

# Programação por Objectos

## Interfaces gráficas

# Introdução (1)

- O Java disponibiliza um conjunto de pacotes para geração de unidades de **interface gráfica (GUI)**.
  - **AWT** (*Abstract Window Toolkit*)
    - pacote java.awt (**import java.awt.\*;**)
    - disponibilizado no JSE 1.1
    - dependente de código nativo
  - **Swing**
    - pacote javax.swing (**import javax.swing.\*;**)
    - disponibilizado no JSE 1.2
    - expande o AWT (17 pacotes na versão 1.4)

# Introdução (2)

- O Swing possui componentes, que seguem a arquitectura MVC:

- **Modelo**

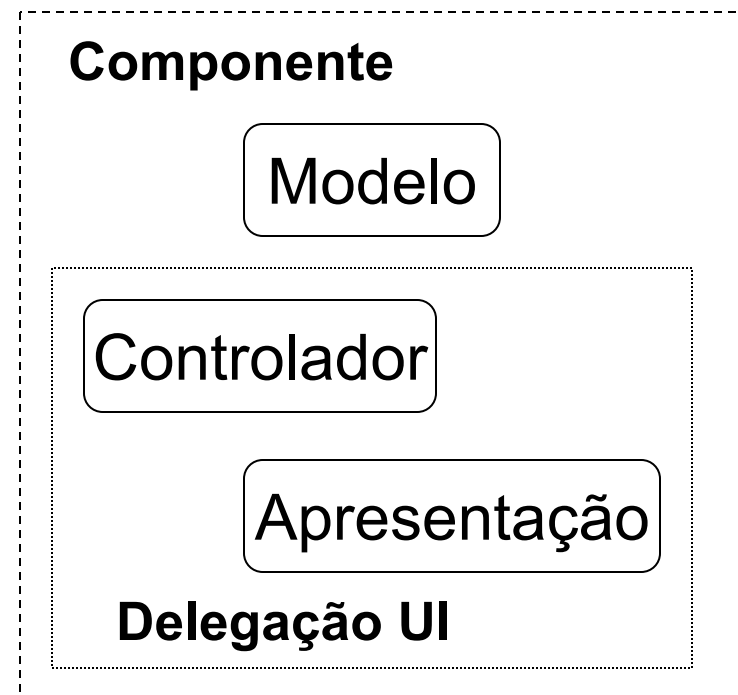
Espaço de dados do componente (ex: barra de elevador possui 3 posições-mínimo, médio e máximo).

- **Apresentação** (*view*)

Forma de visualização do componente (ex: janela Windows, com botão de fecho no canto superior direito).

- **Controlador**

Forma como o componente interage com eventos (ex: clicar no rato).



# Introdução (3)

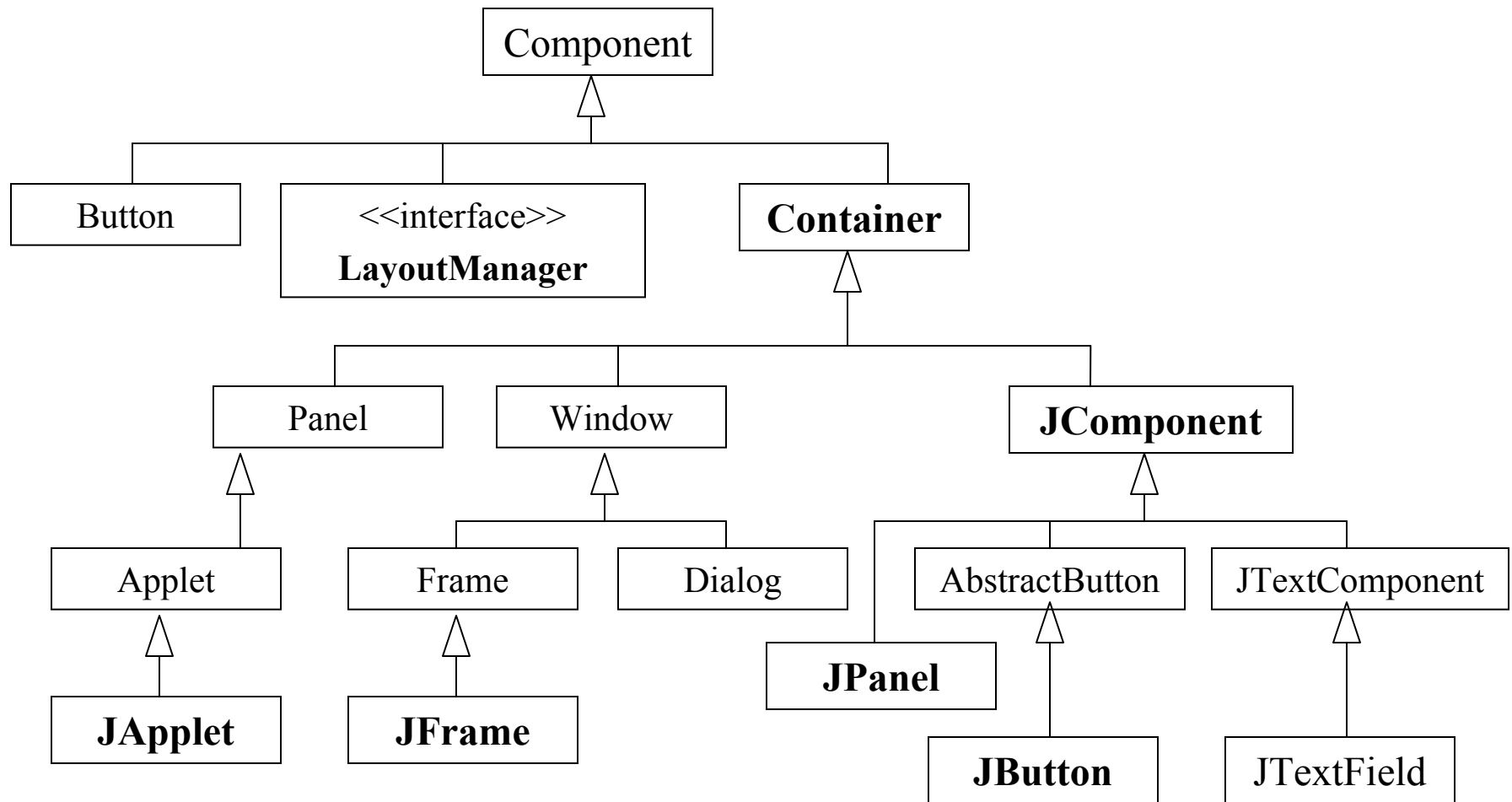
- Componentes do Swing\*

- JComponent
  - JPanel
  - JButton
  - ...
- JFrame
- JDialog
- JApplet

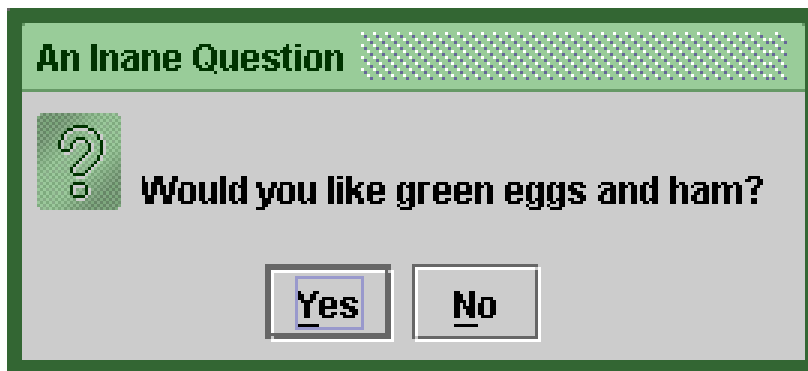
Tudo o que surge na janela faz parte de uma hierarquia de um (ou mais) contentores de inclusão de instâncias de `JFrame`, `JDialog` ou `JApplet`.

