

17

Como construir aplicações gráficas e applets

- **AWT** ou **Abstract Window Toolkit** é o antigo conjunto de ferramentas para interfaces gráficas do Java
 - Serve para oferecer infraestrutura mínima de interface gráfica (nível por baixo)
 - Componentes têm aparência dependente de plataforma
 - Limitado em recursos devido a depender de suporte de cada plataforma para os componentes oferecidos
 - Bugs e incompatibilidades entre plataformas
- **JFC** (**Java Foundation Classes**) oferece uma interface muito mais rica
 - **Swing** é o nome dado à coleção de componentes
 - É preciso importar **java.awt** e **javax.swing** para usar JFC

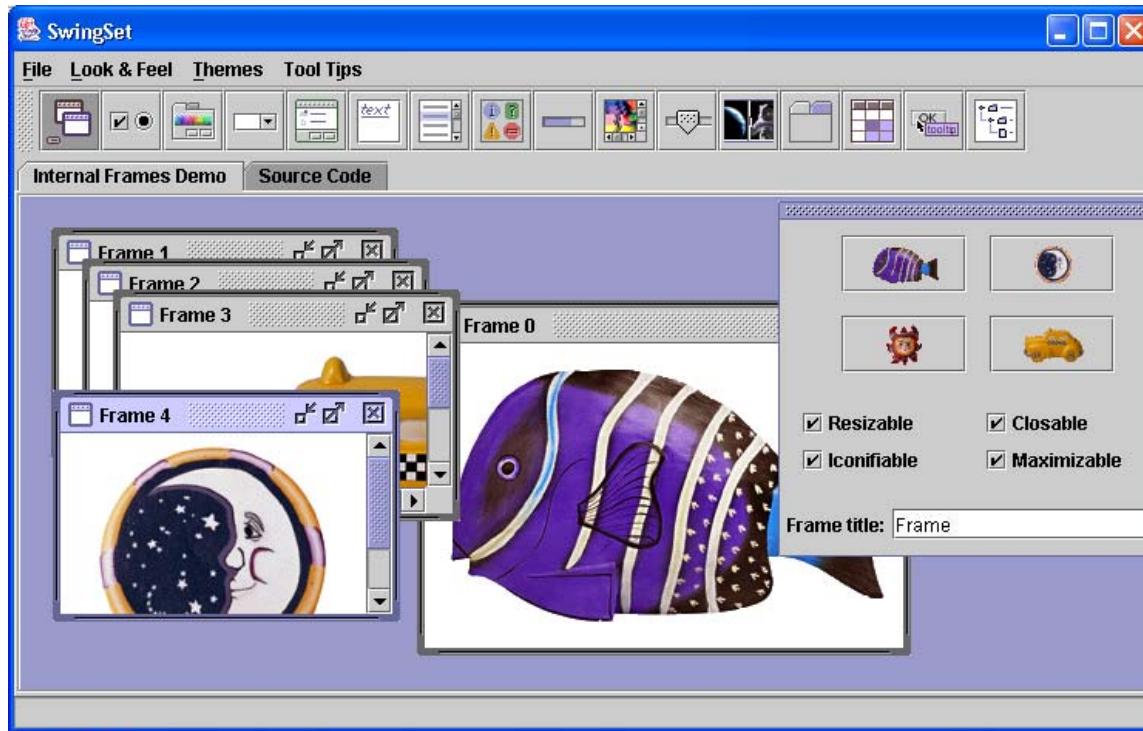
História do AWT

- *Interface gráfica: componentes, layout, eventos*
- *Java 1.0*
 - *Interface que roda de forma medíocre em todas as plataformas (“Abominable” Window Toolkit)*
 - *Modelo de eventos arcaico*
- *Java 1.1*
 - *Melhora do modelo de eventos: por delegação usando design pattern Observer*
- *Java 1.2*
 - *JFC/Swing substitui totalmente componentes AWT*
 - *Mantém e estende a interface de eventos e layout*

Java Foundation Classes

- Parte do J2SE desde Java SDK 1.2. Consiste de:
- **1. Swing**: componentes leves, que não dependem de implementação nativa (veja Java Tutorial)
 - Uma das mais completas bibliotecas gráficas já criadas
 - Baseada em JavaBeans: ferramentas GUI conseguem gerar código legível e reutilizável
- **2. "Look & Feel"**: Drag & drop, cut & paste, undo/redo, il8n, texto estilizado
- Biblioteca de componentes (apenas o Swing) é compatível com JDK 1.1.4
 - Pode ser baixada separadamente e usada com versões limitadas do Java como **J#** da Microsoft e MacOS 9

- Veja demo em `$JAVA_HOME/demo/jfc/SwingSet2/`
 - > `java -jar SwingSet2.jar SwingSet2`



- Como implementar aplicações com Swing?
 - **Java Tutorial:** Swing "trail" possui guias passo-a-passo para uso de cada componente e recurso do JFC e Swing (www.java.sun.com/tutorial)

Tipos de aplicações

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

- Raiz da hierarquia de componentes gráficos
 - Componentes Swing herdam de `javax.swing.JComponent`, que é "neto" de `Component`
- Há um `Component` por trás de tudo que pode ser pintado na tela
- Principais métodos (chamados pelo sistema):
 - `void paint(java.awt.Graphics g)`
 - `void repaint()`
 - `void update(java.awt.Graphics g)`
- O objeto passado como argumento durante a execução (contexto gráfico) é, na verdade, um `java.awt.Graphics2D` (subclasse de `Graphics`)

