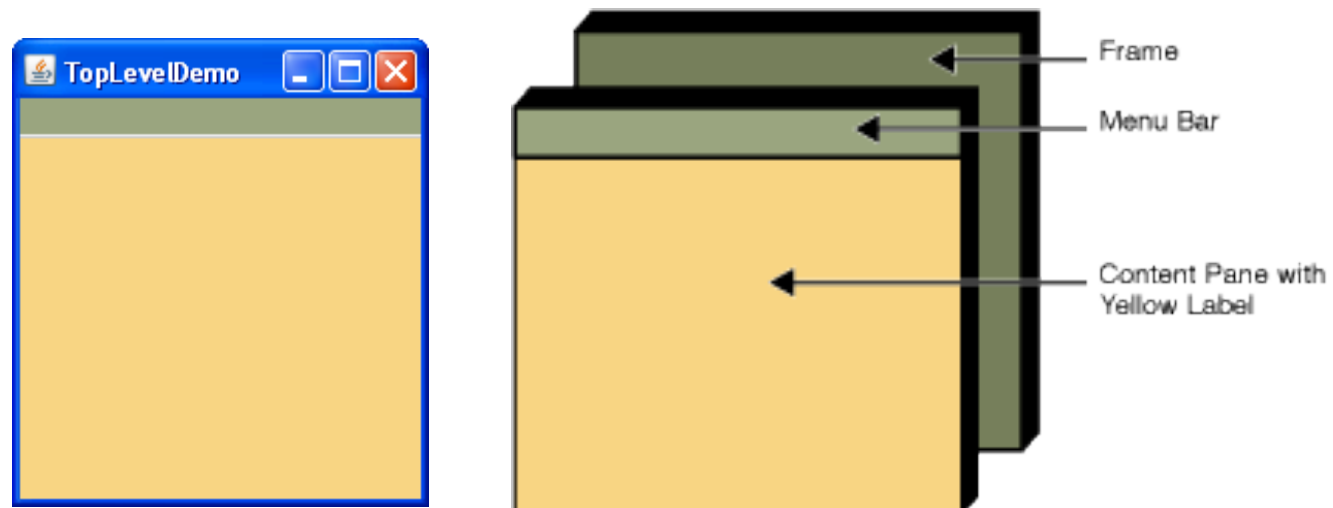
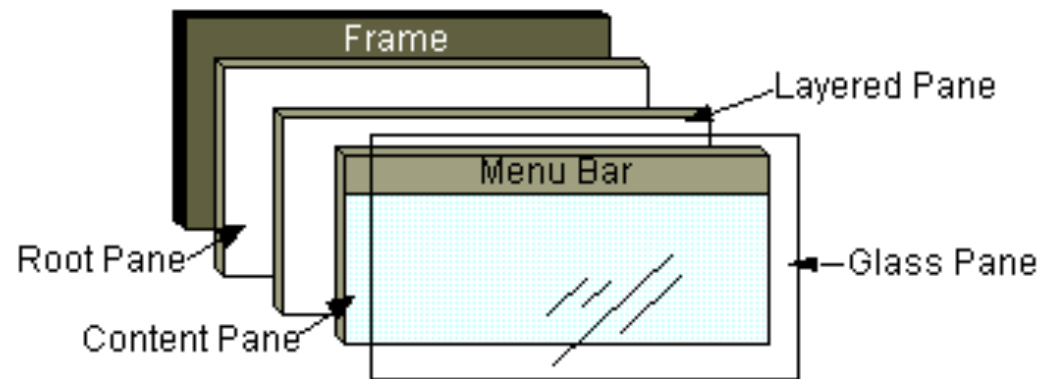
JPanel