**Nota: componentes do AWT possuem mesmo identificador, sem prefixo J!**

# Introdução (4)

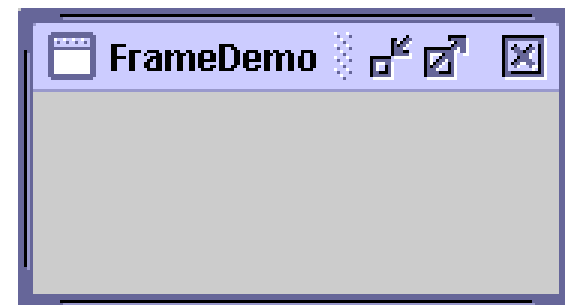


# Componentes Swing: contentores de topo



**JDialog**

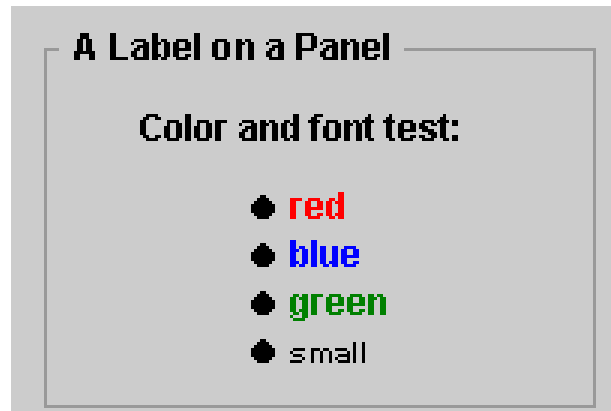
As caixas de dialogo são normalmente mais simples. Existem várias predefinidas em “JOptionPane”. Estas são normalmente *Modal*. O *thread* corrente fica suspenso a espera da resposta do utilizador.



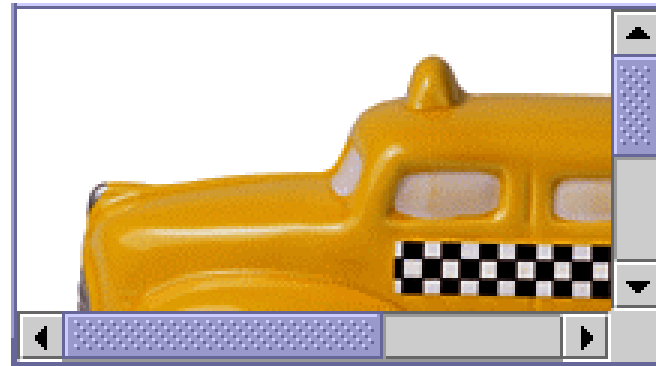
**JFrame**

JFrame corresponde a uma janela normal com uma frame (o contorno da janela que permite o dimensionamento e reposicionamento).

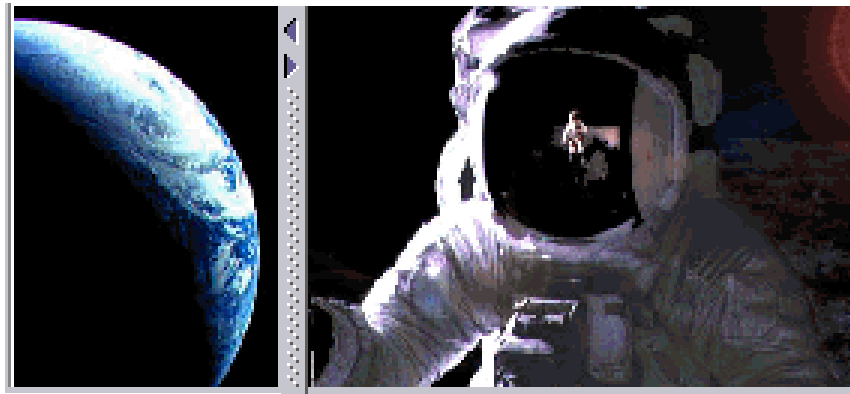
# Componentes Swing: contentores intermédios



