



# Programação II

Interface Gráfica

<http://dl.dropbox.com/u/3025380/prog2/aula12.pdf>

[flavio.cec@unisul.br](mailto:flavio.cec@unisul.br)

# Interface Gráfica

- Todo tipo de software que é construído, independentemente de sua finalidade, possui algum grau de interface com o usuário.

```
***** CUNIT CONSOLE - MAIN MENU *****
(R)un all, (S)elect suite, (L)ist suites, Show (F)ailures, (Q)uit
Enter Command : r

Running Suite : Suite_success
Running test : successful_test_1
Running test : successful_test_2
Running test : successful_test_3
WARNING - Suite initialization failed for Suite_init_failure.
Running Suite : Suite_clean_failure
Running test : successful_test_4
Running test : failed_test_2
Running test : successful_test_1
WARNING - Suite cleanup failed for Suite_clean_failure.
Running Suite : Suite_mixed
Running test : successful_test_2
Running test : failed_test_4
Running test : failed_test_2
Running test : successful_test_4

--Run Summary: Type      Total   Ran   Passed  Failed
suitses             4         3     n/a     2
tests              13        10      7      3
asserts            10        10      7      3

***** CUNIT CONSOLE - MAIN MENU *****
(R)un all, (S)elect suite, (L)ist suites, Show (F)ailures, (Q)uit
Enter Command :
```



**MaintainJ - Start and Stop Tracing**

**Start Tracing:** Click Start Tracing button to turn on tracing. After tracing is turned on, call trace will be logged for any action performed in the application.

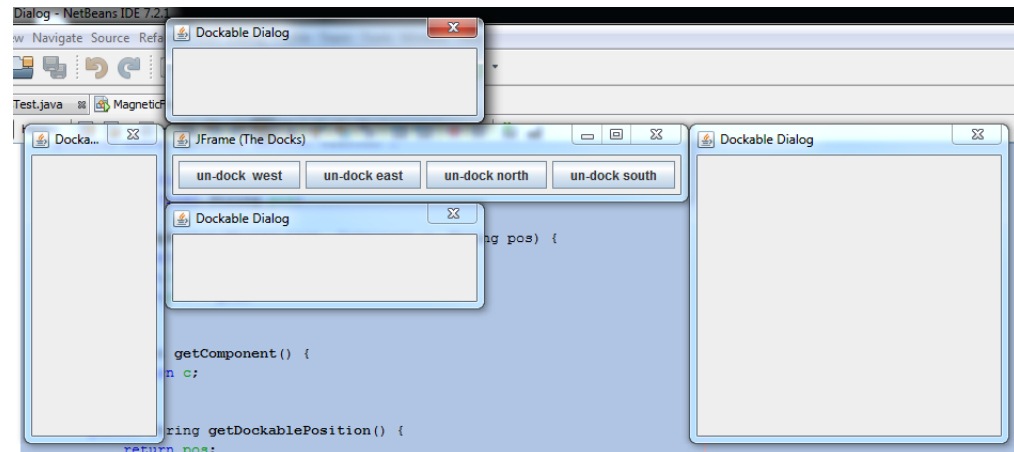
Trace File:

The following trace files are generated. Copy them into any project in the Eclipse workspace and open to view the UML diagrams.

Trace File Name	Number of Calls
C:\Eclipse\w3.3.2\workspace\workspace\TraceFiles\InitialTrace_AllThreads.ser	2965
C:\Eclipse\w3.3.2\workspace\workspace\TraceFiles\InitialTrace_main.ser	1515
C:\Eclipse\w3.3.2\workspace\workspace\TraceFiles\InitialTrace_AWT-Event Queue-0.ser	1450