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

public class Win3 {
    public static void main(String[] args) {
        JFrame f = new JFrame("Teste Swing 3");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(400, 150);
        JPanel content = new JPanel();
        content.setBackground(Color.white);
        content.setLayout(new FlowLayout());
        JButton b1 = new JButton("Button 1");
        JButton b2 = new JButton("Button 2");
        content.add(b1);
        content.add(b2);
        f.setContentPane(content);
        f.setVisible(true);
    }
}
```

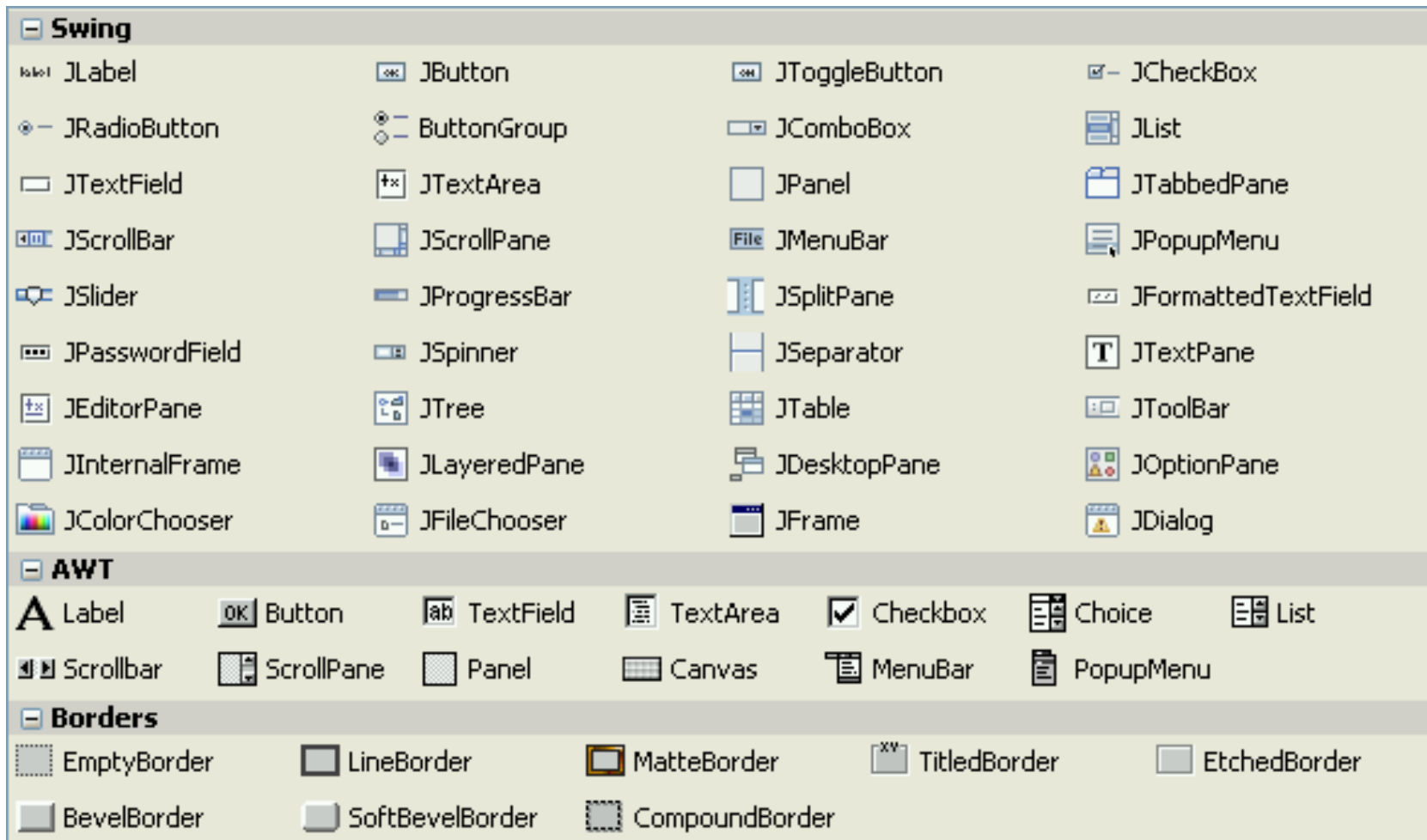


JFrame e JPanel



Componentes

- Com excepção dos componentes de topo, todos os outros componentes Swing (J*) derivam de JComponent



Componentes - Exemplos

- JLabel

- Usado para colocar texto num container. Serve essencialmente de rótulo a outros componentes

```
JLabel();  
JLabel(String texto);  
void setText(String novoTexto);
```

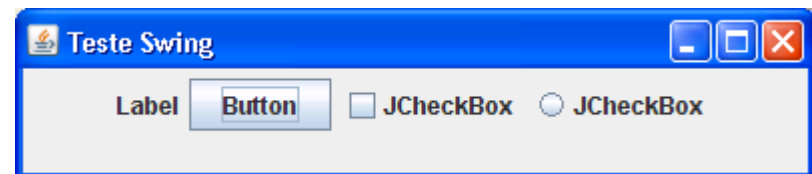
- JButton

- Cria botões. Sempre que um botão é pressionado gera um evento.