Componentes AWT

- Há dois tipos importantes de componentes:
- 1) descendentes diretos de **java.awt.Component**
 - "Apenas" **componentes**: descendentes da classe `Component` que não são descendentes de `Container` (todos os componentes da AWT)
- 2) descendentes de **java.awt.Container**
 - Subclasse de `java.awt.Component`
 - São "**recipientes**." Podem **conter** outros componentes.
 - São descendentes da classe `Container`: `Frame`, `Panel`, `Applet` e `JComponent` (raiz da hierarquia dos componentes Swing)

Containers essenciais

- *Frame (AWT) e **JFrame** (Swing)*
 - *Servem de base para qualquer aplicação gráfica*
- *Panel e **JPanel***
 - *Container de propósito geral*
 - *Serve para agrupar outros componentes e permitir layout em camadas*
- *Applet e **JApplet***
 - *Tipo de Panel (JPanel) que serve de base para aplicações que rodam dentro de browsers*
 - *Pode ser inserido dentro de uma página HTML e ocupar o contexto gráfico do browser*

Exemplo de JFrame

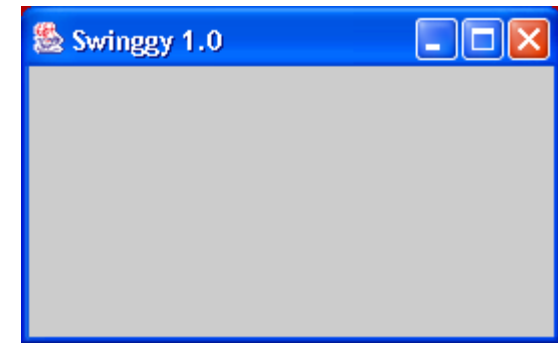
```
import java.awt.*;
import javax.swing.*;

public class Swinggy extends JFrame {

    public Swinggy(String nome) {
        super(nome);

        this.setSize(400,350);
        this.setVisible(true);
    }

    public static void main(String[] args) {
        new Swinggy("Swinggy 1.0");
    }
}
```



- Thread que é responsável pela **atualização** do contexto gráfico
 - Chama **update()** (método de Component) e passa referência para o contexto gráfico como argumento sempre que for necessário redesenhá-lo.
- Método **update(Graphics g)**
 1. Limpa a área a ser redesenhada (contexto gráfico)
 2. Chama **paint(g)**
- Métodos **update()** e **paint()** nunca devem ser chamados diretamente a partir do thread principal
 - Use **repaint()**, que faz o agendamento de uma chamada a **update()** através do AWT thread
 - Sobreponha **update()** se desejar

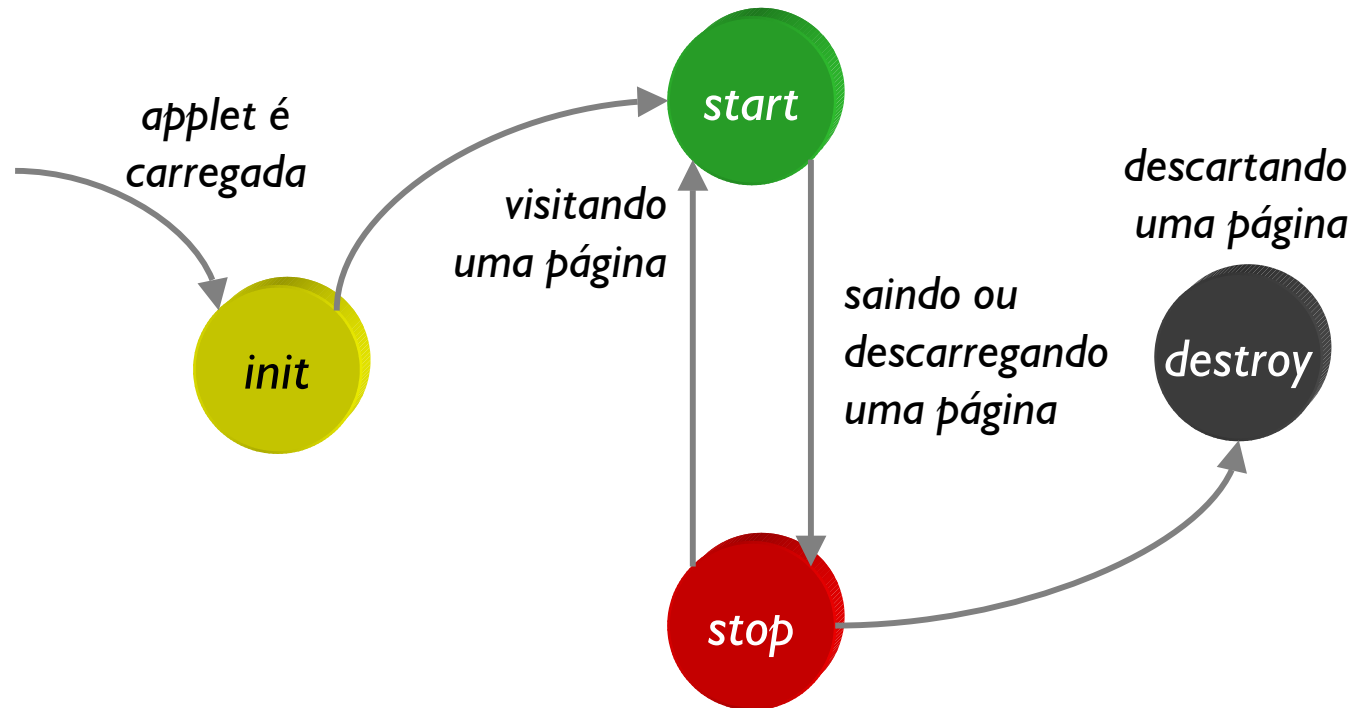
- Representa o **contexto gráfico** de cada componente
- Passado pelo sistema quando chama **update()**
- Programador pode desenhar no componente usando referência recebida via **paint()** ao sobrepor o método:

```
public void paint(Graphics g) {  
    Graphics2D g2 = (Graphics2D) g;  
    Shape s = new Ellipse2D.Double();  
    g2.setColor(Color.red);  
    g2.draw(s);  
}
```

- Para definir o que será desenhado em determinado componente, sobreponha seu método **paint()**
 - Use **Graphics2D!** Mais recursos!