# Interface Gráfica

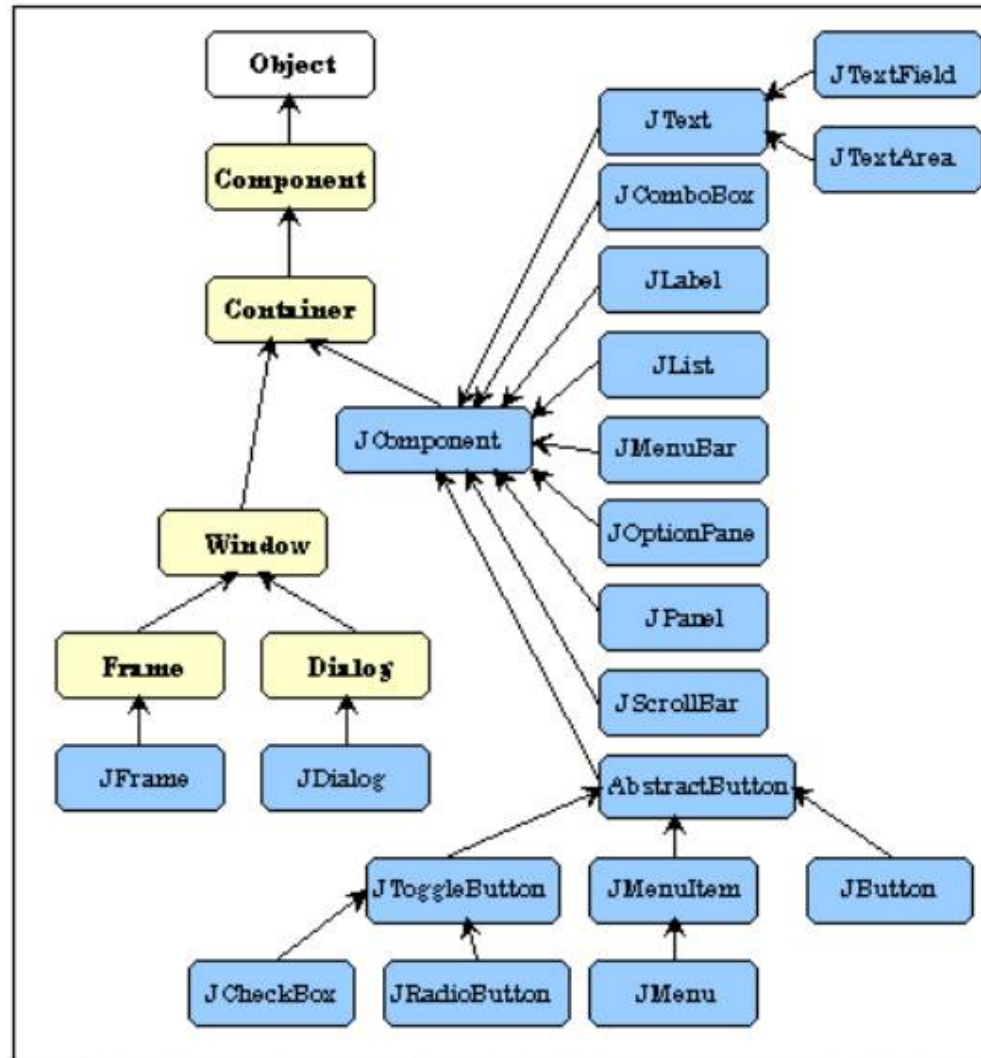
- Normalmente quando o termo “interface gráfica” é usado, ele significa a possibilidade do software ter alguma forma de interação visual, que permita ao usuário do aplicativo não só visualizar componentes na tela, como também efetuar ações sobre estes componentes, como digitar, arrastar, selecionar, clicar, entre outros.



# Interface Gráfica em Java

- O Java possui dois toolkits para trabalhar com interface gráfica:
  - AWT: Pacote mais limitado e antigo do java;
  - Swing: Pacote que traz mais componentes e recursos para interfaces desktop.

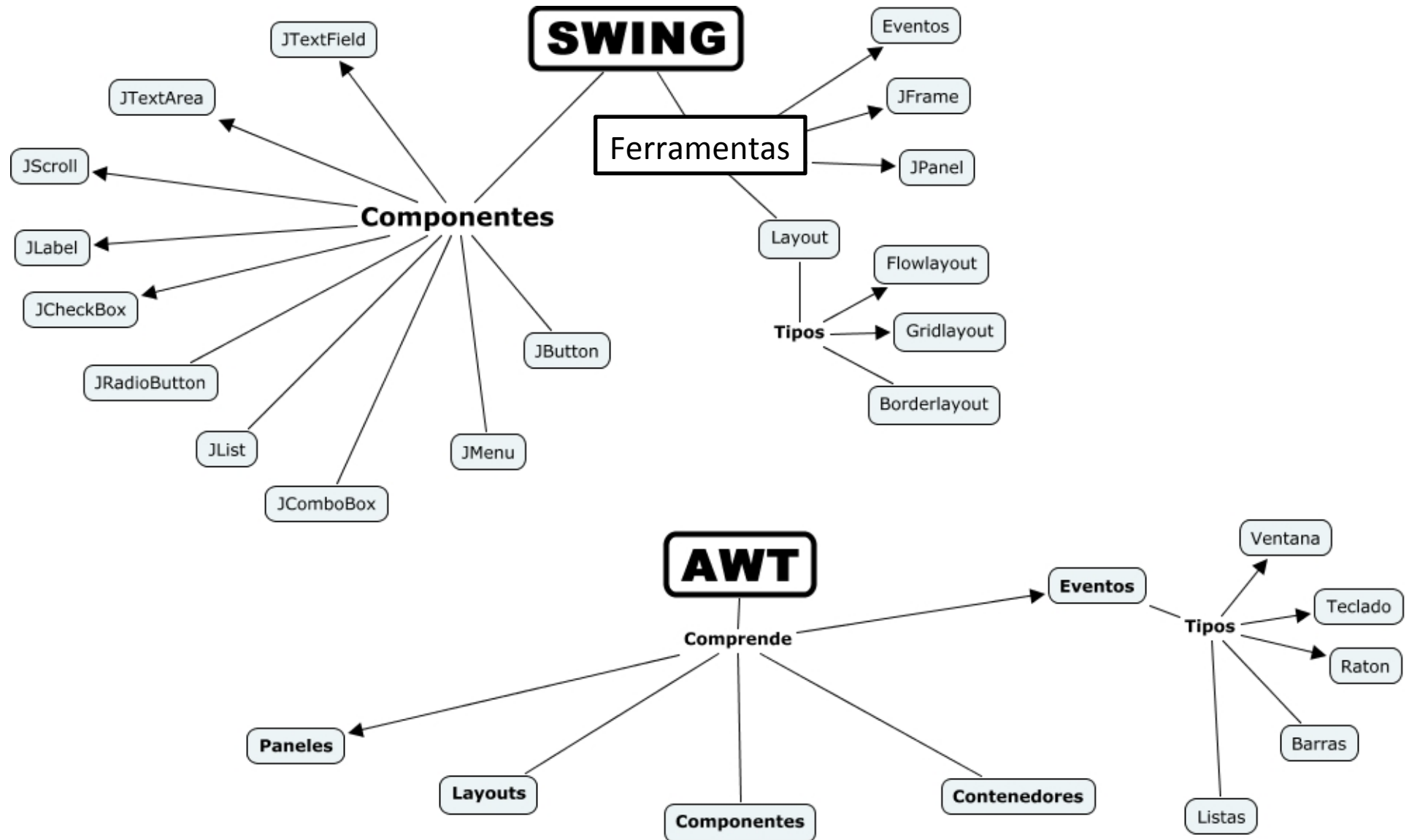
# AWT e Java Swing



This diagram shows a subset of the Swing components (in blue) and how they extend the AWT (in yellow) components.