**JPanel**



**JScrollPane**



**JSplitPane**

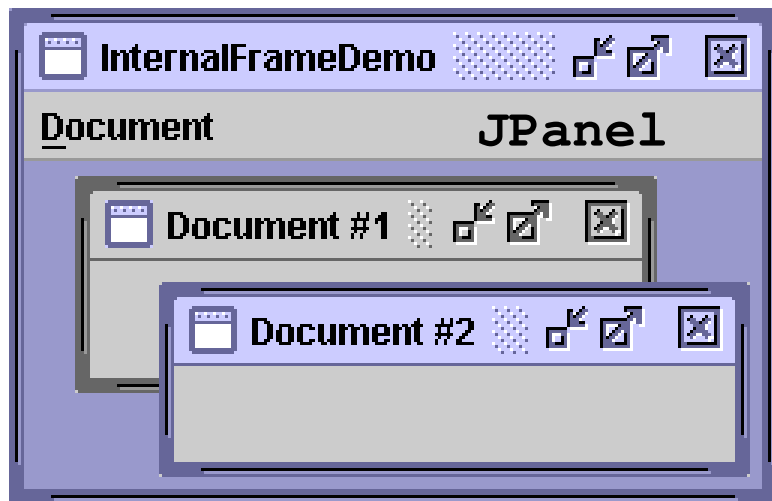


**JTabbedPane**

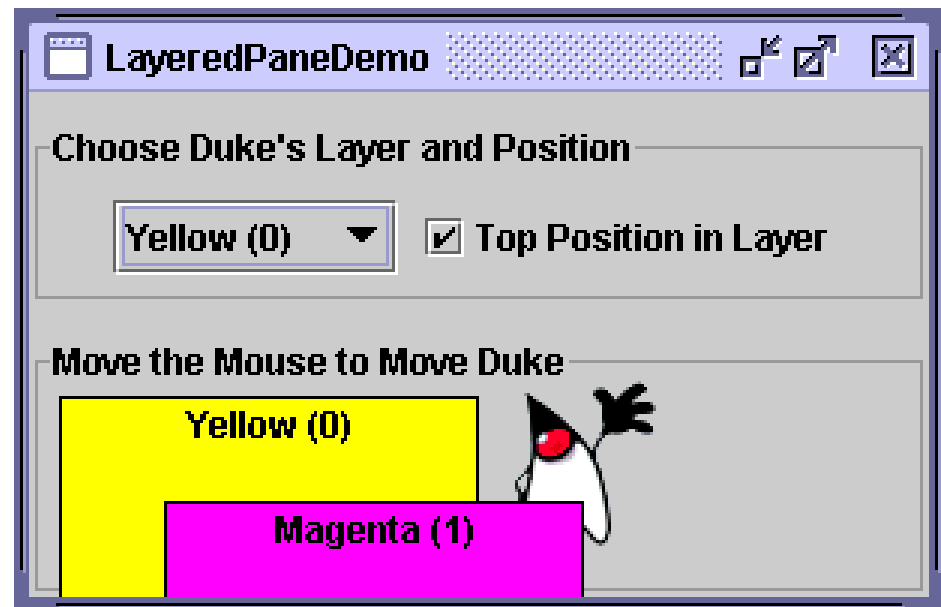


**JToolBar**

# Componentes Swing: contentores especiais



**JInternalFrame**



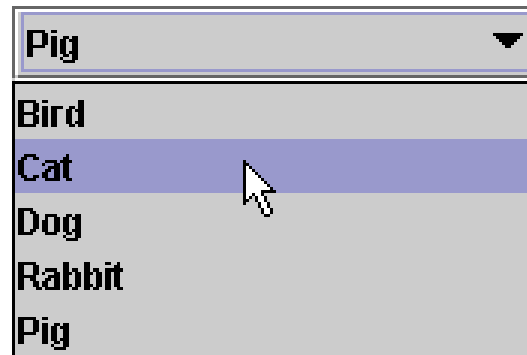
**JLayeredPane**



# Componentes Swing: componentes básicas



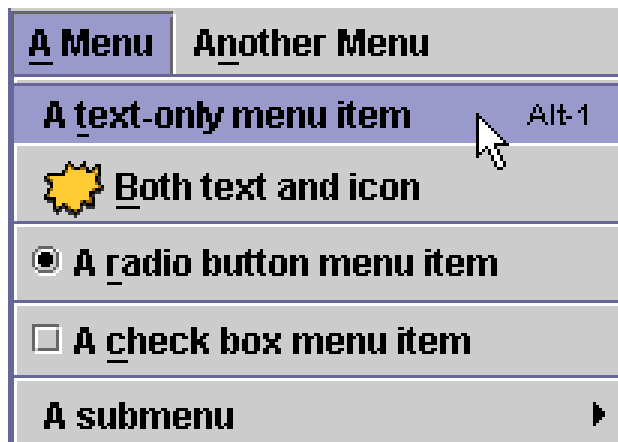
JButton



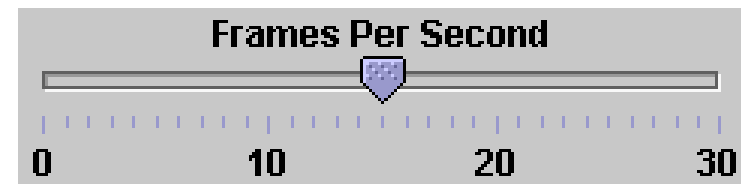
JComboBox



JList



JMenu



JSlider

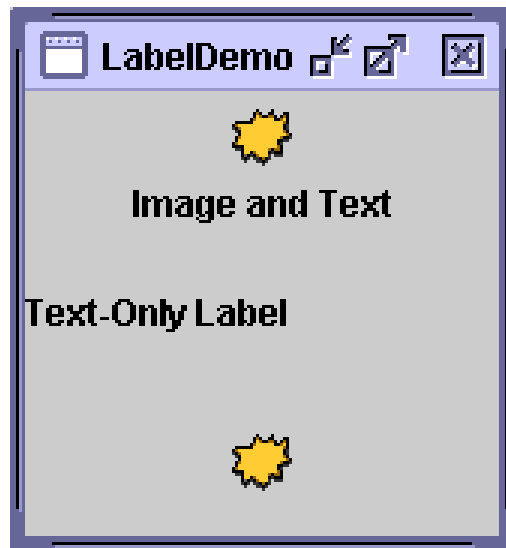


JSpinner



JTextField

# Componentes Swing: componentes básicas



**JLabel**

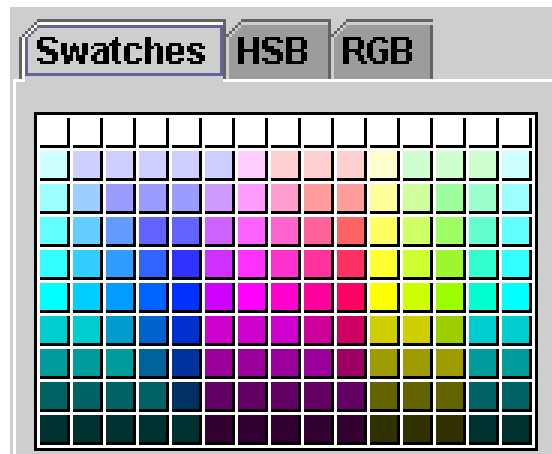


**JProgressBar**



**JToolTip**

# Componentes Swing: componentes avançadas



**JColorChooser**

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

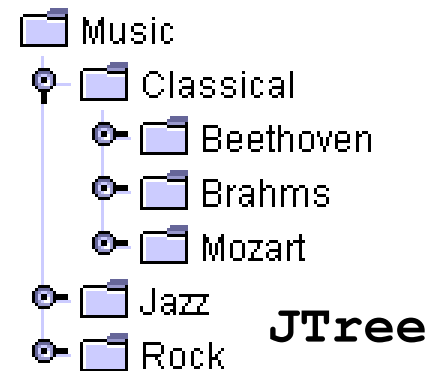
**JTable**



**JFileChooser**

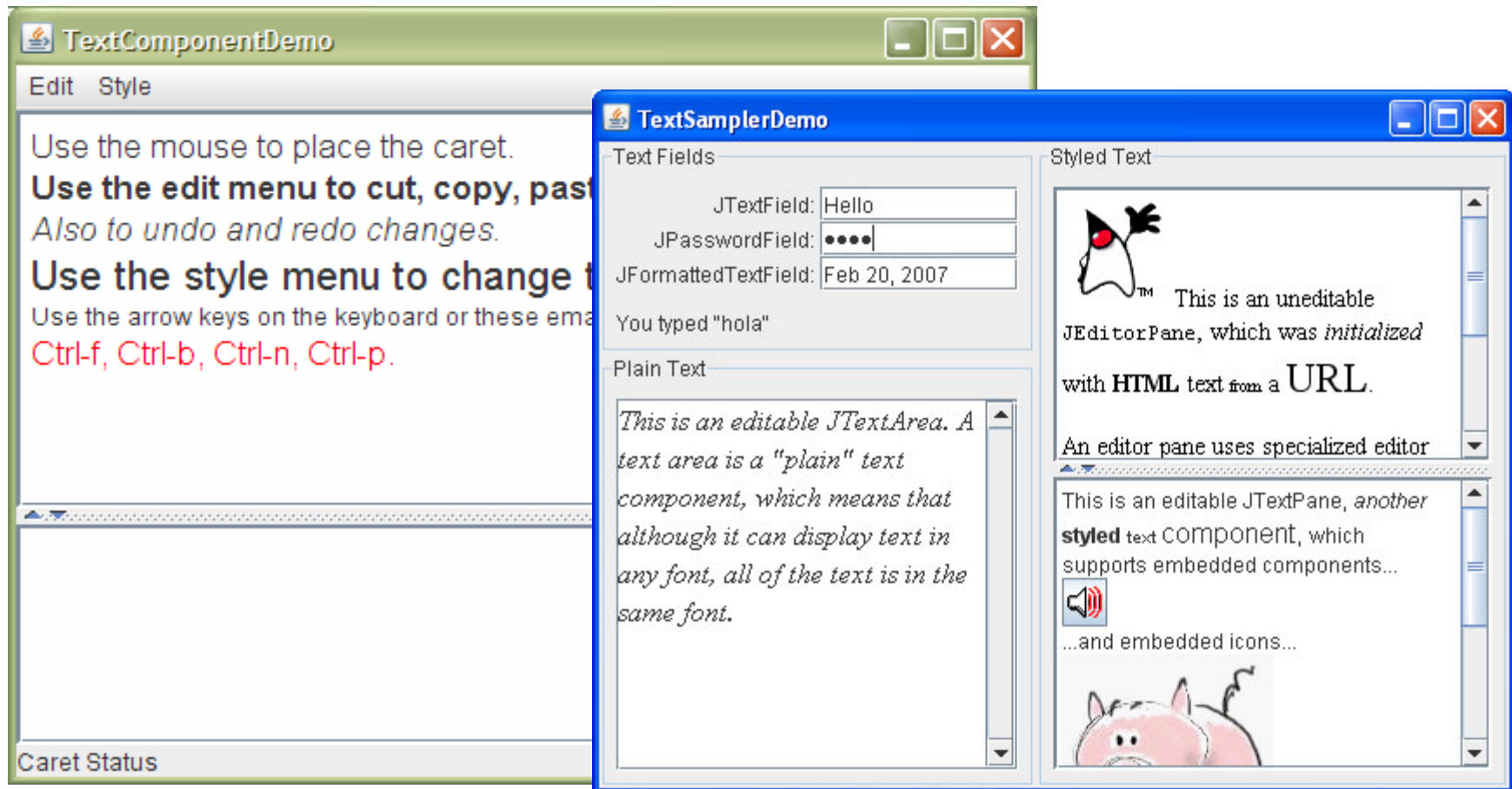


**JText**



**JTree**

# Componentes Swing: componentes avançadas



# JEditorPane

- **text/plain**
  - Plain text, which is the default the type given isn't recognized. The kit used in this case is an extension of DefaultEditorKit that produces a wrapped plain text view.
- **text/html**
  - HTML text. The kit used in this case is the class `javax.swing.text.html.HTMLEditorKit` which provides HTML 3.2 support.
- **text/rtf**
  - RTF text. The kit used in this case is the class `javax.swing.text.rtf.RTFEditorKit` which provides a limited support of the Rich Text Format.

# Contentor (1)

- Um **contentor** recolhe todos os componentes a visualizar na janela.
  - Os componentes vão sendo inseridos a partir do fundo (método `add`), em direcção do utilizador, formando uma pilha.
  - Se necessário, os componentes podem ser deslocados para uma posição específica na pilha, ou eliminados.
- Há dois tipos de contentores:
  - **Panel** (rectângulo com componentes)
  - **Window** (janela)

# Contentor (2)

- Métodos:
  - **Component add(Component comp)**  
adiciona componente no topo da pilha
  - **Component add(Component comp, int index)**  
adiciona componente numa posição específica
  - **void remove(int index)**  
remove componente colocado numa posição
  - **void remove(Component comp)**  
remove componente da pilha

# Componente

- Um **componente** é a representação gráfica de tudo que pode ser desenhado no écran, podendo eventualmente interagir com o utilizador.
- As fronteiras são obrigatoriamente rectangulares (mesmo que o objecto seja um círculo):
  - A **dimensão** de um componente é um objecto **Dimension**, de campos `int height, width` (altura e comprimento)
  - As posições dos 4 lados de um componente num contentor é um objecto **Insets**, de campos `int bottom, left, right, top`.

```
java.lang.Object
├── java.awt.Component
│   ├── java.awt.Container
│       └── javax.swing.Component
```



# Componente

- Cada componente possui o seu sistema de coordenadas ((0,0) no canto superior esquerdo).
  - Uma **posição** é objecto de classe **Point**, de campos `int x, y`.