- *Aplicação gráfica que roda em browser*
 - *Toda a infraestrutura herdada da classe `javax.swing.JApplet` (ou `java.applet.Applet`)*
 - *É um componente de um framework que executa em ambiente de execução (container) no browser*
- *Métodos de JApplet, chamados automaticamente, devem ser sobrepostos. Ciclo de vida:*
 - *`init()` - inicialização dos componentes do applet*
 - *`start()` - o que fazer quando applet iniciar*
 - *`stop()` - o que fazer antes de applet parar*
 - *`destroy()` - o que fazer quando applet terminar*
 - *`paint()` - o que desenhar no contexto gráfico*

Ciclo de vida



- **paint()** é outro método que é chamado automaticamente, mas não faz parte do ciclo de vida do Applet, especificamente
 - Faz parte do ciclo de vida de qualquer aplicação gráfica
 - No Applet, é chamado depois do `start()` e sempre que o contexto gráfico do applet for precisar ser atualizado (redimensionamento da janela do browser, ocultação do applet por outra janela, etc.)

Como construir applets

■ Applet mínimo

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

Comentário usado pelo
appletviewer para exibir
Applet:

> appletviewer HelloApplet.java

```
/*  
 * <applet code="HelloApplet" height="50"  
 *     width="200"></applet>  
 */  
public class HelloApplet extends JApplet {  
    public void init() {  
        Container pane = this.getContentPane();  
        JLabel msg = new JLabel("Hello Web");  
        pane.add(msg);  
    }  
}
```

Como usar applets

- Até Java 1.1: para incluir um applet na página Web usava-se

```
<applet code="pacote.Classe"  
        height="100" width="100">  
    <param name="parametro1" value="valor">  
    <param name="parametro2" value="valor">  
</applet>
```

- Java 2 (J2SDK 1.2 em diante) usa HTML 4.0 (que tornou o tag `<applet>` obsoleto)

```
<object classid="XXX-XXX" ...> ... </object>
```

- Para **gerar** `<object>` a partir de `<applet>` use o **HTML Converter**, distribuído com o SDK:

```
java -jar $JAVA_HOME/lib/htmlconverter.jar -gui
```

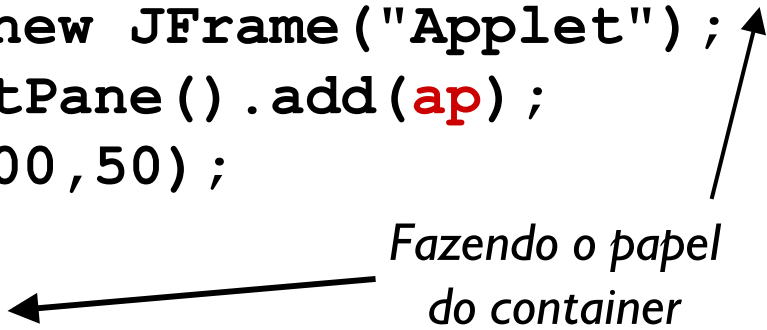

- *JApplets* podem ser incluídos em *JFrames* e *Applets* podem ser incluídos em *Frames*
- Para criar um programa que roda tanto como applet como aplicação
 1. Escreva o applet da forma convencional, implementando *init()*, *start()*, etc. (métodos do framework, que o container usa para controlar o ciclo de vida do applet)
 2. Crie um método *main*, e nele
 - Crie um novo *JFrame* e uma instância do applet
 - Adicione o applet no novo *JFrame*
 - Torne o *JFrame* visível
 - Chame *init()* e *start()* do applet

Exemplo

```
public class HelloApplet extends JApplet {
    public void init() {
        (...)
    }

    public static void main(String[] args) {
        HelloApplet ap = new HelloApplet();
        JFrame f = new JFrame("Applet");
        f.getContentPane().add(ap);
        f.setSize(200,50);
        ap.init();
        ap.start();
        f.setVisible(true);
    }
}
```

Fazendo o papel do container



Restrições dos applets

- Há várias coisas que aplicações comuns podem e que um applet **não** pode fazer:
 - Não pode **carregar bibliotecas** ou definir **métodos nativos**
 - Não pode **ler ou escrever arquivos** na máquina cliente
 - Não pode fazer **conexões de rede** a não ser para a máquina de onde veio
 - Não pode iniciar a **execução** de nenhum programa na máquina do cliente
 - Não tem **acesso** à maior parte das propriedades do sistema
 - Janelas abertas sempre têm aviso de segurança
- Várias restrições podem ser flexibilizadas se o applet for **assinado**.

Applets: vantagens / desvantagens

- **Desvantagens**
 - Restrições
 - Dependência de plug-in e incompatibilidade em browsers
 - Tempo de download
- **Vantagens**
 - Facilidade para realizar comunicação em rede
 - Possibilidade de abrir janelas externas
 - Capacidade de estender o browser em recursos de segurança, protocolos de rede, capacidade gráfica
 - Aplicação sempre atualizada
 - Capacidade de interagir com a página via JavaScript
- **Alternativa (1.4)**
 - **Java Web Start**: aplicações "normais" instaladas via rede