AWT em amarelo  
Swing em azul

# AWT e Java Swing



# Componentes

- **JFrame**: usado para construir uma janela com recursos, tais como, borda, barra de título e botões para fechar, minimizar e maximizar.
- **JLabel** : usado para escrever texto na tela.
- **TextField** : usado para construir um campo de edição, onde o usuário pode digitar qualquer tipo de texto.
- **Button** : usado para construir um botão na tela, permitindo que o usuário clique sobre ele.

# Componentes

- **JCheckBox** : componente que permite apresentar um série de opções que podem ser selecionadas e deselecionadas pelo usuário.
- **JList**: componente que exibe uma lista de itens em uma caixa.
- **JComboBox** : componente que exibe uma lista de opções em uma caixa e permite que o usuário selecione uma opção.
- **JDialog** : componente que permite construir janelas de dialogo.



# Componentes

- **FileChooser** : uma janela que permite ao usuário selecionar um arquivo.
- **JTable** : componente que permite mostrar informações em um formato tabular, como se fosse uma matriz. Esta matriz é composta por linhas e colunas, sendo que as colunas podem ter títulos.

# Componentes

- **JMenuBar** : usado para construção de um menu de opções. Deve ser usado em conjunto com componentes JMenu. Um menu (JMenuBar) é composto por vários JMenu's, colocados juntos.
  - Por exemplo: no software MS-Word, o menu onde aparecem as opções “Arquivo”, “Editar”, “Exibir”, “Inserir”, “Formatar”, etc, correspondem aos JMenu`s, ou seja, cada uma destas opções é um JMenu. Todas estas opções deve estar juntas dentro de um único componente, o JMenuBar.

# Componentes

- **JMenu** : usado para criar os menus que são adicionados ao JMenuBar. Para que um menu seja construído é necessário adicionar componentes JMenuItem, que representam os itens de um menu.
  - Por exemplo: usando o software MS-Word, a opção de menu “Arquivo”, possui diversos itens, como “Novo”, “Abrir”, “Fechar”, “Salvar”, “Salvar como”, etc. Cada um destes itens corresponde a um JMenuItem, que foram adicionados ao JMenu “Arquivo”.

# Componentes

- **JMenuItem** : componente usado para representar um ítem adicionado a um JMenu.

Vamos à pratica...



# Criando uma janela

- Para termos uma janela é necessário que a classe estenda (***extends***) um ***JFrame***.

```
public class TelaInicial extends JFrame {  
    private static final long serialVersionUID = -2037259359850427040L;
```

# Criando uma janela

```
import java.awt.FlowLayout;

import javax.swing.JFrame;
import javax.swing.JLabel;

public class TelaInicial extends JFrame {

    private static final long serialVersionUID = -2037259359850427040L;

    private JLabel label1;

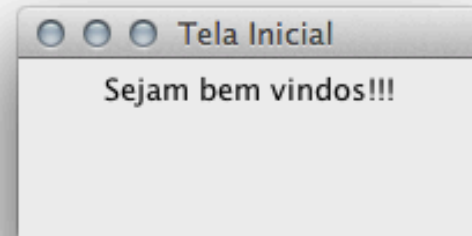
    public TelaInicial() {
        super("Tela Inicial");

        this.configuraTela();
    }

    private void configuraTela() {
        this.setLayout(new FlowLayout()); //seta o gerenciador de layout dos componentes
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(200,100);

        label1 = new JLabel("Sejam bem vindos!!!");
        label1.setToolTipText("Este é o rotulo 1");
        add(label1); //adiciona o componente label1 ao JFrame (janela)
    }

    public static void main(String[] args) {
        TelaInicial tela = new TelaInicial();
        tela.setVisible(true);
    }
}
```