# Componente

- Alteração da posição do contentor, onde o componente é posicionado, não modifica as coordenadas locais!
- Alguns métodos (227 ao todo, muitos herdados de `Container`):
  - 1. **Posição e área**
    - **`Dimension getSize(Dimension rv)`**  
coloca em `rv` dimensões do componente  
[**`setSize`** altera dimensões]
    - **`Dimension getPreferredSize()`**  
obtém dimensões básicas do componente  
[**`setPreferredSize`** altera dimensões básicas]

# Componente

- **Point getLocation(Point rv)**  
coloca em `rv` localização do componente no contentor ascendente  
[**setLocation** altera localização do componente]
- **int getHeight()**  
obtém altura  
[**getWidth** obtém largura]
- **int getX()**  
obtém posição horizontal  
[**getY** obtém posição vertical]
- **Rectangle getBounds()**  
obtém dimensões da componente  
[**setBounds** desloca e altera dimensões do componente]

# Componente

- **Insets** `getInsets(Component comp)`  
devolve as margens interiores dos 4 lados do componente, podem corresponder a espaços ou fronteiras ou um texto.
- **Container** `getParent()`  
obtém contentor onde componente foi inserido

## 2. Aspecto

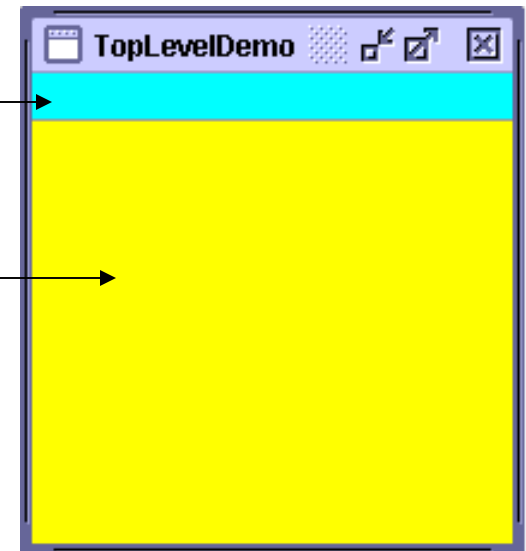
- **void** `paint(Graphics comp)`  
afixa componente no écran

## 3. Reacção a eventos

## 4. Estado

# Classe JFrame

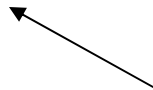
- O contentor **JFrame** contém:
  - Barra de menus (opcional):  
Os menus contêm botões, que chamam métodos quando o rato é premido.
  - Instância de **JPanel**:  
O painel pode conter contentores, JFrame, JDialog ou JApplet, uns em cima dos outros.



```
java.lang.Object
├── java.awt.Component
│   ├── java.awt.Container
│   │   ├── java.awt.Window
│   │   │   ├── java.awt.Frame
│   │   │   │   └── javax.swing.JFrame
```

# Classe JFrame

- Parâmetros do **construtor de JFrame**:
  - **String title**: título da janela.
  - **GraphicsConfiguration gc**: configuração da janela.
- atributos dependem do sistema gráfico adoptado (X11, MS Windows, ...).
- por omissão, o *frame* inicial é invisível.
- Campos:
  - **int EXIT\_ON\_CLOSE**: acção no fecho da talha.



# Classe JFrame

- Métodos:
  - **void setVisible(boolean b)**  
torna janela visível (`true`) ou invisível (`false`)
  - **setDefaultCloseOperation(int operation)**  
define operação quando utilizador fecha janela (usualmente parâmetro é `JFrame.EXIT_ON_CLOSE` mas também pode ser `JFrame.DISPOSE_ON_CLOSE`)
  - **setSize(int width, int height)**  
fixa dimensão do *frame*
  - **pack()**  
dimensiona o *frame* para conter todos os componentes
  - **Container getContentPane()**  
devolve o painel
  - **void setContentPane(Container pane)**  
define o painel

# Classe JFrame

- Neste capítulo são descritos aspectos básicos de inserção no JPanel:
  - Textos (objectos JLabel)
  - Gráficos 2D (objectos Graphics2D)
  - Botões (objectos JButton)


**(Swing é muito mais potente, manual *Java Swing* da O'Reilly tem 1200+ páginas!)**



# Visualização (1)

- Textos e imagens são visualizados por instâncias de `JLabel`.
- Parâmetros do **construtor de `JLabel`**:
  - **`String text`**: texto a visualizar
  - **`Icon image`**: imagem a visualizar
  - **`int horizontalAlignment`**: alinhamento

```
java.lang.Object
├── java.awt.Component
│   ├── java.awt.Container
│   │   ├── javax.swing.JComponent
│   │   │   └── javax.swing.JLabel
```

- 
- `SwingConstants.LEFT`
  - `SwingConstants.CENTER` (omissão figura)
  - `SwingConstants.RIGHT`
  - `SwingConstants.LEADING` (omissão texto)
  - `SwingConstants.TRAILING`

# Visualização (2)

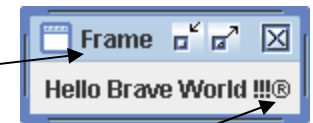
- Métodos:
  - `void setText(String text)`  
altera texto a visualizar
  - `void setHorizontalAlignment(int align)`  
define o alinhamento horizontal do texto

# Visualização (3)

```
import java.awt.*;
import javax.swing.*;
public static void main(String[] args) {
    // constroi frame
    JFrame.setDefaultLookAndFeelDecorated(true);
    JFrame frame = new JFrame("Frame");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // constroi painel de texto
    JPanel p = new JPanel();
    p.add(new JLabel("Hello Brave World !!!\u00ae "));

    frame.setContentPane(p);
    frame.pack();
    frame.setVisible(true);
}
```



# Visualização (4)

- Reposicionamento de um objecto gráfico pode ser efectuado nas seguintes etapas:
  1. Recolher dimensões do objecto (`PreferredSize`)
  2. Deslocar objecto (`setLocation` ou `setBounds`)
  3. Alterar dimensões do *frame* (`setSize`)
  4. Tornar o *frame* visível (`setVisible`)

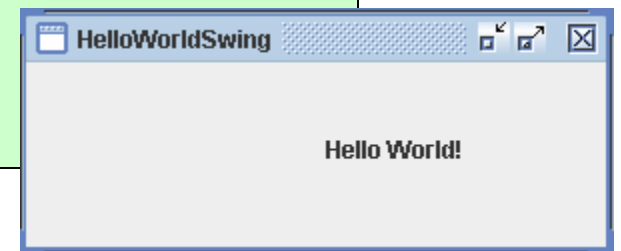
# Visualização (5)

```
// *** imprime mensagem deslocada
public static void main(String[] args) {
    JFrame.setDefaultLookAndFeelDecorated(true);
    JFrame frame = new JFrame("HelloWorldSwing");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    Container pane = frame.getContentPane();
    pane.setLayout(null);

    JLabel b1 = new JLabel("Hello World");
    pane.add(b1);

    // *** desloca o texto
    Dimension size = b1.getPreferredSize();
    b1.setBounds(150, 25, size.width, size.height);
    // *** altera dimensoes da janela
    frame.setSize(300, 100);
    frame.setVisible(true);
}
```



# Gráficos (1)

- Capacidades gráficas do `Graphics2D`:
  - Textos
  - Linhas, rectângulos, polígonos, ovais
  - Cores, fontes e preenchimento (*fill*) de zonas

```
java.lang.Object
├── java.awt.Graphics
│   └── java.awt.Graphics2D
```

# Gráficos (2)

- Gráficos 2D são desenhados, numa extensão da classe `JPanel`, no método:
  - `paintComponent(Graphics g)`
- O método `paintComponent` nunca é chamado directamente.
- O método `paint` chama `paintComponent`, `paintBorder`, and `paintChildren` por esta ordem.
- Todos os objectos são inseridos em coordenadas fixas: se a janela for redimensionada para tamanho menor, pode haver corte de objectos (ou mesmo desaparecimento).

```
public class DrawingPanel extends JPanel {  
    public void paintComponent(Graphics g) {  
        Graphics2D g2 = (Graphics2D) g;  
        super.paintComponent(g);  
        // inserção em g de objectos gráficos  
        // ...  
    }  
}
```

