Programação de Aplicações Gráficas em Java

Rafael Santos

Tópicos

- Brevíssima introdução à linguagem Java
  - Orientação a objetos, criando classes
- Criando aplicações gráficas em Java
  - Janelas, componentes existentes, eventos.
- Escrevendo componentes específicos
- Exemplos

Tópicos

- Veremos...
  - Alguns exemplos práticos com código que funciona.
  - Algumas implementações de conceitos simples que podem ser facilmente expandidos.
- Não veremos...
  - Detalhes sobre a linguagem, componentes ou *layouts* muito complexos, coleções, entrada e saída, JSP+Servlets...
  - Veja http://www.lac.inpe.br/~rafael.santos para apresentações sobre outros tópicos!
  - Perguntas são bem-vindas!
- Estou assumindo um conhecimento básico de Java!

Brevíssima introdução à linguagem Java

Linguagem Java

- Simples, orientada a objetos.
- Herdou muitos conceitos de C, C++, outras.
- Código compilado para *bytecodes*, interpretado por uma máquina virtual.
  - *Bytecodes* compatíveis entre sistemas operacionais*.
  - Base compatível entre máquinas virtuais.
  - APIs dependem da finalidade, mas código de negócio é portátil*!
- Otimização de *bytecodes* melhora a performance.

Princípios básicos de orientação a objetos

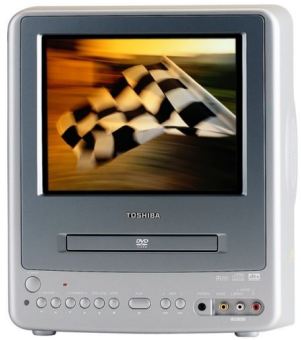
- **Encapsulamento**
- Atributos e funções relevantes a um domínio ou problema são *encapsulados* em uma classe de forma que:
  - Somente atributos e funções relevantes sejam representados;
  - Uma *interface para acesso* seja criada para que usuários/desenvolvedores tenham acesso somente a funções e atributos que podem ser acessados diretamente.

Princípios básicos de orientação a objetos

- **Herança**
- Uma classe pode ser descrita de forma incremental usando classes já existentes.

Princípios básicos de orientação a objetos

- **Polimorfismo**
- Classes que tem relação de herança podem ser processadas de forma igual e transparente (com algumas condições).
- Relação *é-um-tipo-de*.



Princípios básicos de orientação a objetos

- **Interfaces**
  - Contratos de implementação de funções.
  - Se a interface contém uma função X:
    - Na interface esta função é somente declarada.
    - Na classe ela deve ser escrita de forma completa.
  - Classes que implementam uma mesma interface mantém relação *é-um-tipo-de*.
  - Classes podem implementar mais de uma interface.
- **Classes Abstratas**
  - Similares à interfaces, mas com restrições na herança.
  - Podem ter métodos abstratos, atributos e métodos que serão herdados.

Princípios básicos de orientação a objetos

- **Classes** são descritas em Java para representar modelos ou conceitos.
- **Objetos** ou **Instâncias** são criados ou instanciados a partir das classes.
  - Criação pela palavra-chave `new`.
  - **Referências** são usadas para acesso aos objetos.
- Uma classe pode ser usada para criar muitos objetos.
  - Os atributos de cada objeto serão independentes...
    - ... a não ser que tenham sido declarados como estáticos (compartilhados).
- Alguns métodos são “mágicos” (ex. construtores, `toString`)

Princípios básicos de orientação a objetos

- Classes podem ser organizadas em pacotes.
- Existem regras de visibilidade que indicam que atributos e métodos são visíveis...
  - Entre classes do mesmo pacote.
  - Entre classes de diferentes pacotes.
- Dentro de uma classe, todos os métodos e atributos são visíveis.
- Classes podem, evidentemente, usar instâncias de outras classes (e até dela mesma!)