```
JButton();  
JButton(String texto);  
void setText(String novoTexto);  
void setMnemonic(char c);
```

- JCheckBox, JRadioButton

```
JCheckBox();  
JCheckBox(String texto);  
JCheckBox(String texto, boolean state);  
boolean isSelected();  
void setSelected(boolean state);
```

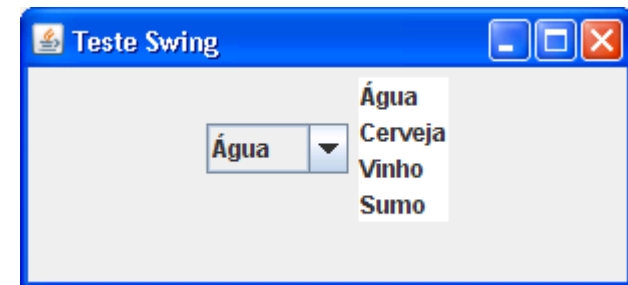


Componentes - Exemplos

- JComboBox

- usado para seleccionar uma opção de uma lista apresentada na forma de pop-up

```
JComboBox() ;  
JComboBox(Object[] itens) ;  
void addItem( Object item ) ;  
Object getSelectedItem() ;  
int getSelectedIndex() ;  
void setEditable( boolean edit ) ;  
void setSelectedIndex( int index ) ;
```



- JList

- usado para fazer selecções de uma lista de items.
- O utilizador observa uma janela de opções e pode seleccionar vários itens.

```
JList() ;  
JList( Object [] itens ) ;  
int setListData(Object [] itens ) ;  
void setSelectionMode( int mode ) ;  
Object getSelectedValue() ;  
Object [] getSelectedValues() ;
```


Componentes - Exemplos

- JTextField

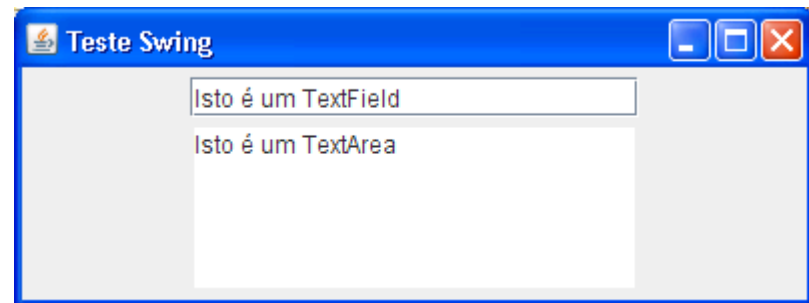
- Apresenta uma linha de texto

```
JTextField();  
JTextField( int cols );  
JTextField( String text, int cols );  
String  getText();  
boolean isEditable();  
void    setEditable( boolean editable );  
void    setText( String text );
```

- JTextArea

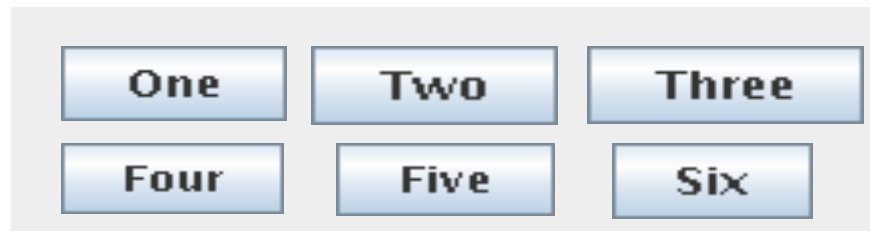
- Apresenta uma área com múltiplas linhas de texto
- semelhante a JTextFields