Java Web Start

- *Aplicação distribuída (cliente e servidor) que viabiliza a distribuição e instalação de aplicações via rede*
 - *Aplicação instalada via Web Start é uma aplicação Java "normal" com **possibilidade** de ter acesso irrestrito ao sistema (usuário deve autorizar esse acesso na instalação)*
 - *Checa, sempre que é inicializada, se houve atualização caso a rede esteja disponível*
- *Use o **cliente** Java Web Start para localizar aplicações remotas e instalá-las na sua máquina*
 - *Em máquinas Windows, cliente é instalado junto com J2SDK*
- *Configure seu **servidor Web** para suportar **JNLP** (Java **N**etwork **L**aunch **P**rotocol) e distribuir aplicações via Java Web Start*
 - *Configuração básica consiste da criação de alguns arquivos XML e definição de novo MIME type no servidor*
 - *Veja links para info sobre Java Web Start na documentação do J2SDK*

Recursos gráficos básicos: Fontes e Cores

- Qualquer componente pode mudar a sua fonte e cor
 - A mudança afeta todos os componentes contidos no componente afetado

■ Cores

- instância da classe *java.awt.Color*

```
componente.setBackground(new Color(255,0,0));  
componente.setForeground(Color.yellow);
```

■ Fontes

- instância da classe *java.awt.Font*

```
Font f = new Font("SansSerif", Font.BOLD, 24);  
componente.setFont(f);
```

Há maneiras mais sofisticadas de lidar com fontes (esta é compatível com AWT)

Posicionamento de componentes: Layouts

- Há duas formas de acrescentar componentes em um container
 - Usar um **algoritmo de posicionamento** (layout manager) para dimensionar e posicionar os componentes (esta é a maneira recomendada e default)
 - Desligar o algoritmo de layout e posicionar e redimensionar os componentes diretamente (pixels)
- Todo container tem um algoritmo de layout default
 - Frame e JFrame: **BorderLayout** (layout "geográfico")
 - Outros Containers: **FlowLayout** (layout seqüencial)

- *Para acrescentar objetos em um JFrame ou JApplet, é preciso obter uma interface opaca chamada **ContentPane***
 - *O ContentPane é uma área **independente de plataforma** que cobre a **área útil** do JFrame*
 - *O layout é definido no ContentPane*
 - *Objetos são adicionados no ContentPane*
 - *Cores e fontes devem ser definidas a partir do ContentPane*
- *Para obter o ContentPane use*
`Container pane = frame.getContentPane();`
- *Para definir um layout (diferente de BorderLayout)*
`pane.setLayout(referência_para_layout);`

FlowLayout, JButton, Icon

- *FlowLayout*

- *É o layout mais simples*
- *Dispõe os objetos um depois do outro como se fossem letras digitadas em um editor de texto*
- *Estilo default é centralizado (pode ser alterado)*

```
pane.setLayout(new FlowLayout());
```

- *JButton*

- *Botões simples*
- *Aceitam textos ou imagens (através da interface Icon)*

```
JButton b1 = new Button("texto");
```

```
JButton b2 = new Button("texto", icone);
```

- *Ícones de imagem*

```
Icon icone = new ImageIcon("caminho");
```

- *Caminho deve ser relativo (de preferência) e usar sempre "/" como separador*

Componentes de texto

- *JTextField*
 - campo de entrada de dados simples
- *JPasswordField*
 - campo para entrada de dados ocultos
- *JTextArea*
 - campo de entrada de texto multilinha
- *JEditorPane*
 - editor que entende HTML e RTF
- *JTextPane*
 - editor sofisticado com vários recursos

Principais métodos

- *getText()*: recupera o texto contido no componente
- *setText(valor)*: substitui o texto com outro
- Veja documentação para mais detalhes

Exemplo

```
public class Swinggy2 extends JFrame {  
  
    public Swinggy2(String nome) {  
        super(nome);  
  
        Container ct = this.getContentPane();  
        ct.setLayout(new FlowLayout());  
  
        Icon icone = new ImageIcon("jet.gif");  
        JButton b1 = new JButton("Sair");  
        JButton b2 = new JButton("Viajar", icone);  
  
        ct.add(b1);  
        ct.add(b2);  
  
        this.setSize(400, 350);  
        this.setVisible(true);  
    }  
}
```



- Algoritmos de *layout* podem ser combinados para obter qualquer configuração
 - Mais fáceis de manter e reutilizar
 - *Layout* em camadas e "orientado a objetos"
 - Controlam posicionamento e dimensão de componentes
- Algoritmos de *layout* podem ser criados implementando interface *LayoutManager* (e *LayoutManager2*)
- Para desligar layouts

```
pane.setLayout(null);
```
- Agora é preciso definir posição e tamanho de cada componente

```
componente.setBounds(x, y, larg, alt);
```

Exemplo

```
private JButton b1, b2, b3;
public Swinggy3(String nome) {
    Container ct = this.getContentPane();
    ct.setLayout(null);

    Icon pataverm = new ImageIcon("redpaw.gif");
    Icon pataverd = new ImageIcon("greenpaw.gif");
    Icon pataazul = new ImageIcon("bluepaw.gif");
    b1 = new JButton("Vermelha", pataverm);
    b2 = new JButton("Verde", pataverd);
    b3 = new JButton("Azul", pataazul);
    b1.setBounds(10, 10, 150, 40);
    b2.setBounds(10, 60, 150, 40);
    b3.setBounds(10, 110, 150, 40);
    ct.add(b1);
    ct.add(b2);
    ct.add(b3);
}
```



Aplicação fica mais simples com vetores

```
private JButton[] b;  
private String[] txt = {"Roda", "Para", "Pausa", "Sai",  
                        "Olha", "Urgh!"};  
private String[] img = { "greenpaw.gif", "redpaw.gif",  
                          "bluepaw.gif", "jet.gif",  
                          "eye.gif",      "barata.gif"};  
public Swinggy4(String nome) {  
    Container ct = this.getContentPane();  
    ct.setLayout(null);  
    b = new JButton[txt.length];  
    for (int i = 0; i < b.length; i++) {  
        b[i] = new JButton(txt[i],  
                           new ImageIcon(img[i]));  
        b[i].setBounds(10, 10+(i*50), 150, 40);  
        ct.add(b[i]);  
    } // (...)  
}
```



À medida em que a complexidade desta solução aumentar, você poderá querer encapsular a lógica em um `LayoutManager`!