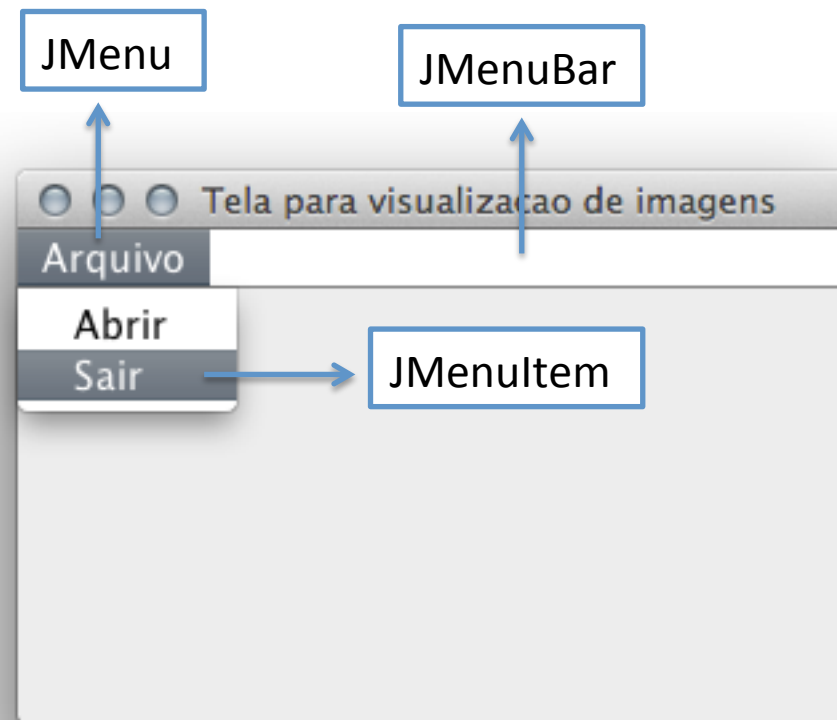
Tela de visualização de imagens



# Tela de visualização de imagens

## Criando a janela:

```
public class TelaVerImagem extends JFrame {  
    private static final long serialVersionUID = 1L;  
  
    public TelaVerImagem() {  
        super("Tela para visualizacao de imagens");  
  
        this.configuraTela();  
  
        this.preparandoMenu();  
    }  
  
    private void configuraTela() {  
        this.setSize(800, 600);  
  
        //Finaliza a aplicacao caso essa janela seja fechada.  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
  
    private void preparandoMenu() {  
        JMenuBar barraMenu = new JMenuBar();  
        this.setJMenuBar(barraMenu);  
  
        JMenu menuArquivo = new JMenu("Arquivo");  
        barraMenu.add(menuArquivo);  
  
        JMenuItem itemAbrir = new JMenuItem("Abrir");  
        menuArquivo.add(itemAbrir);  
  
        JMenuItem itemSair = new JMenuItem("Sair");  
        menuArquivo.add(itemSair);  
    }  
  
    public static void main(String[] args) {  
        TelaVerImagem tela = new TelaVerImagem();  
        tela.setVisible(true);  
    }  
}
```



# Tela de visualização de imagens

- Tratamento de eventos:
  - O swing trabalha com um modelo baseado em “listener”, que ficam aguardando algum estímulo (clique no mouse, movimento...) para gerar uma ação.
  - **ActionEvents**: objeto gerado a partir de um estímulo em um elemento (clique item menu);
  - **ActionListener**: os objetos que querem “escutar” um evento como o de cima necessitam implementar esta interface.

# Tela de visualização de imagens

- Tratamento de eventos:

```
private void preparandoMenu() {  
    JMenuBar barraMenu = new JMenuBar();  
    this.setJMenuBar(barraMenu);  
  
    JMenu menuArquivo = new JMenu("Arquivo");  
    barraMenu.add(menuArquivo);  
  
    JMenuItem itemAbrir = new JMenuItem("Abrir");  
    itemAbrir.addActionListener(new ListenerTela());  
    menuArquivo.add(itemAbrir);  
  
    JMenuItem itemSair = new JMenuItem("Sair");  
    itemSair.addActionListener(new ListenerTela());  
    menuArquivo.add(itemSair);  
}  
  
private class ListenerTela implements ActionListener {  
  
    @Override  
    public void actionPerformed(ActionEvent evento) {  
        String acao = evento.getActionCommand();  
  
        if("Abrir".equals(acao)) {  
            JOptionPane.showMessageDialog(null, "Abrir evento");  
        } else if("Sair".equals(acao)) {  
            fecharTela();  
        }  
    }  
}  
  
private void fecharTela() {  
    this.setVisible(false);  
    System.exit(0);  
}
```

A classe que trata o evento e implementa **ActionListener**, pode ser implementada como uma InnerClass (classe interna)