Programando em Java

- **Tudo** são classes (exceto atributos de tipos nativos).
- **Nada** é declarado independente (fora) de uma classe.
- Estrutura geral:
  1. Declaração de pacotes;
  2. Importação de classes externas;
  3. Declaração da classe e atributos (ex. extensão).
  4. Declaração de campos e métodos.
- Boas práticas:
  - Cada classe em um arquivo com nome igual e extensão .java;
  - Atributos declarados antes dos métodos;
  - Indentação e comentários!

Um exemplo de classe

```
package applets;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;

public class Circulo
  {
  protected Color cor;
  protected int x,y;
  protected int raio;

  public Circulo(Color c,int x,int y,int r)
    {
    cor = c;
    this.x = x; this.y = y; raio = r;
    }

  public void draw(Graphics2D g)
    {
    g.setColor(cor);
    Ellipse2D.Float circ = new Ellipse2D.Float(x-raio,y-raio,raio*2,raio*2);
    g.draw(circ);
    }

  }
```

Um exemplo de classe

- No exemplo anterior vimos...
  - Declaração de pacote.
  - Importação de classes externas necessárias.
  - Declaração da classe.
  - Atributos e classe com declarações de visibilidade.
  - Construtor.
  - Métodos.

Um exemplo de classe com herança

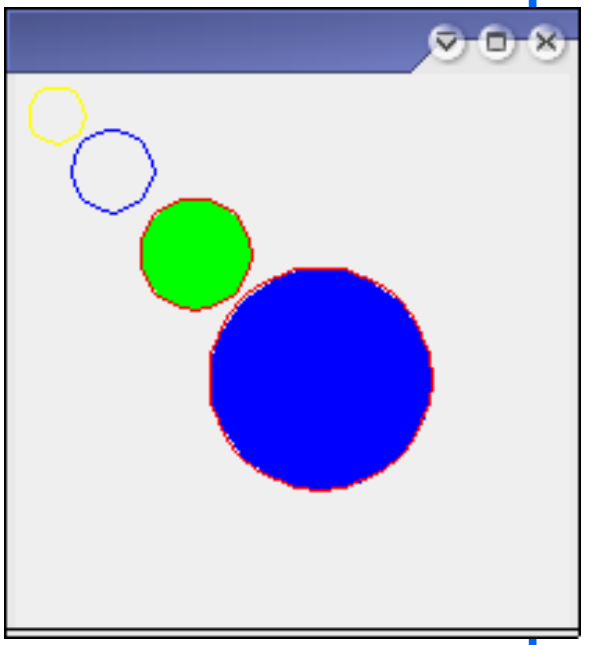
```
package applets;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;

public class CirculoPreenchido extends Circulo
  {
  protected Color corP;

  public CirculoPreenchido(Color c,Color p,int x,int y,int r)
    {
    super(c,x,y,r);
    corP = p;
    }

  public void draw(Graphics2D g)
    {
    g.setColor(corP);
    Ellipse2D.Float circ = new Ellipse2D.Float(x-raio,y-raio,raio*2,raio*2);
    g.fill(circ);
    super.draw(g);
    }

  }
```

Um exemplo de classe com herança

- No exemplo anterior vimos...
  - Declaração de classe que herda de outra.
  - Execução de construtor ancestral (*super*).
  - Acesso a campos herdados (diretamente).
  - Acesso a métodos herdados (*super*).
  - Sobrecarga de métodos (métodos com mesmo nome nas duas classes).