Outros *Layout Managers*

- ***GridLayout*** (linhas, colunas)
 - *Layout* que posiciona os elementos como elementos de uma tabela
- ***BorderLayout***
 - *Layout* que posiciona elementos em quatro posições "cardeais" e no centro
 - Norte e Sul têm prioridade sobre Leste e Oeste que têm prioridade sobre Centro
 - Constantes `BorderLayout.CENTER`, `BorderLayout.WEST`, `BorderLayout.NORTH`, etc.
- ***BoxLayout*** e ***GridBagLayout***
 - Permitem *layouts* sofisticados com amplo controle

Exemplo com BorderLayout

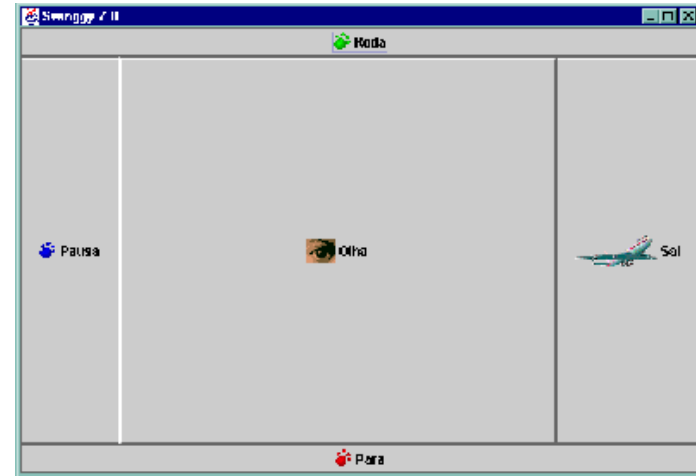
(...)

```
ct.setLayout(new BorderLayout());  
b = new JButton[txt.length];  
String[] pos = {BorderLayout.NORTH,  
                BorderLayout.SOUTH,  
                BorderLayout.WEST,  
                BorderLayout.EAST,  
                BorderLayout.CENTER};
```

```
for (int i=0; i < Math.min(b.length, pos.length); i++) {  
    b[i] = new JButton(txt[i], new ImageIcon(img[i]));  
    ct.add(pos[i], b[i]);  
}
```

(...)

*Veja restante do código
(vetores b, txt e img)
em slides anteriores*



Preferred Size dos componentes

- *Layout Managers são "tiranos"*
 - É **impossível** controlar tamanho e posição de componentes se um *LayoutManager* estiver sob controle
 - Para flexibilizar regras de posicionamento, configure o *Layout Manager* usado através de seus construtores e métodos
 - Para flexibilizar regras de dimensionamento, altere o *preferred size* dos seus componentes
- *Tamanhos preferidos dos componentes*
 - ***setPreferredSize()***, disponível em **alguns** componentes, permite definir o seu tamanho ideal
 - ***getPreferredSize()***, disponível em **todos os componentes** pode ser sobreposto em subclasses e será chamado pelos *Layout Managers* que o respeitam (*Flow*, *Border*)

Regras de BorderLayout

- As áreas de *Border Layout* só aceitam **um componente**
 - Se for necessário ter mais de um componente no *NORTH*, por exemplo, é preciso primeiro adicioná-los dentro de um único componente que será adicionado (um *Panel*, por exemplo)
- Regras de ocupação de espaço
 - *NORTH* e *SOUTH* **têm prioridade** sobre a ocupação da **largura** (usam todo o espaço disponível) mas têm **altura limitada pelo preferred size** do componente
 - *EAST* e *WEST* tem **altura limitada** apenas pela existência ou não de componentes no *NORTH* e/ou *SOUTH* e tem **largura limitada pelo preferred size** do componente
 - *CENTER* **ignora preferred size** e ocupa todo o espaço que puder, mas é limitado pela existência de *NORTH*, *SOUTH*, *EAST* ou *WEST*
- Construtores de *BorderLayout* permitem controle detalhado de espaçamento e outros detalhes

Exemplo com GridLayout

```
(...)  
ct.setLayout(new GridLayout(3, 2));  
  
b = new JButton[txt.length];  
for (int i = 0; i < b.length; i++) {  
    b[i] = new JButton(txt[i],  
                        new ImageIcon(img[i]));  
    ct.add(b[i]);  
}  
(...)
```



*Redimensione a janela
e veja o resultado*

Regras de GridLayout

- *Cada célula aceita um componente*
 - *Se houver mais células que componentes, células não preenchidas ficarão em branco*
 - *Se houver mais componentes que células, estes não serão mostrados*
- *Regras de ocupação de espaço*
 - *Qualquer componente adicionado ocupa toda a célula*
 - *GridLayout ignora preferred size dos componentes: aumento do frame estica todos os componentes*
 - *Para manter o preferred size de um componente, pode-se adicioná-lo em um componente que o respeita (por exemplo, um que use FlowLayout) e adicionar este componente em GridLayout*

Mesmo exemplo com FlowLayout

```
(...)  
public Swinggy5(String nome) {  
    super(nome);  
    Container ct = this.getContentPane();  
    ct.setLayout(new FlowLayout());  
    b = new JButton[txt.length];  
    for (int i = 0; i < b.length; i++) {  
        b[i] = new JButton(txt[i],  
                           new ImageIcon(img[i]));  
        ct.add(b[i]);  
    }  
    this.setSize(400,350);  
    this.setVisible(true);  
} (...)
```