```
int      getRows();  
void     setRows( int rows );
```



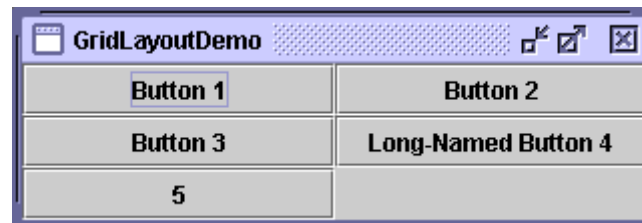
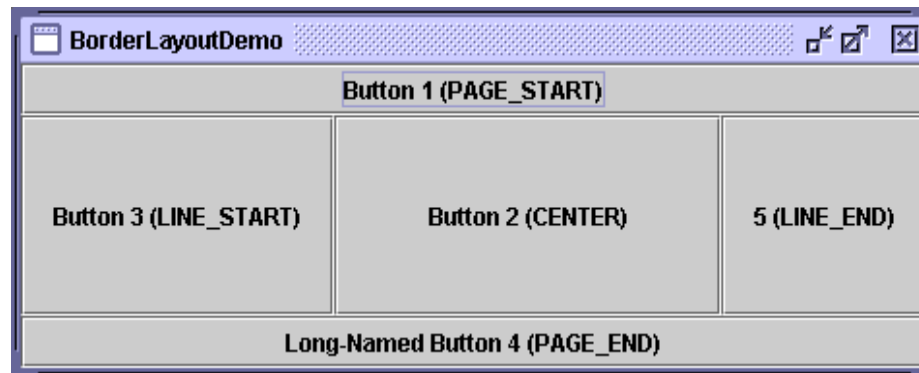
Organização de componentes - Layout

- Como dispor os diversos componentes num painel?
 - Podemos definir as localizações manualmente
 - ou ... usar `java.awt.LayoutManagers`!
- Cada painel contém um objecto que implementa a interface `LayoutManager`
 - Permite organizar os componentes automaticamente
 - Por omissão é `java.awt.FlowLayout`, que arruma os componentes da esq. para a dir. e de cima para baixo
- Podemos passar um `LayoutManager` no construtor do painel ou invocar o método `setLayout`



Layout Managers

- Existem diversos tipos de layout. Exemplos:
 - FlowLayout
 - BorderLayout
 - GridLayout



Border layout

- O *BorderLayout* divide o painel em 5 regiões
 - NORTH, SOUTH, EAST, WEST, CENTER



- Na adição de um componente ao contentor é necessário indicar a região