Exemplo mais complexo

```
package applets;

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

public class TestaCirculo extends JComponent
  {

  public Dimension getPreferredSize()
    {
    return new Dimension(200,200);
    }

  public void paintComponent(Graphics g)
    {
    Graphics2D g2d = (Graphics2D)g;
    Circulo x = new Circulo(Color.YELLOW,15,15,10); x.draw(g2d);
    new Circulo(Color.BLUE,35,35,15).draw(g2d);
    CirculoPreenchido y = new CirculoPreenchido(Color.RED,Color.GREEN,65,65,20); y.draw(g2d);
    Circulo z = new CirculoPreenchido(Color.RED,Color.BLUE,110,110,40); z.draw(g2d);
    }

  public static void main(String[] args)
    {
    JFrame f = new JFrame();
    f.add(new TestaCirculo());
    f.setVisible(true);
    f.pack();
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
  }
```



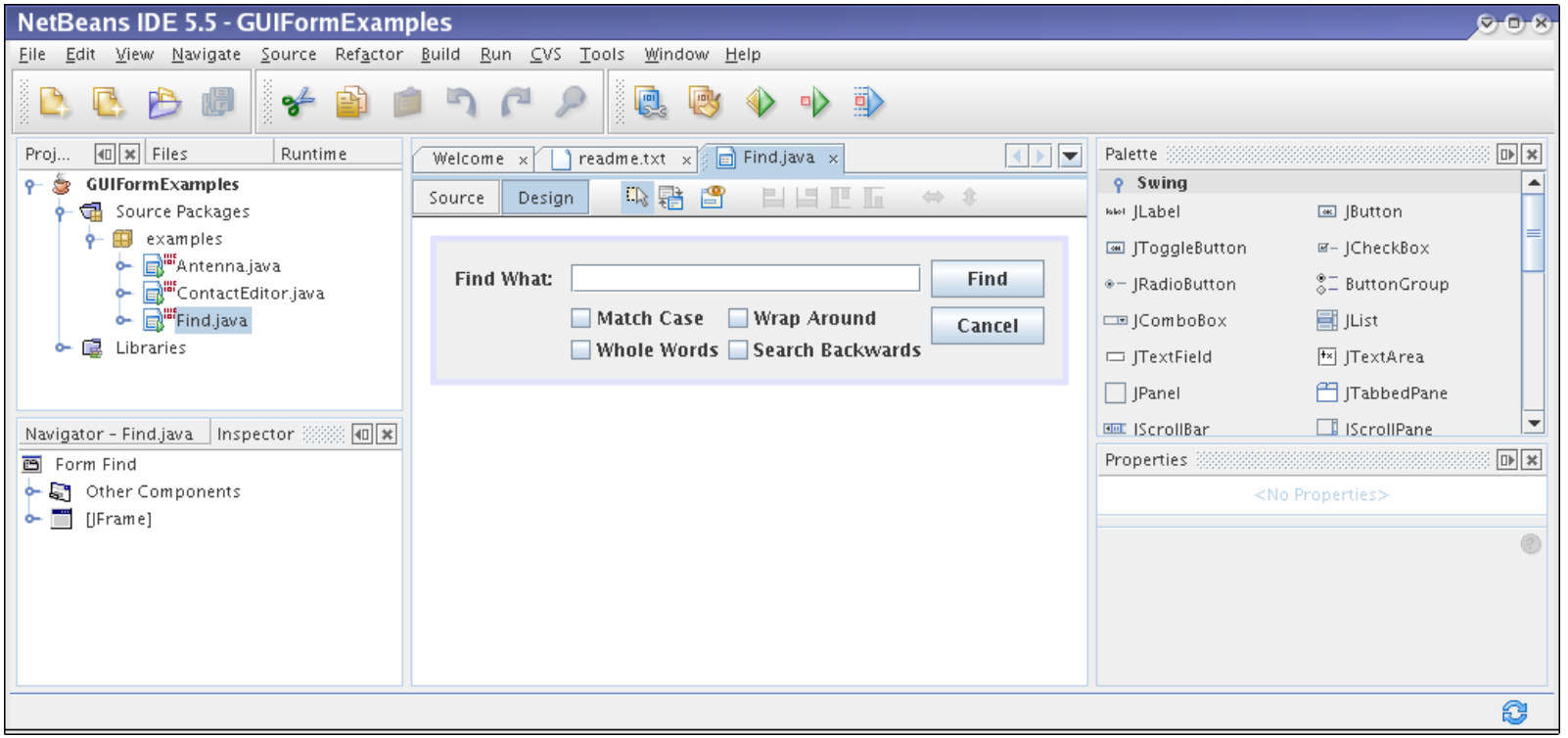
Exemplo mais complexo

- No exemplo anterior vimos...
  - Mais herança.
  - Métodos “mágicos” (`getPreferredSize`, `paintComponent`) sobreescritos.
  - Método “mágico” `main`.
  - Instâncias anônimas!
  - Polimorfismo (exemplo bem simples).