# Tela de visualização de imagens

```
private static final long serialVersionUID = 1L;

private JLabel figura;

private ImageIcon imagem;

public TelaVerImagem() {
    super("Tela para visualizacao de imagens");

    this.configuraTela();

    this.preparandoMenu();
}

private void configuraTela() {
    this.setLayout(new FlowLayout()); //seta o gerenciador de layout
    this.setSize(800, 600);

    //Finaliza a aplicação caso essa janela seja fechada.
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    this.carregaImagem(null);
    this.add(figura);
}

private void preparandoMenu() {
    JMenuBar barraMenu = new JMenuBar();
    this.setJMenuBar(barraMenu);

    JMenu menuArquivo = new JMenu("Arquivo");
    barraMenu.add(menuArquivo);

    JMenuItem itemAbrir = new JMenuItem("Abrir");
    itemAbrir.addActionListener(new ListenerTela());
    menuArquivo.add(itemAbrir);

    JMenuItem itemSair = new JMenuItem("Sair");
    itemSair.addActionListener(new ListenerTela());
    menuArquivo.add(itemSair);
}
```

```
private void carregaImagem(String caminho) {
    if(caminho == null || caminho.isEmpty()) {
        caminho = "conteudo/figura1.jpg";

        imagem = new ImageIcon(caminho);
        figura = new JLabel(imagem);
        this.add(figura);
    }

    File arquivo = new File(caminho);

    if(arquivo.exists()) {
        imagem = new ImageIcon(caminho);
        figura.setIcon(imagem);
        figura.repaint();
    } else {
        JOptionPane.showMessageDialog(this, "O caminho é inválido");
    }
}

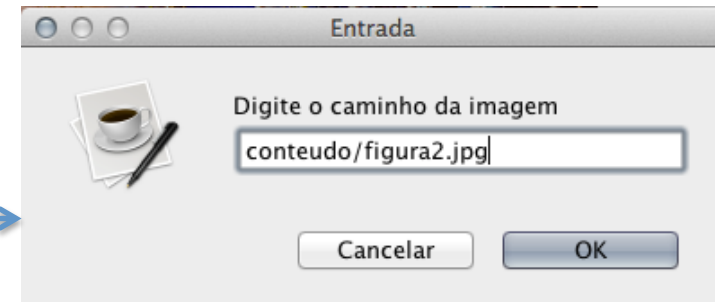
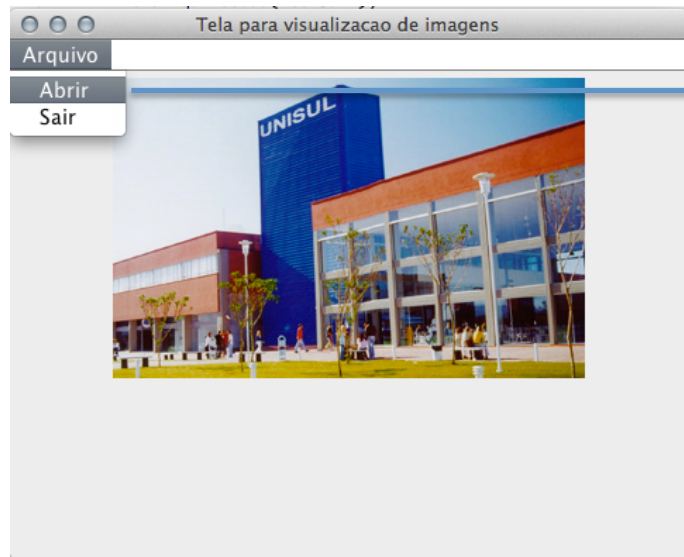
private class ListenerTela implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent evento) {
        String acao = evento.getActionCommand();

        if("Abrir".equals(acao)) {
            String caminho = JOptionPane.showInputDialog("Digite o " +
                "caminho da imagem");
            carregaImagem(caminho);
        } else if("Sair".equals(acao)) {
            fecharTela();
        }
    }
}

private void fecharTela() {
    this.setVisible(false);
    System.exit(0);
}
```

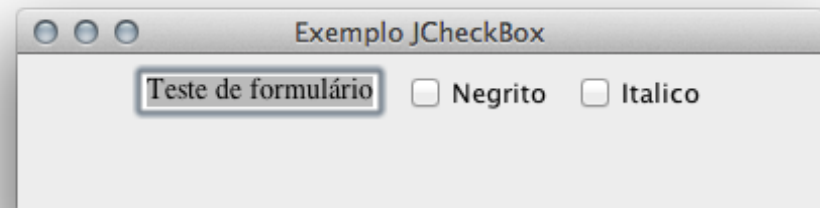
# Tela de visualização de imagens



# Formulários

# Tela de formulário inicial

```
public class TelaFormulario extends JFrame {  
  
    private static final long serialVersionUID = 1L;  
    private JTextField textfield;  
    private JCheckBox checkNegrito;  
    private JCheckBox checkItalico;  
  
    public TelaFormulario(){  
        super("Exemplo JCheckBox");  
  
        this.setLayout(new FlowLayout()); //configura o l  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.setSize(400,100);  
  
        this.configuraTela();  
    }  
  
    private void configuraTela() {  
        textfield = new JTextField("Teste de formulário");  
        textfield.setFont(new Font("Serif", Font.PLAIN, 14));  
        checkNegrito = new JCheckBox("Negrito");//cria o primeiro componente  
        checkItalico = new JCheckBox("Italico");//cria o segundo componente  
  
        this.add(textfield); //adiciona textfield ao frame  
        this.add(checkNegrito); //adiciona os dois componentes ao frame  
        this.add(checkItalico); //cria o objeto ouvinte de evento  
  
        TrataEventoCheckBox trataevento = new TrataEventoCheckBox(); //adiciona esse  
        //objeto a cada um dos componentes. Veja que o método agora é addItemListener()  
        checkNegrito.addItemListener(trataevento);  
        checkItalico.addItemListener(trataevento);  
    }  
}
```

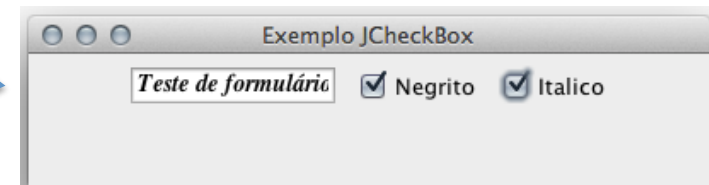
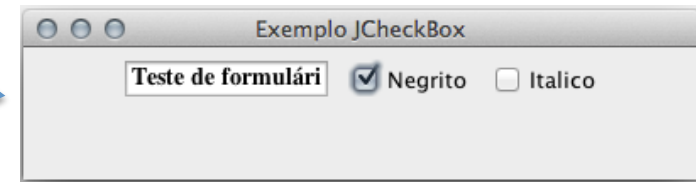