*Redimensione a janela
e veja o resultado*



Regras de FlowLayout

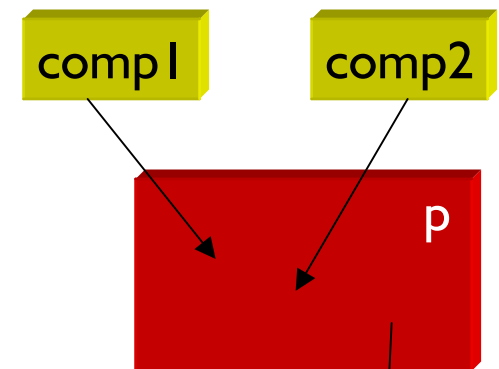
- *Elementos são adicionados em seqüência*
 - *Comportamento default é posicionar elementos lado a lado até que a linha em que estão colocados não mais couber nenhum*
 - *Elementos são centralizados por default*
- *Construtores permitem controle sobre espaçamento e alinhamento dos componentes*
 - *Espaço entre componentes é alterado para todos os componentes*
 - *Alinhamento pode ser pela direita ou pela esquerda, além de default (pelo centro)*
- *Preferred size é respeitado para todos os componentes*

Combinação de Layouts

- Componentes podem ser combinados em recipientes (como `JPanel`) para serem tratados como um conjunto

```
JPanel p = new JPanel();  
p.setLayout(layout do JPanel);  
p.add(comp1);  
p.add(comp2);  
pane.add(BorderLayout.EAST, p);
```

- Possibilita a criação de layouts complexos que consistem de várias camadas
 - Cada `JPanel` é uma camada



Exemplo (1/2)

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

public class Swinggy8 extends JFrame {

    public Swinggy8(String nome) {
        super(nome);

        Container framePane = this.getContentPane();
        framePane.setLayout(new BorderLayout());

        JPanel botoes = new JPanel();
        botoes.setBackground(Color.yellow);
        botoes.setLayout(new GridLayout(3,1));
        botoes.add(new JButton("Um"));
        botoes.add(new JButton("Dois"));
        botoes.add(new JButton("Três"));
        JPanel lateral = new JPanel();
        lateral.add(botoes);
    }
}
```


Exemplo (2/2)

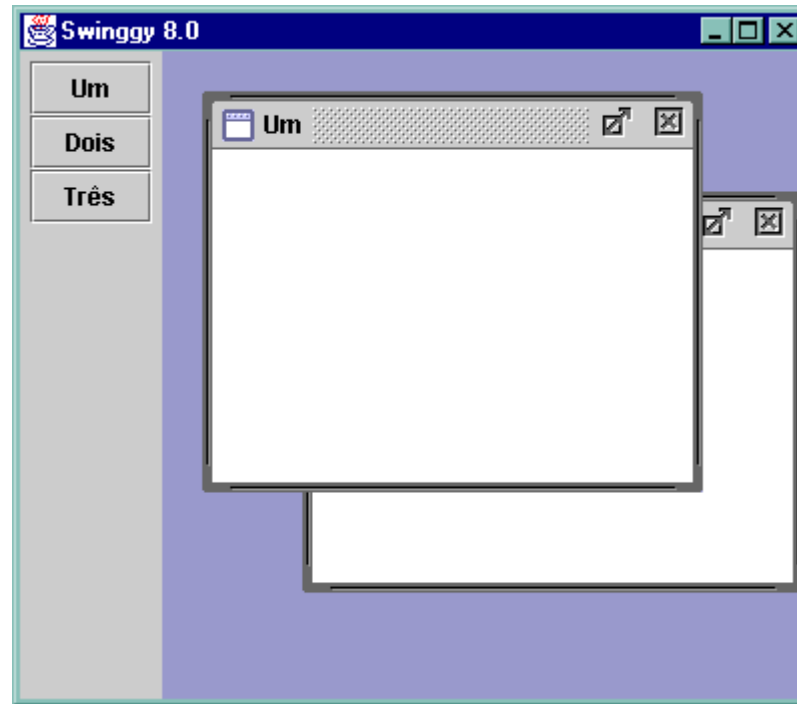
```
JInternalFrame if1 =
    new JInternalFrame("Um", true, true, true);
JInternalFrame if2 =
    new JInternalFrame("Dois", true, true, true);
if1.getContentPane().add(new JEditorPane());
if2.getContentPane().add(new JEditorPane());
if1.setBounds(20,20, 250,200);
if2.setBounds(70,70, 250,200);
if1.setVisible(true);
if2.setVisible(true);
```

```
JDesktopPane dtp = new JDesktopPane();
dtp.add(if1); dtp.add(if2);
framePane.add(BorderLayout.CENTER, dtp);
framePane.add(BorderLayout.WEST, lateral);
this.setSize(400,350);
this.setVisible(true);
```

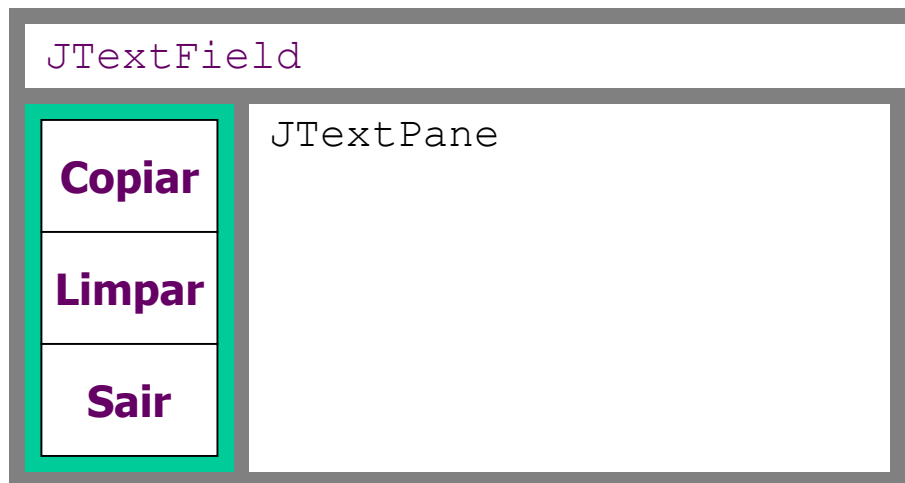
```
}
```