# Gráficos (3)

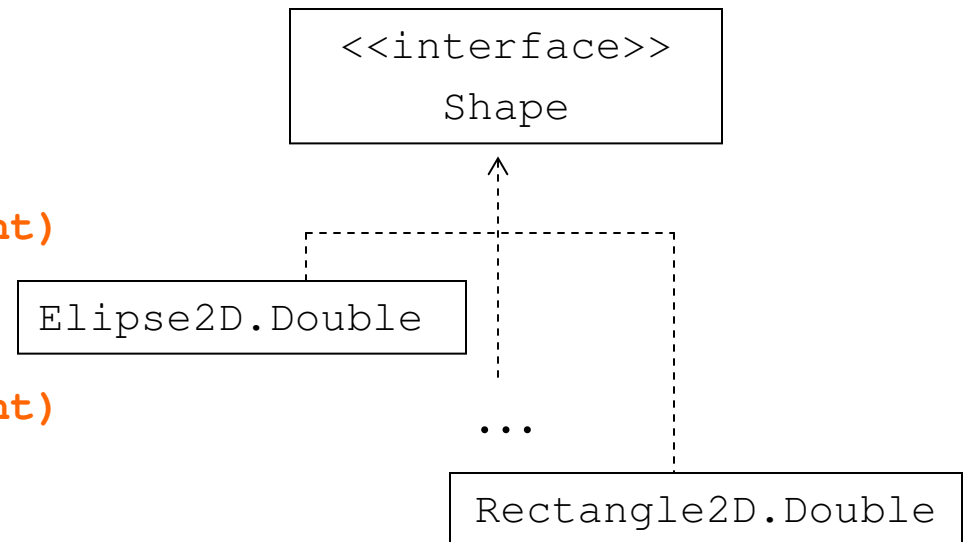
- Métodos:
  - `drawLine(int x1, int y1, int x2, int y2)`  
linha entre (x1,y1) e (x2,y2)
  - `drawOval(int x, int y, int width, int height)`  
oval
  - `drawPolygon(int[] xPoints, int[] yPoints, int nPoints)`  
polígono fechado, definido por tabelas de coordenadas
  - `drawRect(int x, int y, int width, int height)`  
rectângulo
  - `drawString(String str, int x, int y)`  
escreve cadeia de caracteres



# Gráficos (4)

- Interface **Shape** determina definições básicas de objectos geométricos. As classes estáticas de implementação encontram-se no pacote **java.awt.geom**.

- **Ellipse2D.Double**(  
double x, double y,  
double width, double height)
- **Rectangle2D.Double**(  
double x, double y,  
double width, double height)
- **RoundRectangle2D.Double**(  
double x, double y,  
double width, double height)



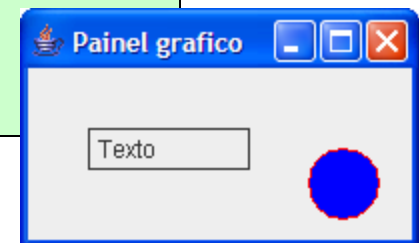
# Gráficos (5)

- A classe `Color` define campos das principais cores (black, blue, green, red,...).
- Cores alteradas a partir da chamada do método:
  - `setPaint(Color c)`
- Métodos `Graphics2D` de alteração de cores:
  - `draw(Shape s)`: contorno do objecto
  - `fill(Shape s)`: interior do objecto

```
java.lang.Object
└── java.awt.Color
```

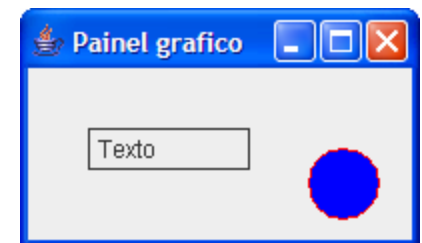
# Gráficos (6)

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
public class DrawingPanel extends JPanel {
    public void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        super.paintComponent(g);
        /*** insere rectangulo com texto
        g2.drawRect(30,30,80,20);
        g2.drawString("Texto",35,45);
        /*** insere circulo azul
        Ellipse2D circ = new Ellipse2D.Double(140,40,35,35);
        g2.setPaint(Color.blue);
        g2.fill(circ);
        g2.setPaint(Color.red);
        g2.draw(circ);
    }
}
```



# Gráficos (7)

```
public static void main(String[] args) {  
    JFrame f = new JFrame("Painel grafico");  
    f.setSize(200,120);  
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    f.getContentPane().add(new DrawingPanel());  
    f.setVisible(true);  
}
```



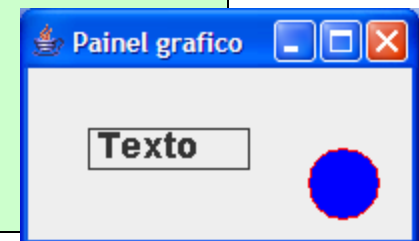
# Gráficos (8)

- Fontes são objectos **Font** e o **construtor de Font** possui 3 parâmetros:
  - **String name** (p.e., “SansSerif”)
  - **int style** (campos de `Font`, p.e., `Font.PLAIN` e `Font.BOLD`)
  - **int size** (tamanho da fonte)

```
java.lang.Object
└── java.awt.Font
```

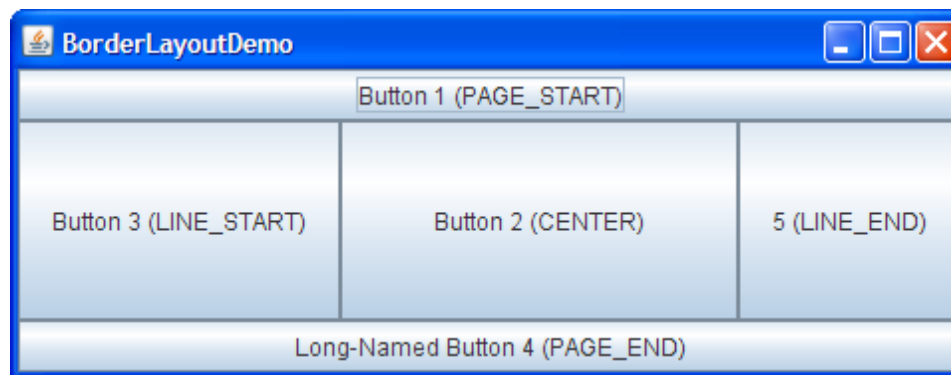
# Gráficos (9)

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
public class DrawingPanel extends JPanel {
    public void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        super.paintComponent(g);
        /*** insere rectangulo com texto com fonte pretendida
        g2.drawRect(30,30,80,20);
        Font f = new Font("SansSerif",Font.BOLD,18);
        g2.setFont(f);
        g2.drawString("Texto",35,45);
        /*** insere circulo azul
        Ellipse2D circ = new Ellipse2D.Double(140,40,35,35);
        g2.setPaint(Color.blue);
        g2.fill(circ);
        g2.setPaint(Color.red);
        g2.draw(circ);
    }
}
```



# Disposição gráfica (1)

- O Swing fornece diversas formas de disposição gráfica (*layout*) de contentores:
  - A interface **LayoutManager** define as operações básicas de disposição de contentores em áreas distintas.
  - O Swing fornece ao programador 7 implementações:
    1. **BorderLayout**: [omissão do Window] dispõe de 5 áreas, topo (PAGE\_START), base (PAGE\_END) e três a meio (LINE\_START, CENTER, LINE\_END).



# Disposição gráfica (2)

- 2. **BoxLayout**: dispõe os componentes na vertical ou na horizontal.
- 3. **CardLayout**: permite variar componentes de uma área.
- 4. **FlowLayout**: [omissão do Panel] dispõe os objectos uns ao lado dos outros.
- 5. **GridBagLayout**: distribui os componentes numa grelha, possibilitando que um componente ocupe mais de uma célula.
- 6. **GridLayout**: dispõe contentores numa matriz.
- 7. **SpringLayout**: disponibiliza um controlo das distâncias entre as fronteiras de cada componente.



# BoxLayout (1)

- Parâmetros do **construtor de BoxLayout**:
  - **Container target**
  - **int axis**: eixos de empilhamento dos componentes
    - **BoxLayout.PAGE\_AXIS** – vertical, do topo para a base
      - Normalmente equivalente a **Y\_AXIS** que já não deve ser utilizado
    - **BoxLayout.LINE\_AXIS** – horizontal, da esquerda para a direita
      - Normalmente equivalente a **X\_AXIS** que já não deve ser utilizado
  - **Exemplo**:

```
JPanel panel = new JPanel();
panel.setLayout(new BoxLayout(panel, BoxLayout.PAGE_AXIS));
```

Enganos num dos “panels” dá excepções do tipo “BoxLayout can't be shared”

## BoxLayout (2)

```
import java.awt.*;
import javax.swing.*;
public class MyComponent extends JComponent {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        // *** recolhe dimensoes
        int width = getWidth();
        int height = getHeight();

        // *** desenha rectângulo vermelho
        g.setColor(Color.red);
        g.drawRect(0, 0, width-1, height-1);

        // *** desenha yo!
        g.setColor(Color.black);
        g.drawString("yo!", 20, 20);
    }
}
```