# Tela de formulário inicial

```
private class TrataEventoCheckBox implements ItemListener{
    private int valorNegrito = Font.PLAIN; //controla o estilo da fonte Negrito private
    private int valorItalico = Font.PLAIN; //controla o estilo da fonte Itálico
    public void itemStateChanged(ItemEvent event){
        //verifica a origem do evento. Trata o evento para o componente checkNegrito
        if (event.getSource() == checkNegrito){
            if (checkNegrito.isSelected()) {
                valorNegrito = Font.BOLD;
            } else {
                valorNegrito = Font.PLAIN;
            }
        }
        //verifica a origem do evento. Trata o evento para o componente checkItalico
        if (event.getSource() == checkItalico){
            if (checkItalico.isSelected()) {
                valorItalico = Font.ITALIC;
            } else {
                valorItalico = Font.PLAIN;
            }
        }
        textfield.setFont(new Font("Serif", valorNegrito + valorItalico, 14));
    }
}

public static void main(String[] args) {
    new TelaFormulario().setVisible(true);
}
```





Tela de formulário para soma

# Tela de formulário para soma

```
public class TelaFormularioSoma extends JFrame {
    private static final long serialVersionUID = 1L;
    private JLabel rotuloValor1;
    private JLabel rotuloValor2;
    private JTextField campoValor1;
    private JTextField campoValor2;
    private JTextField campoResultado;
    private JLabel rotuloResultado;
    private JButton botaoSoma;
    private JButton botaoLimpar;

    public TelaFormularioSoma(){
        super("Exemplo JTextField");

        this.setLayout(new FlowLayout());
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(275,180);

        this.configuraTela();
    }

    private class TrataEventoBotao implements ActionListener{
        public void actionPerformed(ActionEvent event){
            campoResultado.setEditable(true);

            if(campoValor1.getText() != null && campoValor2.getText() != null
                && !campoValor1.getText().isEmpty()
                && !campoValor2.getText().isEmpty()) {

                int valor1 = Integer.parseInt(campoValor1.getText());
                int valor2 = Integer.parseInt(campoValor2.getText());

                campoResultado.setText(String.valueOf(valor1+valor2));
            } else {
                JOptionPane.showMessageDialog(null, "Não são permitidos valores nulos");
            }
        }
    }
}
```

```
private void configuraTela() {
    rotuloValor1 = new JLabel("Primeiro Número");
    rotuloValor1.setToolTipText("Digite o primeiro número");
    this.add(rotuloValor1);

    campoValor1 = new JTextField(10);
    this.add(campoValor1);

    rotuloValor2 = new JLabel("Segundo Número");
    rotuloValor2.setToolTipText("Digite o segundo número");
    this.add(rotuloValor2);

    campoValor2 = new JTextField(10);
    this.add(campoValor2);

    botaoSoma = new JButton("Soma");

    TrataEventoBotao trataevento = new TrataEventoBotao();
    botaoSoma.addActionListener(trataevento);
    rotuloResultado = new JLabel("Resultado");
    rotuloResultado.setToolTipText("Resultado da Soma");
    this.add(rotuloResultado);

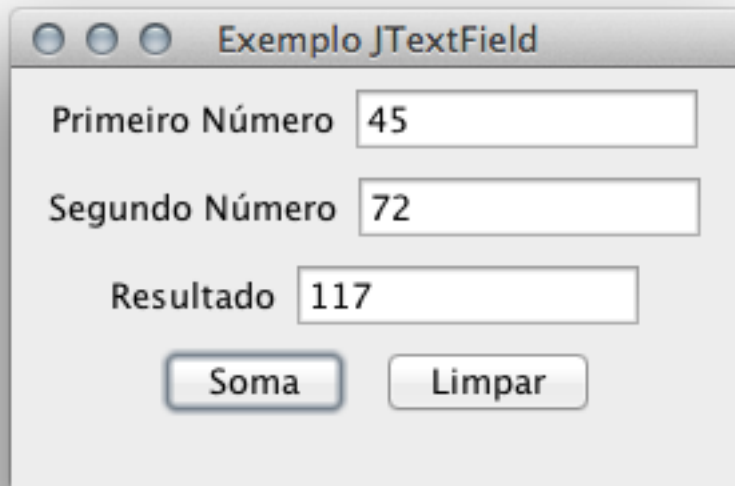
    campoResultado = new JTextField(10);
    campoResultado.setEditable(false);
    this.add(campoResultado);

    this.add(botaoSoma);

    botaoLimpar = new JButton("Limpar");
    botaoLimpar.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent arg0) {
            campoValor1.setText("");
            campoValor2.setText("");
            campoResultado.setText("");
        }
    });
    this.add(botaoLimpar);
}
```