```
panel.add(dp, java.awt.BorderLayout.CENTER);
```

Border Layout

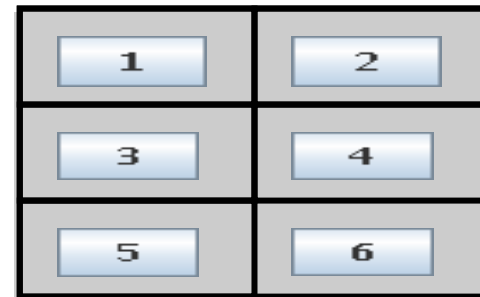
- Não é necessário preencher todas as áreas
 - Cada região tem dimensão (0, 0) por omissão
 - O BorderLayout ajusta automaticamente os componentes
- Exemplos:



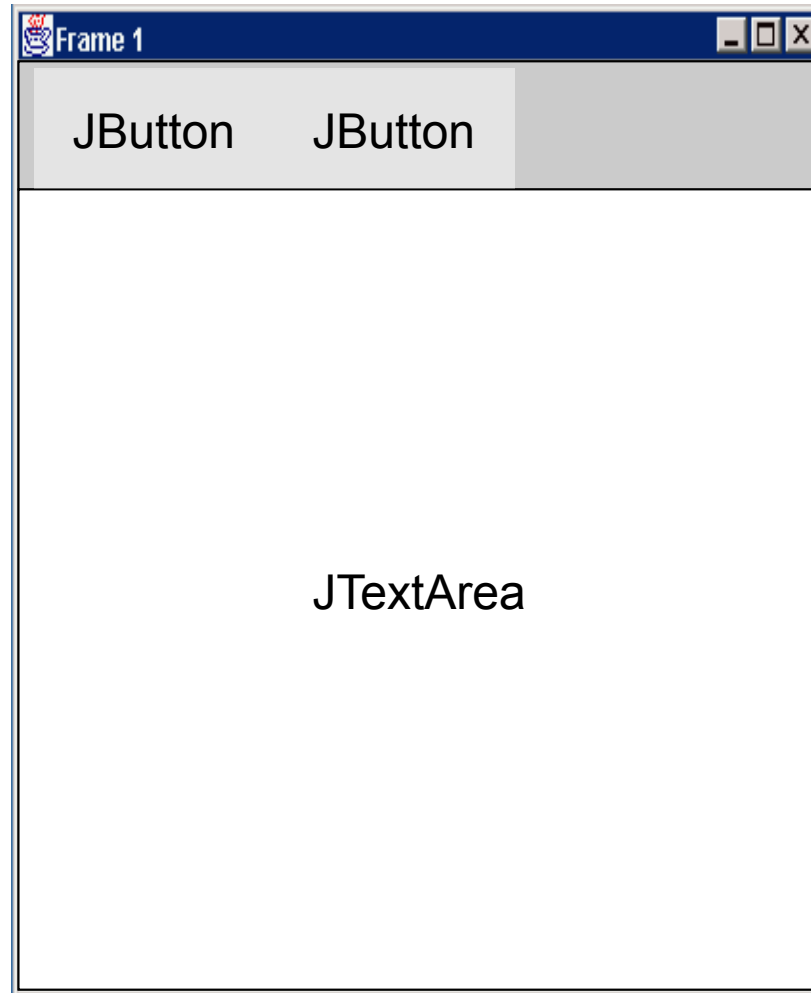
Grid Layout

- Dispõe os componentes segundo uma grelha de linhas e colunas
- Redimensiona cada componente de forma a que todos tenham a mesma dimensão
 - igual ao elemento maior
- Os componentes são adicionados linha a linha da esquerda para a direita (e de cima para baixo)
- Exemplo:

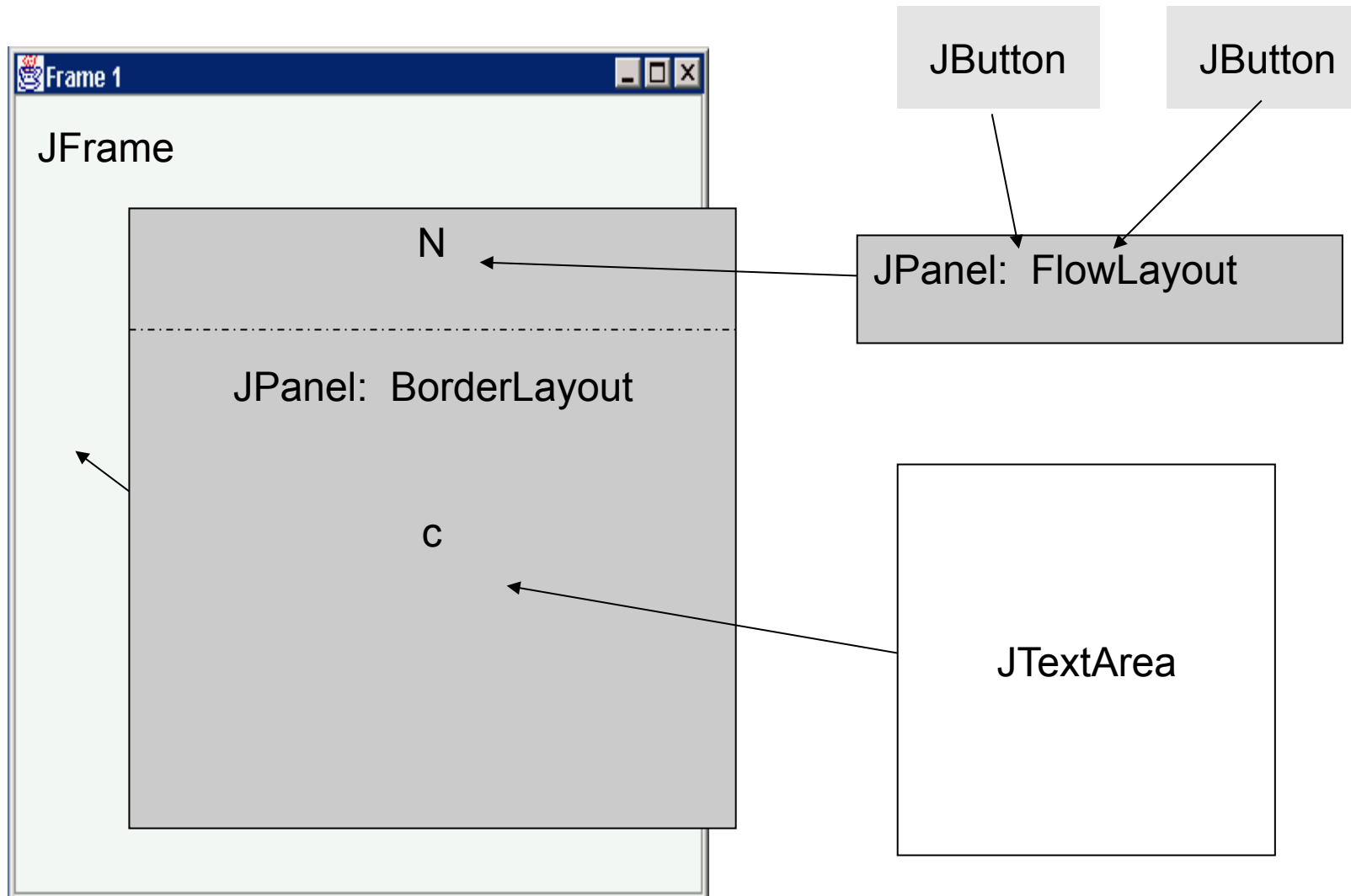
```
JPanel gpanel = new JPanel();  
gpanel.setLayout(new GridLayout(3, 2));  
gpanel.add(new JButton("1"));  
gpanel.add(new JButton("2"));  
gpanel.add(new JButton("3"));  
gpanel.add(new JButton("4"));  
gpanel.add(new JButton("5"));  
gpanel.add(new JButton("6"));
```



Planificação da interface



Planificação da interface

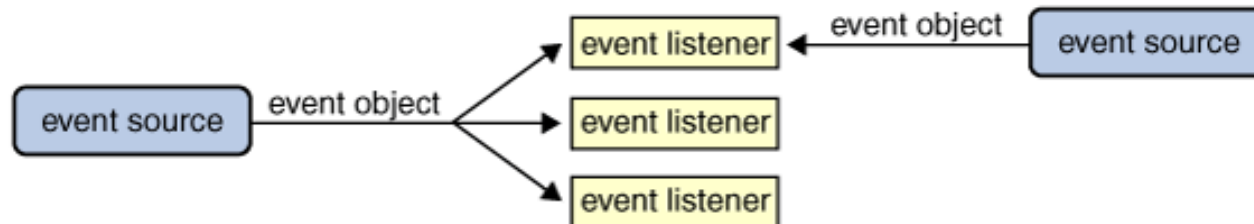


Programação por Eventos

- Já vimos como construir janelas, painéis, componentes,
...
 - mas precisamos ainda de saber como lidar com eventos (selecção de um menu, premir de um botão, arrasto do rato, ..)