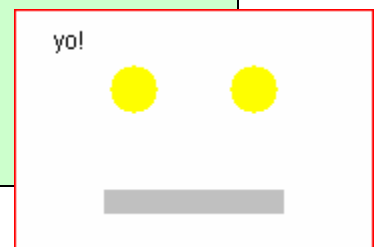
# BoxLayout (3)

```
// *** desenha olhos
int eyeY = height/3;
int left = width/3;
int right = 2*width/3;
int radius = width/15;
g.setColor(Color.yellow);
g.fillOval(left-radius, eyeY-radius, radius*2, radius*2);
g.fillOval(right-radius, eyeY-radius, radius*2, radius*2);

// *** desenha boca
g.setColor(Color.lightGray);
g.fillRect(width/4, 3*height/4, width/2, height/10);

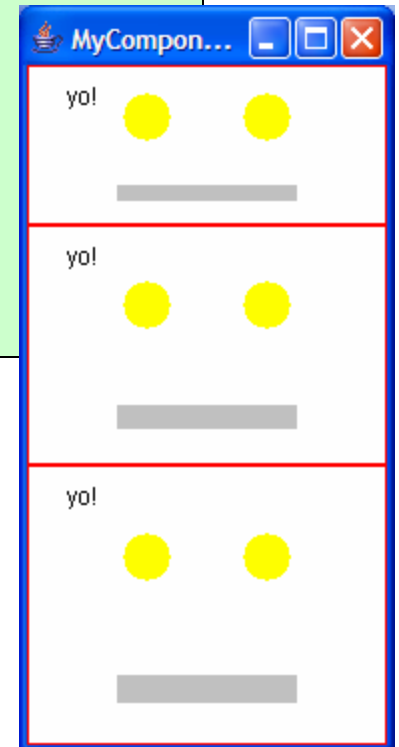
}

public MyComponent(int width, int height) {
    super();
    setPreferredSize(new Dimension(width,height));
}
}
```



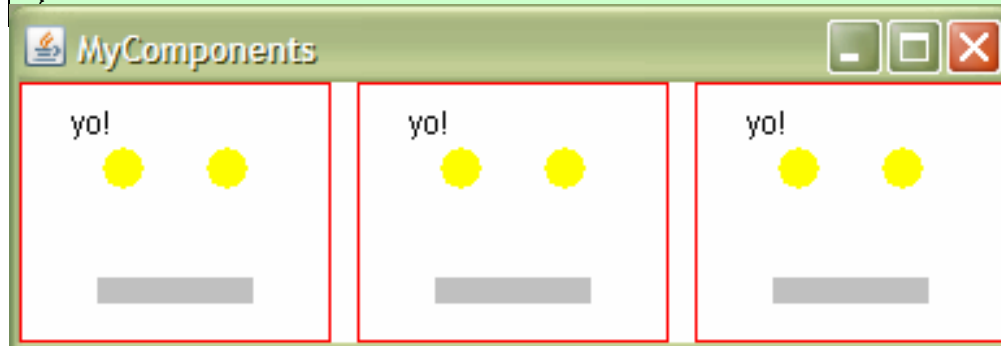
# BoxLayout (4)

```
public static void main(String[] args) {  
    JFrame frame = new JFrame("MyComponents");  
    JComponent content = (JComponent) frame.getContentPane();  
    content.setBackground(Color.white);  
    content.setLayout(new BoxLayout(content, BoxLayout.Y_AXIS));  
    content.add(new MyComponent(180, 80));  
    content.add(new MyComponent(180, 120));  
    content.add(new MyComponent(180, 140));  
    frame.pack();  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.setVisible(true);  
}
```



# BoxLayout (5)

```
public static void main(String[] args) {  
    JFrame frame = new JFrame("MyComponents");  
    JComponent content = (JComponent) frame.getContentPane();  
    content.setBackground(Color.white);  
    content.setLayout(new BoxLayout(content, BoxLayout.X_AXIS));  
    content.add(new MyComponent(120, 100));  
    content.add(Box.createHorizontalStrut(10));  
    content.add(new MyComponent(120, 100));  
    content.add(Box.createHorizontalStrut(10));  
    content.add(new MyComponent(120, 100));  
    frame.pack();  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.setVisible(true);  
}
```



use  
`createHorizontalStrut(...)`  
Para colocar espaços.

# BoxLayout

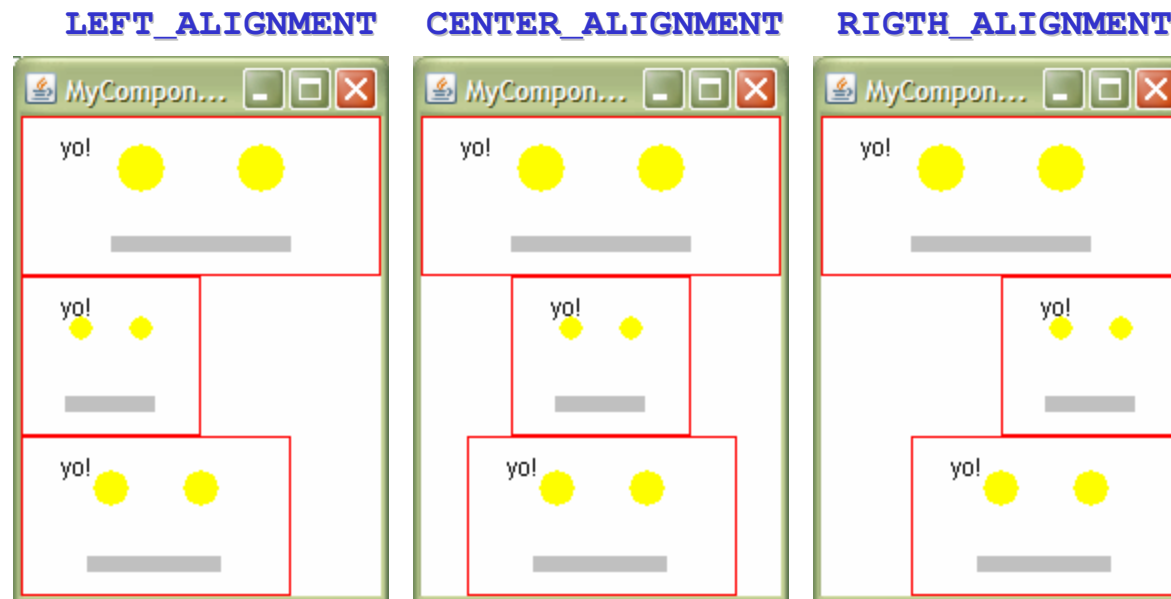
- Ao redefinir o tamanho de um contentor deve ser ter cuidado de usar igualmente,
  - `setPreferredSize`,
  - `setMaximumSize`
  - `setMinimumSize`
- para que todos os parâmetros tenham valores coerentes.

```
public MyComponent(int width, int height) {  
    super();  
    setPreferredSize(new Dimension(width,height));  
    setMaximumSize(new Dimension(width,height));  
    setMinimumSize(new Dimension(width,height));  
    setAlignmentX(JComponent.LEFT_ALIGNMENT);  
}
```

# BoxLayout

```
public MyComponent(int width, int height) {  
    super();  
    setPreferredSize(new Dimension(width,height));  
    setMaximumSize(new Dimension(width,height));  
    setMinimumSize(new Dimension(width,height));  
    setAlignmentX(JComponent.LEFT_ALIGNMENT);  
}
```

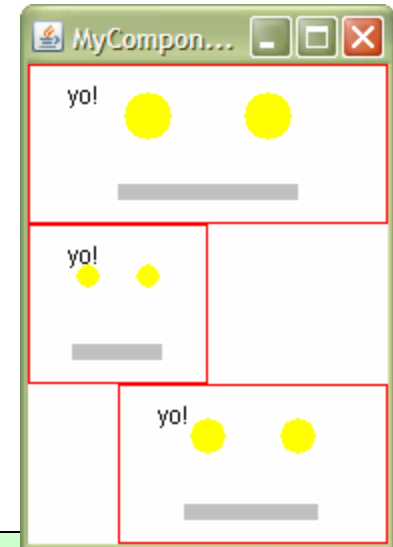
Todos os componentes num **BoxLayout** devem todos ter o mesmo alinhamento.



O alinhamento é um atributo do componente e não do contentor.