```
}
```

Exemplo: resultado



- 1. Construa uma aplicação gráfica que contenha três botões (JButton), um JTextField e um JTextPane distribuídos da seguinte forma



Use BorderLayout para distribuir os componentes JTextField, JTextPane e JPanel

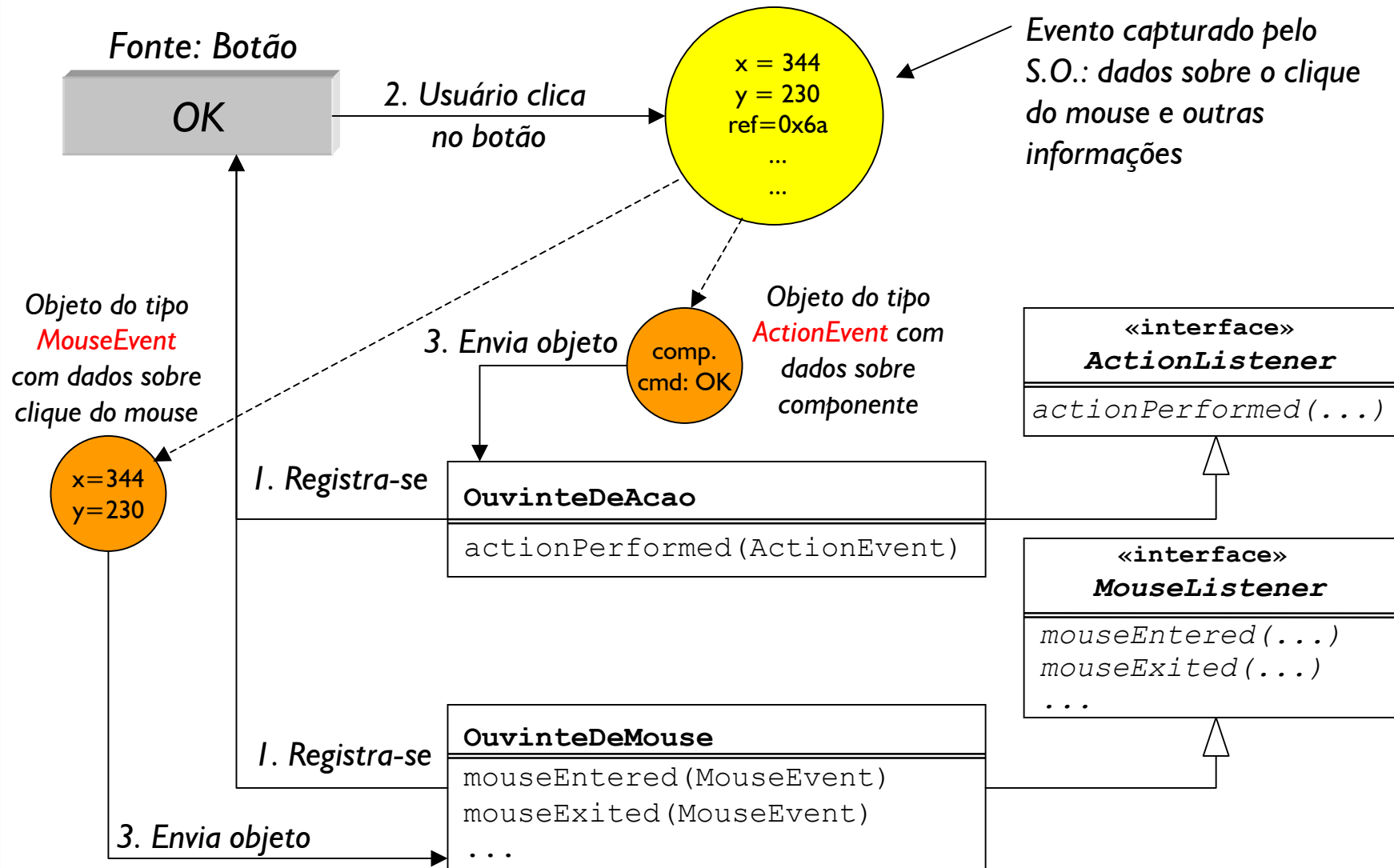
Use GridLayout para distribuir os botões

- 2. Experimente usar outros componentes e outros layouts (veja SwingSet demo)

- *Eventos em Java são objetos*
 - *Subclasses de `java.util.EventObject`*
- *Todo evento tem um objeto que é sua fonte*
 - `Object fonte = evento.getSource();`
- *Métodos de ouvintes (listeners) que desejam tratar eventos, recebem eventos como argumento*

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

Fontes, Eventos, Ouvintes



Eventos da Interface Gráfica

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

- Cada evento tem uma interface *Listener* correspondente que possui **métodos padrão** para tratá-los
 - **ActionEvent**: *ActionListener*
 - **MouseEvent**: *MouseListener* e *MouseMotionListener*
 - **ItemEvent**: *ItemListener*
 - **AdjustmentEvent**: *AdjustmentListener*
 - **TextEvent**: *TextListener*
 - **WindowEvent**: *WindowListener*
 - **FocusEvent**: *FocusListener*
 - **KeyEvent**: *KeyListener*
 - ...
 - **XXXEvent**: *XXXListener*

Como ligar a fonte ao listener

- Na ocorrência de um evento, em uma fonte, todos os listeners registrados serão notificados
 - É preciso antes cadastrar os listeners na fonte
`fonte.add<Listener>(referência_para_listener);`

- Exemplo:

- `JButton button = new JButton("Fonte");`
`ActionListener ouvinte1 = new OuvinteDoBotao();`
`MouseListener ouvinte2 = new OuvinteDeCliques();`
`button.addActionListener(ouvinte1);`
`button.addMouseListener(ouvinte2);`

- O mesmo objeto que é fonte às vezes também é listener, se implementar as interfaces

- Ainda assim, é necessário registrar a fonte ao listener (o objeto não adivinha que ele mesmo tem que capturar seus eventos)