Criando aplicações gráficas em Java

Como criar aplicações gráficas em Java

- A maioria das IDEs permite a criação de interfaces gráficas (aplicações, *applets*, diálogos) por composição visual.
  - Realmente útil para *layouts* complexos.
- É **importante** saber como código relacionado funciona para compreender o que a IDE escreve!



Criando uma janela gráfica (*Frame*)

- Simples! Criamos uma classe que herda de `JFrame`.
  - O construtor pode ser usado para montar a interface gráfica.
  - A própria classe pode ter um método `main` que simplesmente cria uma instância dela mesma.
- Vantagens do uso do mecanismo de herança:
  - Vários métodos de `JFrames` podem ser executados pela nossa classe.
  - Podemos sobreescrever métodos com comportamento específico.

Criando uma janela gráfica (*Frame*)

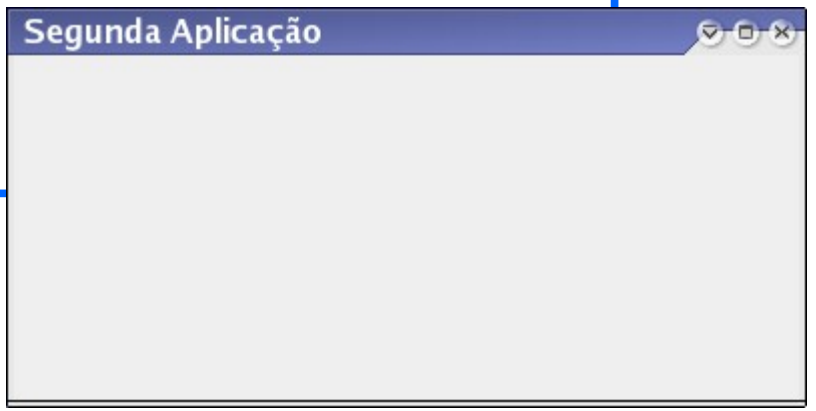
```
import javax.swing.JFrame;

public class PrimeiraAplicacao extends JFrame
  {

  public PrimeiraAplicacao()
    {
    super("Primeira Aplicação");
    }

  public static void main(String[] args)
    {
    new PrimeiraAplicacao();
    }
  }
```

Criando uma janela gráfica (*Frame*)

- Ao executar o código, nada aparece!
- Faltou executar métodos que definem aparência e comportamento básico da aplicação!
- Reveja o código. O que você acha que esta aplicação faz?