# BoxLayout

- Alinhamento misto num BoxLayout
  - Definir um novo JPanel
  - Usar `Box.createHorizontalGlue()`



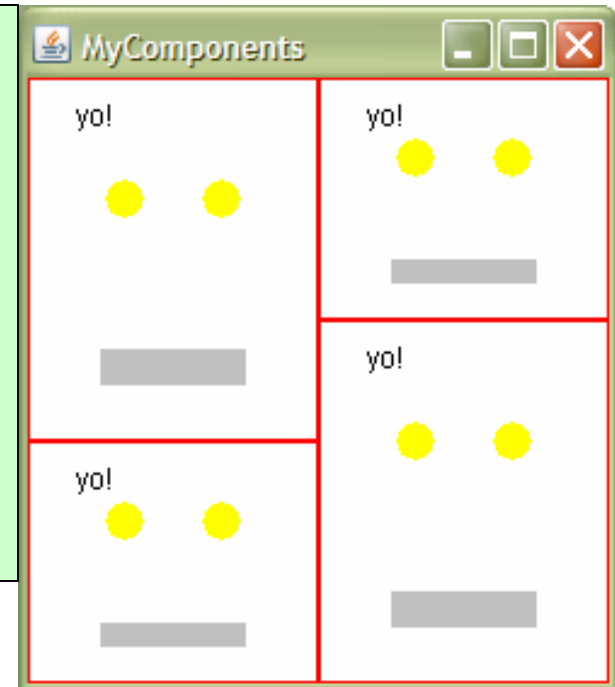
```
JComponent content = (JComponent) frame.getContentPane();
content.setBackground(Color.white);
content.setLayout(new BoxLayout(content, BoxLayout.Y_AXIS));
content.add(new MyComponent(180, 80));
    JPanel panel = new JPanel();
    panel.setBackground(Color.white);
    panel.setLayout(new BoxLayout(panel, BoxLayout.X_AXIS));
    panel.add(new MyComponent(90, 80));
    panel.add(Box.createHorizontalGlue());
    panel.setAlignmentX(JComponent.RIGHT_ALIGNMENT);
content.add(panel);
content.add(new MyComponent(135, 80));
```



# BoxLayout

- Combinando caixas com vertical e horizontal layout.

```
content.setBackground(Color.white);
content.setLayout(new
BoxLayout(content,BoxLayout.LINE_AXIS));
    Box left = Box.createVerticalBox();
    left.add(new MyComponent(120,150));
    left.add(new MyComponent(120,100));
content.add(left);
    Box righth = Box.createVerticalBox();
    righth.add(new MyComponent(120,100));
    righth.add(new MyComponent(120,150));
content.add(righth);
```

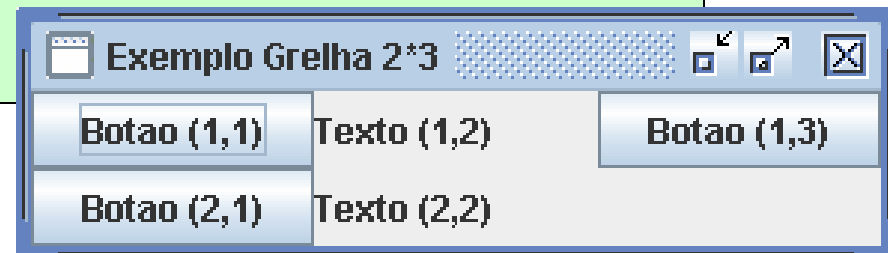


# GridLayout (1)

- Parâmetros do **construtor de GridLayout**:
  - **int rows**: número de linhas (1º parâmetro)
  - **int columns**: número de colunas (2º parâmetro)
  - **int hgap**: distância na horizontal entre células (3º parâmetro)
  - **int vgap**: distância na vertical entre células (4º parâmetro)

# GridLayout

```
public static void main(String[] args) {  
    JFrame.setDefaultLookAndFeelDecorated(true);  
    JFrame frame = new JFrame("Exemplo Grelha 2*3");  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    JPanel pane = new JPanel();  
    pane.setLayout(new GridLayout(2,3));  
    pane.add(new JButton("Botao (1,1)"));  
    pane.add(new JLabel("Texto (1,2)"));  
    pane.add(new JButton("Botao (1,3)"));  
    pane.add(new JButton("Botao (2,1)"));  
    pane.add(new JLabel("Texto (2,2)"));  
  
    frame.setContentPane(pane);  
    frame.pack();  
    frame.setVisible(true);  
}
```



# GridBagLayout

```
JPanel pane = new JPanel(new GridBagLayout());  
GridBagConstraints c = new GridBagConstraints();
```

**c.gridx** and **c.gridy** : Escolhe a posição na grelha

**c.gridwidth**, **c.gridheight** : A altura e largura do componente em zonas da grid

**c.weightx**, **c.weighty** : Especifica como espaço extra (ou em falta) deve ser distribuído pelos componentes. Componentes com maior peso recebem mais espaço componente com menos peso recebem menos.

Etc...

```
pane.add(theComponent, c);
```

# Eventos

- Um **evento** é um acontecimento atómico de um dispositivo externo (ex: clicar num botão do rato ou premir caracter no teclado).
- **As acções desencadeadas pelo evento são executadas através da chamada de um método específico, devendo o programador indicar o objecto ouvinte (*listener*) que fica à espera do evento.**
- Classes de manipulação dos eventos devem importar **`import java.awt.event.*;`**

# Eventos

- Swing define 10 tipos de eventos, gerados por componentes:
  - **ActionEvent** (componentes: Button, List, MenuItem)
  - AdjustmentEvent (componente: Scrollbar)
  - ComponentEvent (componentes: Choice, Component)
  - ContainerEvent (componente: Container)
  - FocusEvent (componente: Component)
  - ItemEvent (componentes: CheckBox, List)
  - KeyEvent (componente: Component)
  - MouseEvent (componente: Component)
  - TextEvent (componente: TextComponent)
  - WindowEvent (componente: Window)

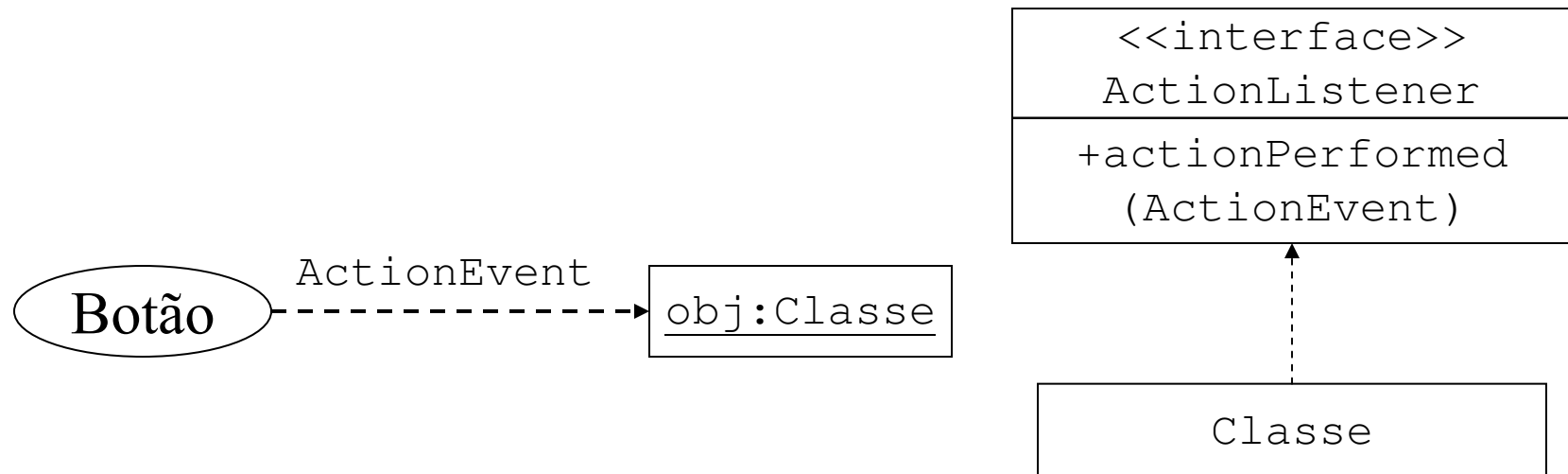
# Eventos

- O objecto à espera deve implementar uma interface, que depende do tipo de evento gerado.

Evento	Interface	Métodos
<b>ActionEvent</b>	<b>ActionListener</b>	<b>actionPerformed()</b>
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged()
ComponentEvent	ComponentListener	componentHidden() componentMoved() componentResized() componentShown()
ContainerEvent	ContainerListener	componentAdded() componentRemoved()

# Eventos

- Um botão é uma instância de **JButton**.
- No AWT/Swing, a classe implementa a interface `ActionListener`, e o método `actionPerformed` processa o evento.