Eventos

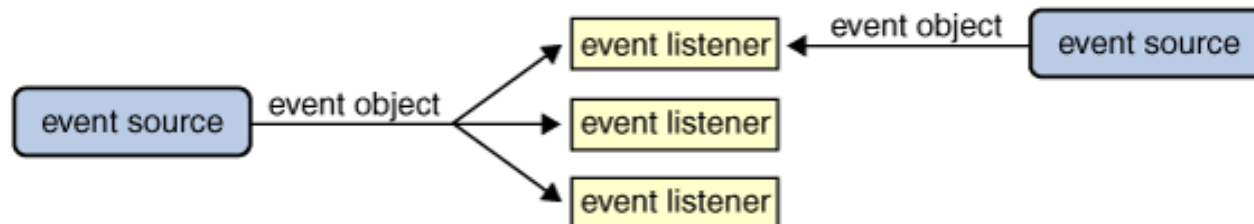
- O que são?
 - Objectos gerados quando uma determinada acção ocorre (teclado, rato, etc.) e que descrevem o que sucedeu
 - Existem vários tipos de classes de eventos para tratar as diversas categorias de acções desencadeadas pelo utilizador
 - Subclasses de `java.util.EventObject`
- Quem gera o evento?
 - Componentes (**event source**).
 - Todo o evento tem associado um objecto fonte (quem gerou)
`Object fonte = evento.getSource();`



Eventos

- Quem recebe e trata o evento?
 - Objectos receptores/ouvintes (**Listeners** ou **Adapters**)
 - Qualquer objecto pode ser notificado de um evento
- Os métodos dos ouvintes (listeners) que desejam tratar eventos, recebem eventos como argumento

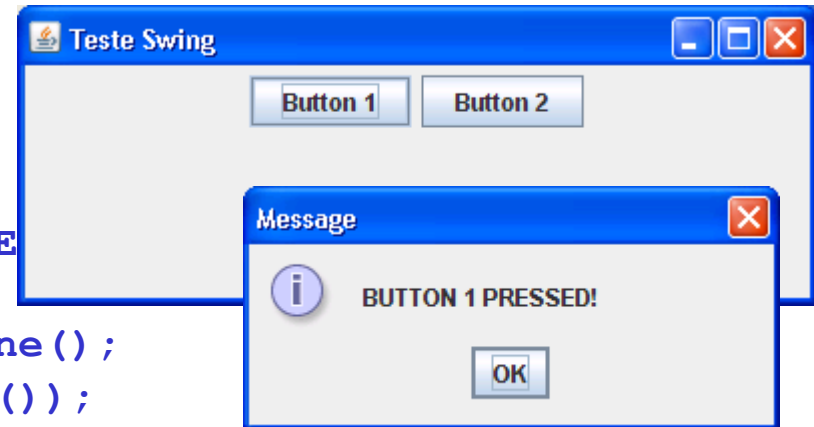
```
public void eventoOcorreu(EventObject evento) {  
    System.out.println(evento+ " em " +evento.getSource());  
}
```
- Os ouvintes precisam ser registrados nas fontes
 - Quando ocorre um evento, um método de todos os ouvintes registrados é chamado e o evento é passado como argumento



Exemplo

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