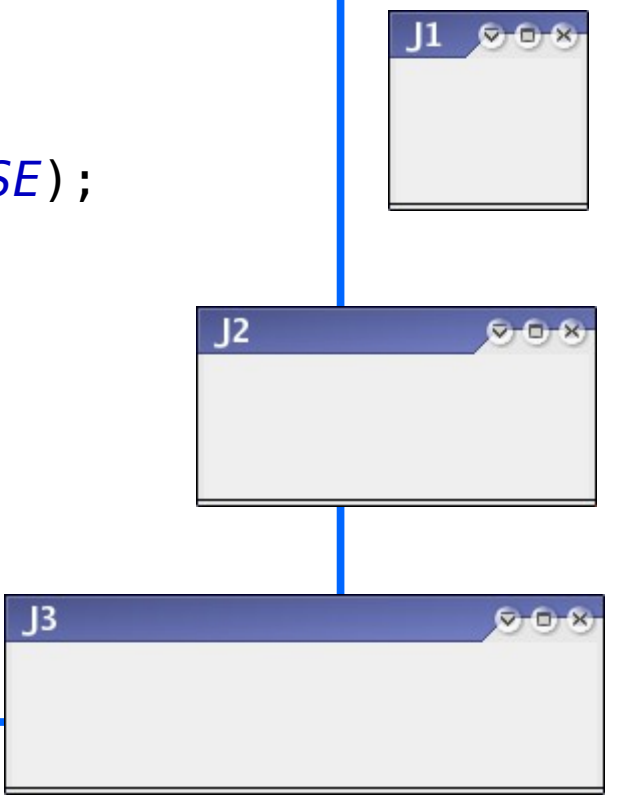
Criando uma janela gráfica (*Frame*)

```
import javax.swing.JFrame;

public class SegundaAplicacao extends JFrame
  {

  public SegundaAplicacao()
    {
    super("Segunda Aplicação");
    setSize(400,200);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new SegundaAplicacao();
    }
  }
```



Criando uma janela gráfica (*Frame*)

```
import javax.swing.JFrame;

public class TerceiraAplicacao extends JFrame
  {

  public TerceiraAplicacao(String t,int l,int a)
    {
    super(t);
    setSize(l,a);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new TerceiraAplicacao("J1",100,100);
    new TerceiraAplicacao("J2",200,100);
    new TerceiraAplicacao("J3",300,100);
    }
  }
```



Criando uma janela gráfica (*Frame*) 2

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

public class AplicacaoComComponente extends JFrame
  {

  public AplicacaoComComponente(String t,int l,int a)
    {
    super(t);
    getContentPane().add(new JLabel("Mensagem de boas-vindas"));
    setSize(l,a);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new AplicacaoComComponente("Mensagem",100,100);
    }
  }
```

Criando uma janela gráfica (*Frame*) 2

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

public class AplicacaoComComponente2 extends JFrame
  {

  public AplicacaoComComponente2(String t)
    {
    super(t);
    getContentPane().add(new JLabel("Mensagem de boas-vindas"));
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new AplicacaoComComponente2("Mensagem");
    }
  }
```

Usando componentes

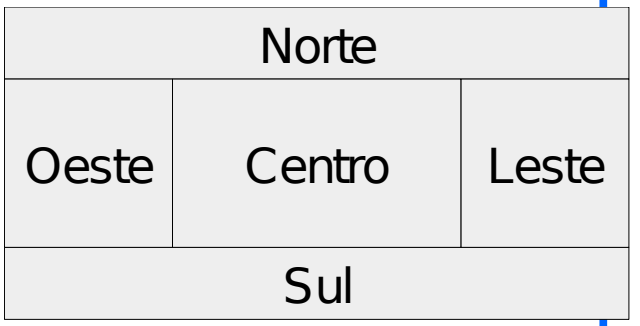
- Existem vários tipos de componentes de interfaces com usuários
  - Customizáveis por chamada a métodos ou herança.
- Para criar aplicações com interfaces gráficas:
  - Criamos instâncias das classes dos componentes e as adicionamos ao `JFrame`.
  - Modificamos atributos destas instâncias.
  - Adicionamos estas instâncias à interface gráfica.
  - Registramos **eventos** que indicam o que fazer se houver interação com o componente.