# Eventos

- Forma de um objecto tratar, no Swing, um evento `ActionEvent`:

1. Implementar interface `ActionListener`

```
class MyClass implements ActionListener {
```

2. Dentro da classe definir o método

```
void actionPerformed(ActionEvent e) {  
    // código de reacção ao evento  
}
```

```
}
```

3. Adicionar, no botão, o objecto que processa o evento

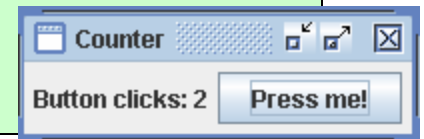
```
void addActionListener(ActionListener l)
```

# Eventos

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class SwingApplication {
    private int numClicks = 0;
    private JLabel label = new JLabel("Button clicks: " + numClicks);
    private JButton button = new JButton("Press me!");

    public SwingApplication(JPanel jp) {
        jp.add(label);
        jp.add(button);
        button.addActionListener(new Handler());
    }

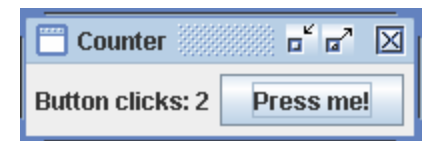
    private class Handler implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            numClicks++;
            label.setText("Button clicks: " + numClicks);
        }
    }
}
```



# Eventos

```
import javax.swing.*;
import java.awt.*;
public class Main {
    public static void main(String[] args) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("Counter");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel p = new JPanel();
        frame.setContentPane(p);
        SwingApplication app = new SwingApplication(p);
        frame.pack();
        frame.setVisible(true);
    }
}
```



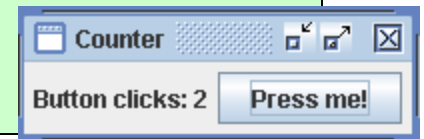
# Eventos

- Também é possível fazer:

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

public class SwingApplication {
    private int numClicks = 0;
    private JLabel label = new JLabel("Button clicks: " + numClicks);
    private JButton button = new JButton("Press me!");

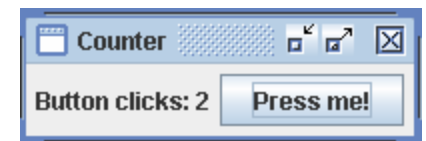
    public SwingApplication(JPanel jp) {
        jp.add(label);
        jp.add(button);
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                numClicks++;
                label.setText("Button clicks: " + numClicks);
            }
        });
    }
}
```



# Eventos

- No código em baixo é efectuada uma implementação do interface (também com funciona para extensão de classes) ActionListener.
- Utiliza-se uma **classe aninhada anónima** cujo corpo se define logo a seguir ao constructor do objecto da super classe. Como é anónima a classe apenas pode ser acedida neste ponto.

```
button.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        numClicks++;  
        label.setText("Button clicks: " + numClicks);  
    }  
});
```



# LookAndFeel

- Pode adicionar o seguinte código no início do método main.

```
try {  
  
    javax.swing.UIManager.setLookAndFeel("com.sun.java.  
swing.plaf.windows.WindowsLookAndFeel");  
  
} catch (ClassNotFoundException ex) {  
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
} catch (InstantiationException ex) {  
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
} catch (IllegalAccessException ex) {  
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
} catch (UnsupportedLookAndFeelException ex) {  
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
}
```

# Setting the lookAndFeel

- Para cross platform:

```
// cross-platform (also called "Metal")
UIManager.setLookAndFeel(
    UIManager.getCrossPlatformLookAndFeelClassName()
);
```

- Para o default do sistema:

```
UIManager.setLookAndFeel(
    UIManager.getSystemLookAndFeelClassName()
);
```

- *javaHomeDirectory*\lib file swing.properties:

```
# Swing properties
swing.defaultlaf=com.sun.java.swing.plaf.windows.WindowsLookAndFeel
```

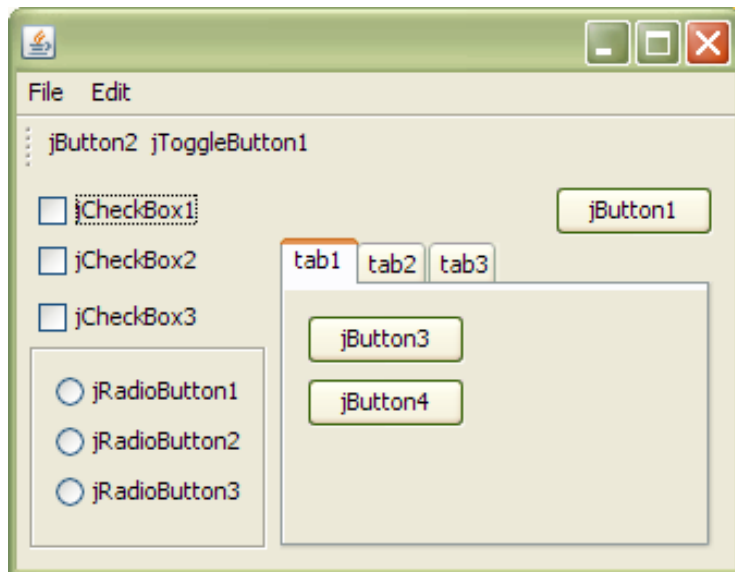
- ou alterar a system property.

```
System.setProperty("swing.defaultlaf",
    "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
```

# LookAndFeel

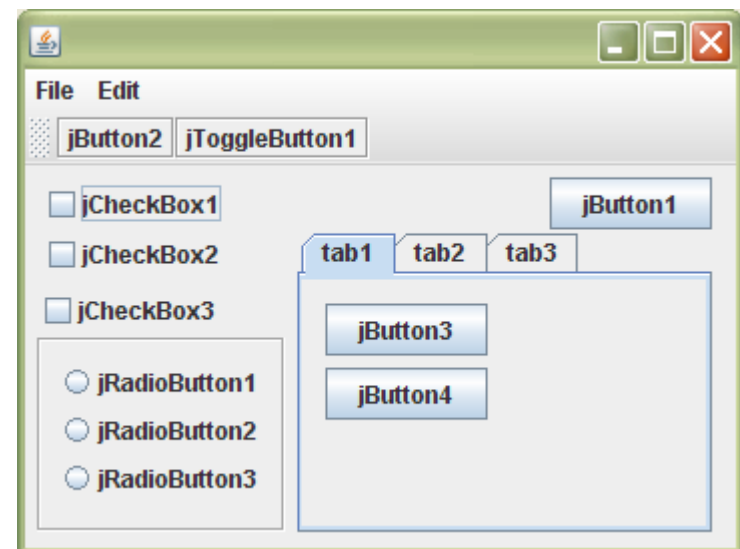
- Example of “look and feel”s in a windows computer
  - javax.swing.plaf.metal.MetalLookAndFeel
  - com.sun.java.swing.plaf.motif.MotifLookAndFeel
  - com.sun.java.swing.plaf.windows.WindowsLookAndFeel
  - com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel

windows



LEEC@IST

Metal (default)



Swing – 64/68



# System properties

- **Acessiveis através de:**
  - getProperty(String key, String def)
  - setProperty(String key, String value)
- java.version                      Java Runtime Environment version
- java.home                         Java installation directory
- java.class.path                  Java class path
- java.library.path                List of paths to search when loading libraries
- java.io.tmpdir                    Default temp file path
- java.compiler                    Name of JIT compiler to use
- java.ext.dirs                    Path of extension directory or directories
- **os.name**                         **Operating system name**
- os.version                        Operating system version
- file.separator                    File separator ("/" on UNIX)
- path.separator                    Path separator (":" on UNIX)
- line.separator                    Line separator ("\n" on UNIX)
- **user.name**                      **User's account name**
- **user.home**                      **User's home directory**
- **user.dir**                        **User's current working directory**

# Java Web Start

- Use a Java Network Launching Protocol, or “.JNLP” file:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for Calculator -->
<jnlp
  spec="1.0+"
  codebase="http://sips.inesc-id.pt/~pacl"
  href="calc.jnlp">
  <information>
    <title>Calculadora</title>
    <vendor>IST</vendor>
    <description kind="short">A simple calculator</description>
    <offline-allowed/>
  </information>
  <resources>
    <j2se version="1.6+"/>
    <jar href="calc.jar"/>
  </resources>
  <application-desc main-class="Calculadora"/>
</jnlp>
```

# Java Web Start

- Colocar o “java.jar” no servidor.
- Adicionar uma referência ao ficheiro “.JNLP” na página:

```
<body topmargin="30" leftmargin="30">
```

```
<h3>Java</h3>
```

```
    <P>Uma calculadora em Java que corre usando o Java Web  
    Start.</P>
```

```
    <P><A href="calc.jnlp">Calculadora</A></P>
```

- Garantir que o tipo MIME da extensão .jnlp está configurado no servidor. Por exemplo no Apache adicionar:
  - “application/x-java-jnlp-file JNLP” no ficheiro “mime.types”

# Java Beans

- Uma classe de java que obedece a um conjunto de normas para os nomes dos seus métodos....
- No Netbeans pode-se fazer *addFromJar* e adicionar um *bean* á palete de componente de forma a ser utilizado como um componente já existente.
- É utilizando *reflection* para determinar as propriedades, *bindings*, eventos que aparecem na janela de *properties*.