```
this.addWindowListener(this);
```

Fonte



Listener



Como implementar um listener

- Crie uma nova classe que declare implementar o(s) listener(s) desejado(s)

```
public class MeuListener 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

Quais os listeners, métodos, eventos?

- Consulte a documentação no pacote `java.awt.event`
- Veja em cada **listener** a assinatura dos métodos que você deve implementar

```
public void actionPerformed(ActionEvent evt)
```

- Veja em cada **evento** os métodos que você pode chamar dentro do listener para obter as informações desejadas (por exemplo textos, coordenadas, teclas apertadas, etc.)

```
String comando = evt.getActionCommand();
```

- Veja em cada **componente-fonte** os métodos que você pode chamar para obter informações sobre o componente:

```
Object fonte = evt.getSource();  
if (fonte instanceof JButton)  
    JButton b = (JButton) fonte;  
    String label = b.getLabel();
```

Alguns Eventos, Listeners e Métodos

ActionEvent	ActionListener	<code>actionPerformed (ActionEvent)</code>
ItemEvent	ItemListener	<code>itemStateChanged (ItemEvent)</code>
KeyEvent	KeyListener	<code>keyPressed (KeyEvent)</code> <code>keyReleased (KeyEvent)</code> <code>keyTyped (KeyEvent)</code>
MouseEvent	MouseListener	<code>mouseClicked (MouseEvent)</code> <code>mouseEntered (MouseEvent)</code> <code>mouseExited (MouseEvent)</code> <code>mousePressed (MouseEvent)</code> <code>mouseReleased (MouseEvent)</code>
	MouseMotionListener	<code>mouseDragged (MouseEvent)</code> <code>mouseMoved (MouseEvent)</code>
TextEvent	TextListener	<code>textValueChanged (TextEvent)</code>
WindowEvent	WindowListener	<code>windowActivated (WindowEvent)</code> <code>windowClosed (WindowEvent)</code> <code>windowClosing (WindowEvent)</code> <code>windowDeactivated (WindowEvent)</code> <code>windowDeiconified (WindowEvent)</code> <code>windowIconified (WindowEvent)</code> <code>windowOpened (WindowEvent)</code>

- Alguns listeners possuem uma classe **Adapter** que implementa todos os métodos, sem instruções
 - Implementação vazia: {}
 - Só existe para listeners que têm **mais de um método**
- São úteis quando um Ouvinte precisa implementar apenas um dentre vários métodos de um Listener
 - Pode sobrepor a implementação desejada do método do Adapter e não precisa se preocupar com os outros
 - Não são úteis em ouvintes que já estendem outras classes ou quando implementam diferentes listeners
- O nome do adapter é semelhante ao do Listener
 - MouseListener: **MouseAdapter**,
 - WindowListener: **WindowAdapter**, ...

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

public class Swinggy9 extends JFrame {
    public Swinggy9(String nome) {
        (...)
        JButton b1 = new JButton("Sair");
        JButton b2 = new JButton("Viajar", icone);
        JTextField tf = new JTextField(10);
        b1.setActionCommand("Saindo");
        b2.setActionCommand("Viajando");

        ActionListener listener = new Eco();
        b1.addActionListener(listener);
        b2.addActionListener(listener);
        (...)
    }

    // Classe interna!!
    class Eco implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            tf.setText(e.getActionCommand());
        }
    }
}
```

Tratamento de eventos com classes internas

- É comum utilizar-se classes internas, mais especificamente, classes anônimas no tratamento de eventos de uma GUI
- Vantagens incluem a possibilidade de enxergar os componentes que geralmente são atributos `private`
- O exemplo anterior usa uma classe interna de instância (também chamada de classe aninhada, ou `embedded`)
- O exemplo abaixo usa classes anônimas (compare os dois!):

```
b1.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        tf.setText(e.getActionCommand());  
    }  
});
```

```
b2.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        tf.setText(e.getActionCommand());  
    }  
});    (...)
```

- 3. *Implemente os eventos para a aplicação*
 - *Copiar deve acrescentar o texto do JTextField no JEditorPane e limpar o JTextField*
 - *Limpar deve limpar o JTextField*
 - *Sair deve sair do programa*
- 4. *Implemente os botões como itens do menu "Operações"*
 - *Use JMenuBar, JMenu e JMenuItem*
- 5. *Implemente um JToggleButton "desenhar/escrever" que troque o JTextPane por um JCanvas e permita rabiscar com o mouse (use MouseEvent)*
 - *Veja aplicação-exemplo no CD*

- 6. *Implemente uma interface gráfica para a aplicação da biblioteca (capítulo 7).*
 - *Use o JTabbedPane para criar diferentes painéis: um para entrada de Agentes (autores, editores), outro para entrada de Publicações (livros, revistas, artigos) e outro para buscas.*
 - *Use um combo-box para selecionar cada tipo de agente ou publicação*
 - *Desabilite campos não utilizados.*
 - *Implemente os eventos chamando os métodos da fachada (Biblioteca)*

Curso J100: Java 2 Standard Edition

Revisão 17.0

© 1996-2003, Helder da Rocha
(helder@acm.org)

 argonavis.com.br