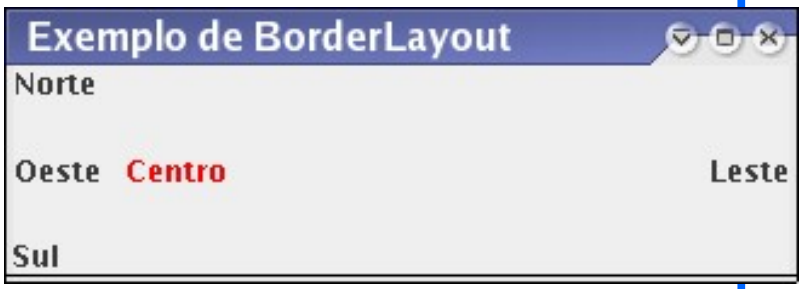
Usando componentes

- Como os componentes serão arranjados e exibidos na interface gráfica?
  - *Layouts*.
- Existem vários *layouts* simples, implementados como classes.
  - Podemos implementar *layouts* diferente ou não usar nenhum!
- Para usar um *layout*, executamos um método para indicar o *layout* do painel de conteúdo da aplicação.
- O *layout* também indica como os componentes serão rearranjados se o tamanho da janela mudar.

Alguns *layouts* gráficos: *BorderLayout*

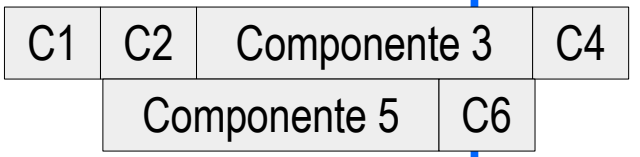
```
import java.awt.*;
import javax.swing.*;

public class ExBorderLayout extends JFrame
  {
  public ExBorderLayout()
    {
    super("Exemplo de BorderLayout");
    Container c = getContentPane();
    c.setLayout(new BorderLayout());
    c.add(new JLabel("Norte"),BorderLayout.NORTH);
    c.add(new JLabel("Sul"),BorderLayout.SOUTH);
    c.add(new JLabel("Leste"),BorderLayout.EAST);
    c.add(new JLabel("Oeste"),BorderLayout.WEST);
    JLabel centro = new JLabel("Centro");
    centro.setForeground(Color.RED);
    centro.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
    c.add(centro,BorderLayout.CENTER);
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new ExBorderLayout();
    }
  }
```