```
public class Win6 extends JFrame {  
    public Win6() {  
        super("Teste Swing");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setSize(400, 150);  
        Container content = getContentPane();  
        content.setLayout(new FlowLayout());  
        JButton b1 = new JButton("Button 1");  
        JButton b2 = new JButton("Button 2");  
        content.add(b1);  
        content.add(b2);  
        b1.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                JOptionPane.showMessageDialog(getContentPane(),  
                    "BUTTON 1 PRESSED!");  
            }  
        });  
        setVisible(true);  
    }  
    public static void main(String[] args) {  
        new Win6();  
    }  
}
```



Tipos de Eventos

- `java.awt.event.*`

- `ActionEvent`
- `AdjustmentEvent`
- `ComponentEvent`
- `ContainerEvent`
- `FocusEvent`
- `InputEvent`
- `InputMethodEvent`
- `InvocationEvent`
- `ItemEvent`
- `KeyEvent`
- `MouseEvent`
- `MouseWheelEvent`
- `PaintEvent`
- `TextEvent`

Tipos de Receptores (EventListener)

- `java.awt.event.*`

- ActionListener
- ContainerListener
- FocusListener
- InputMethodListener
- ItemListener
- KeyListener
- MouseListener
- MouseMotionListener
- MouseWheelListener
- TextListener
- WindowFocusListener
- WindowListener

Geralmente uma interface **EventListener** (+ do que um método) tem associada uma classe **Adapter** que fornece uma implementação vazia ({}) dos seus métodos

Tipos de Receptores (EventListener)

Interface Listener	Classe Adapter	Métodos
ActionListener	-----	actionPerformed(ActionEvent)
AdjustmentListener	-----	adjustmentValueChanged(adjustmentEvent)
ComponentListener	ComponentAdapter	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
ContainerListener	ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
FocusListener	FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)
ItemListener	-----	itemStateChanged(ItemEvent)
KeyListener	KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener	MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)

Tipos de Receptores (EventListener)

Interface Listener	Classe Adapter	Métodos
MouseMotionListener	MouseMotionAdapter	mouseDragged (MouseEvent) mouseMoved(MouseEvent)
TextListener	-----	textValueChanged(TextEvent)
WindowListener	WindowAdapter	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)

Formas alternativas de implementação

1. A **própria classe** do objecto que gera o evento implementa a interface *Listener* ou estende a classe *Adapter* do evento.