# Tela de formulário para soma



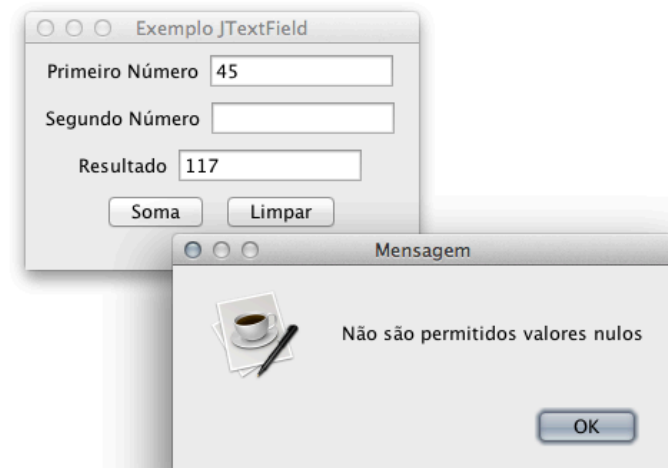
Exemplo JTextField

Primeiro Número 45

Segundo Número 72

Resultado 117

Soma Limpar



Exemplo JTextField

Primeiro Número 45

Segundo Número

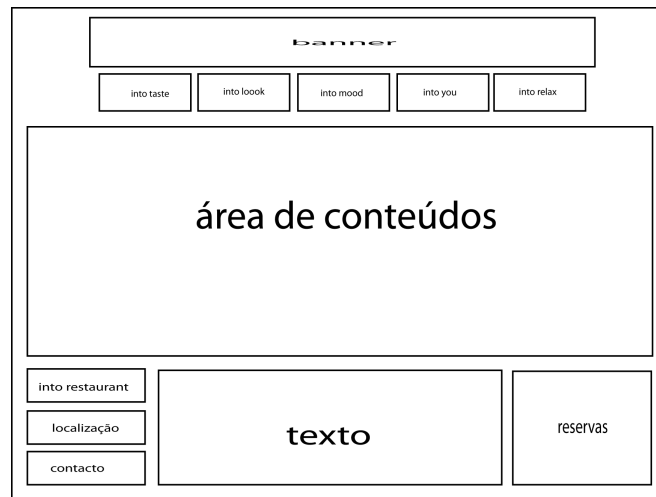
Resultado 117

Soma Limpar

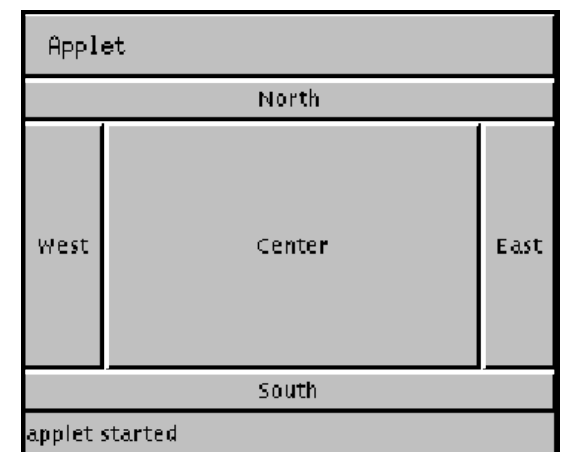
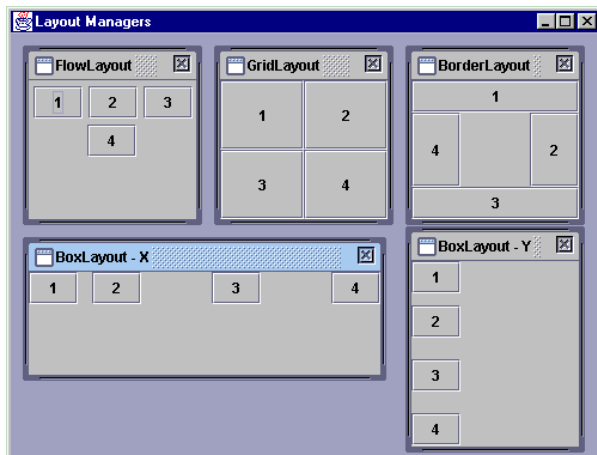
Mensagem

Não são permitidos valores nulos

OK



# Layout



# Layout

- Forma de organizar os componentes na tela;
- Todos os gerenciadores de layout implementam a interface `LayoutManager` que faz parte do pacote `java.awt`.
- Existe o método `setLayout` da classe `Container` que aceita um objeto que implementa a interface `LayoutManager` como um argumento.

# Layout

- Veja três maneiras básicas de organizar componentes em uma GUI (*Graphical user interface*):
  - Posicionamento absoluto;
  - Gerenciadores de layout; e
  - Programação visual em uma IDE.

# Layout

- Posicionamento absoluto:
  - fornece o maior nível de controle sobre a aparência de uma GUI, pois configura o layout de um ***Container*** como ***null*** podendo especificar a posição absoluta de cada componente GUI em relação ao canto superior esquerdo do Container usando os métodos ***Component setSize*** e ***setLocation*** ou ***setBounds***.

# Layout

- Gerenciadores de layout:
  - é mais simples e mais rápido para criar uma posição GUI com posicionamento absoluto, mas acaba perdendo controle sobre tamanho e o posicionamento dos componentes GUI.



# Layout

- Programação visual em uma IDE:
  - facilita o desenvolvimento em GUI, pois toda IDE tem uma ferramenta de design que permite arrastar e soltar (*drag and drop*) os componentes para uma área de desenho.
  - Eclipse:
    - <http://www.eclipse.org/windowbuilder/>
  - Netbeans (vem nativo):

# Layout

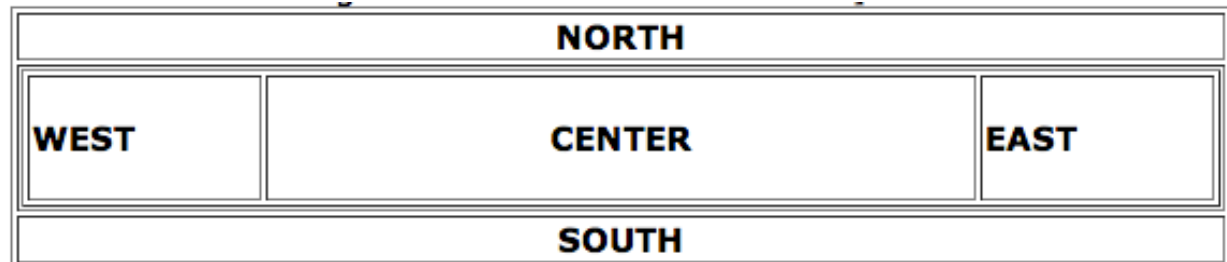
- Cada contêiner individual pode ter apenas um gerenciador de layout, mas vários contêineres no mesmo aplicativo podem utilizar cada um gerenciador de layout.
- Os gerenciadores de layout são diversos, mas vamos focar em três:
  - `FlowLayout`;
  - `BorderLayout`; e
  - `GridLayout`.