```
public class MinhaClasse implements WindowListener {...}
      ou
public class MinhaClasse extends WindowAdapter {...}
```

2. Construir uma **classe externa** que implemente a interface ou estenda a classe *Adapter* do evento

```
public class MinhaClasse { ... }
public class EventoExt implements WindowListener {...}
      ou
public class EventoExt extends WindowAdapter {...}
```

Formas alternativas de implementação

3. Construir uma **classe membro interna** que implemente a interface ou estenda a classe *Adapter* respectiva

```
public class MinhaClasse { ...  
    class EventoInt implements WindowListener {...}  
    ou  
    class EventoInt extends WindowAdapter {...}  
}
```

4. Construir uma **classe anônima interna** que implemente a interface ou a classe *Adapter*

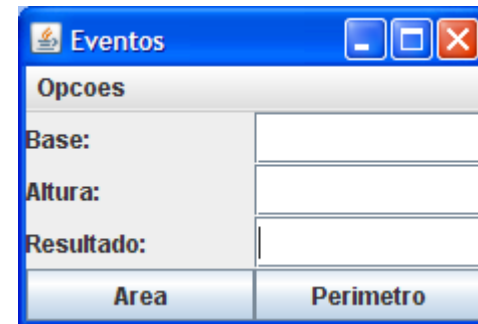
```
public class MinhaClasse { ...  
    janela.addWindowListener( new WindowAdapter( ) {  
        public void windowClosing(WindowEvent e) {  
            System.exit(0);  
        }  
    });  
}
```

Como escrever um ActionListener

- São os eventos mais fáceis e os mais usados
- Implementa-se um ActionListener para responder a uma intervenção do utilizador
 - premir de um botão
 - duplo click numa lista
 - escolha de um menu
 - retorno num campo de texto
- O resultado é o envio de uma mensagem “actionPerformed” a todos os ActionListeners que estão registrados no componente fonte.

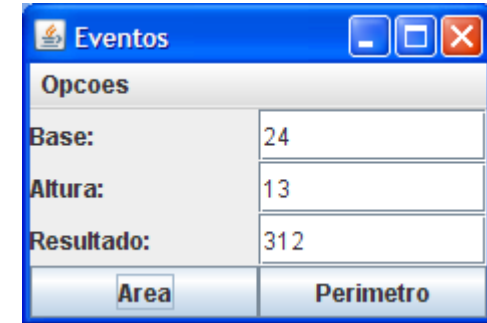
ActionListener - Exemplo

```
public class ActionListenerDemo extends JFrame {  
    private JMenuBar jmbBarraMenu;  
    private JMenu jmopcoes;  
    private JMenuItem jmiSair;  
    private JButton bArea, bPerimetro;  
    private JPanel p1;  
    private JTextField jtfBase, jtfAltura, jtfResultado;  
    private JLabel jlBase, jlAltura, jlResultado;  
  
    public ActionListenerDemo() {  
        jmbBarraMenu=new JMenuBar();  
        jmopcoes=new JMenu("Opcoes");  
        jmiSair=new JMenuItem("Sair");  
        jmopcoes.add(jmiSair);  
        jmbBarraMenu.add(jmopcoes);  
        setJMenuBar(jmbBarraMenu);  
  
        p1 = new JPanel(new GridLayout(4,2));  
        jlBase = new JLabel("Base: ");  
        jlAltura = new JLabel("Altura: ");  
        jlResultado = new JLabel("Resultado: ");  
        jtfBase = new JTextField(10);  
        jtfAltura = new JTextField(10);  
        jtfResultado = new JTextField(10);  
        bArea = new JButton("Area");  
        bPerimetro = new JButton("Perimetro");  
        p1.add(jlBase); p1.add(jtfBase); p1.add(jlAltura); p1.add(jtfAltura);  
        p1.add(jlResultado); p1.add(jtfResultado); p1.add(bArea); p1.add(bPerimetro);  
        getContentPane().add(p1);  
    }  
}
```



ActionListener - Exemplo

```
jtfBase.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        jtfBase.transferFocus();  
    }  
});  
bArea.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        jtfResultado.setText(String.valueOf(Integer.parseInt(jtfBase  
            .getText())  
            * Integer.parseInt(jtfAltura.getText())));  
    }  
});  
bPerimetro.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        jtfResultado.setText(String.valueOf(2  
            * (Integer.parseInt(jtfBase.getText()) + 2  
            * (Integer.parseInt(jtfAltura.getText()))));  
    }  
});  
jmiSair.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        System.exit(0);  
    }  
});  
}
```



Opcoes	
Base:	24
Altura:	13
Resultado:	312
<div>Area Perimetro</div>	

Sumário

- Java Foundation Classes é um conjunto muito extenso de classes!
- Não é fácil dominar todas as funcionalidades
- Manter em mente a metáfora de construção de interfaces
 - Containers, componentes e gestão de eventos
- Usar IDE ajuda!
 - NetBeans,...