# Layout

- **FlowLayout:**
  - Ocorre quando os componentes GUI são colocados em um contêiner da esquerda para a direita na ordem em que são adicionados no contêiner.
  - Quando a borda do contêiner é alcançada, os componentes continuarão a ser exibidos na próxima linha.
  - A classe ***FlowLayout*** permite aos componentes GUI ser alinhados à esquerda, centralizados (padrão) e alinhados à direita.
- Layout utilizado até o momento nos exemplos.

# Layout

- BorderLayout:
  - É um gerenciador de layout que organiza os componentes, sendo a parte superior do contêiner dividida em cinco regiões:
    - NORTH;
    - SOUTH;
    - EAST;
    - WEST; e
    - CENTER.



# BorderLayout

```
public class TelaBorderLayout extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;
    private JButton[] botoes; //ARRAY DE BOTÕES PARA OCULTAR PARTES
    private static final String[] nomes = {"Esconde Norte", "Esconde Sul", "Esconde Leste", "Esconde Oeste", "Esconde Centro"};
    private BorderLayout layout;

    public TelaBorderLayout() {
        super("Tela BorderLayout");
        this.configuraTela();
    }

    private void configuraTela() {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(400,200);

        layout = new BorderLayout(5,5); //ESPAÇOS DE 5 PIXELS
        this.setLayout(layout);
        botoes = new JButton[nomes.length]; //CONFIGURA O TAMANHO DO ARRAY

        //CRIA JBUTTONS E REGISTRA OUVINTES
        for(int count = 0; count < nomes.length; count++) {
            botoes[count] = new JButton(nomes[count]);
            botoes[count].addActionListener(this);
        }

        //ADICIONA A POSIÇÃO DOS BOTÕES
        this.add(botoes[0], BorderLayout.NORTH);
        this.add(botoes[1], BorderLayout.SOUTH);
        this.add(botoes[2], BorderLayout.EAST);
        this.add(botoes[3], BorderLayout.WEST);
        this.add(botoes[4], BorderLayout.CENTER);
    }

    @Override
    public void actionPerformed(ActionEvent evento) {
        //VERIFICA A ORIGEM DE EVENTO E O PAINEL DE CONTEÚDO DE LAYOUT CORRESPONDENTE
        for(JButton botao : botoes) {
            if(evento.getSource() == botao) {
                botao.setVisible(false); //OCULTA O BOTÃO QUANDO CLICADO
            } else {
                botao.setVisible(true);
            }
        }

        layout.layoutContainer(getContentPane());
    }
}
```



# Layout

- GridLayout:
  - É um gerenciador de layout que divide o contêiner em uma grade de modo que os componentes podem ser colocados nas linhas e colunas. A classe **GridLayout** estende a classe **Object** e implementa a interface **LayoutManager**;
  - Cada componente no **GridLayout** tem os mesmos tamanhos, onde podem ser inserida uma célula na parte superior esquerda da grade que prossegue da esquerda para a direita até preencher por completa.

# GridLayout

```
public class TelaGridLayout extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;
    private JButton[] botoes;
    private static final String[] nomes = {"um", "dois", "três", "quatro", "cinco", "seis"};
    private boolean toggle = true;
    private Container container;
    private GridLayout gridLayout1;
    private GridLayout gridLayout2;

    public TelaGridLayout() {
        super("Demo Grid Layout");
        this.configuraTela();
    }

    private void configuraTela() {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300,200);

        gridLayout1 = new GridLayout(2, 3, 5, 5); //2 POR 3; LACUNAS DE 5
        gridLayout2 = new GridLayout(3,2); //3 POR 2; NENHUMA LACUNA
        container = getContentPane(); //OBTÉM O PAINEL DE CONTEÚDO
        this.setLayout(gridLayout1);
        botoes = new JButton[nomes.length];

        for(int count = 0; count < nomes.length; count++) {
            botoes[count] = new JButton(nomes[count]);
            botoes[count].addActionListener(this); //OUVINTE REGISTRADO
            add(botoes[count]);
        }

        //TRATA EVENTOS DE BOTÃO ALTERNANDO ENTRE LAYOUTS
        public void actionPerformed(ActionEvent event) {
            if(toggle)
                container.setLayout(gridLayout2);
            else
                container.setLayout(gridLayout1);

            toggle = !toggle; //ALTERNA PARA O VALOR OPOSTO
            container.validate(); //REFAZ O LAYOUT DO LAYOUT
        }

        public static void main(String[] args) {
            new TelaGridLayout().setVisible(true);
        }
    }
}
```



# Exercício

- Projete e implemente as telas para o sistema da Biblioteca Universitária da Unisul.
- Mais exemplos de implementações e componentes em:

<http://www.java2s.com/Code/Java/Swing-JFC/CatalogSwing-JFC.htm>