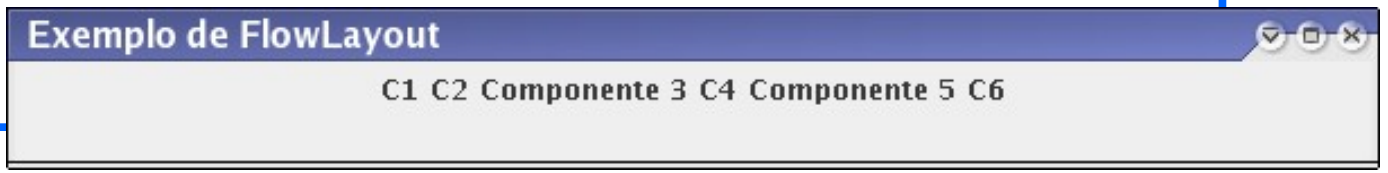




Alguns *layouts* gráficos: *FlowLayout*

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

public class ExFlowLayout extends JFrame
  {

  public ExFlowLayout()
    {
    super("Exemplo de FlowLayout");
    Container c = getContentPane();
    c.setLayout(new FlowLayout());
    c.add(new JLabel("C1"));
    c.add(new JLabel("C2"));
    c.add(new JLabel("Componente 3"));
    c.add(new JLabel("C4"));
    c.add(new JLabel("Componente 5"));
    c.add(new JLabel("C6"));
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new ExFlowLayout();
    }
  }
```

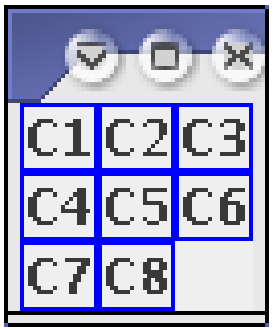


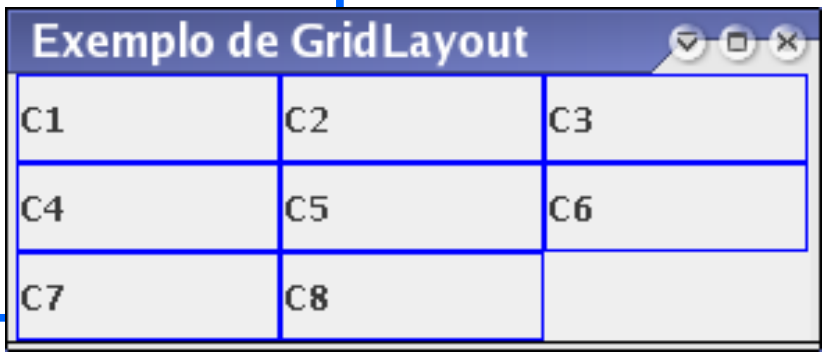


Alguns *layouts* gráficos: *GridLayout*

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

public class ExGridLayout extends JFrame
  {

  public ExGridLayout()
    {
    super("Exemplo de GridLayout");
    Container c =  getContentPane();
    c.setLayout(new GridLayout(3,3));
    for(int i=1;i<=8;i++)
      {
      JLabel l = new JLabel("C"+i);
      l.setBorder(BorderFactory.createLineBorder(Color.BLUE));
      c.add(l);
      }
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new ExGridLayout();
    }
  }
```

Comp 1	Comp 2	Comp 3
Comp 4	Comp 5	Comp 6
Comp 7	Comp 8	





Outros *layouts* gráficos

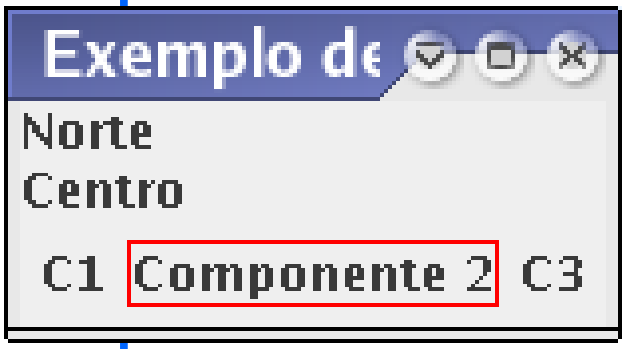
- *BoxLayout*: permite arranjar componentes em uma única linha ou coluna.
- *CardLayout*: permite empilhar conjuntos de componentes na direção Z.
- *GridBagLayout*: permite arranjo de componentes com proporções diferentes.
- *SpringLayout*: permite arranjar componentes relativamente uns aos outros.

Layouts podem ser combinados!

- Podemos adicionar uma instância de `JPanel` como um componente.
- Esta instância de `JPanel` pode ter seu próprio *layout* e ter outros componentes.
- Múltiplas combinações permitem alta flexibilidade mas com complexidade de código.

Layouts podem ser combinados!

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

public class ExLayoutMisto extends JFrame
  {
  public ExLayoutMisto()
    {
    super("Exemplo de Layout Misto");
    Container c = getContentPane();
    c.setLayout(new BorderLayout());
    c.add(new JLabel("Norte"),BorderLayout.NORTH);
    c.add(new JLabel("Centro"),BorderLayout.CENTER);
    JPanel painel = new JPanel(new FlowLayout());
    painel.add(new JLabel("C1"));
    JLabel c2 = new JLabel("Componente 2");
    c2.setBorder(BorderFactory.createLineBorder(Color.RED));
    painel.add(c2);
    painel.add(new JLabel("C3"));
    c.add(painel,BorderLayout.SOUTH);
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new ExLayoutMisto();
    }
  }
```

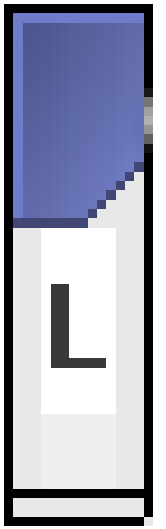


Layout nulo

- Podemos simplesmente não usar *layouts*: `setLayout(null)`.
- Devemos posicionar cada componente manualmente, com coordenadas em pixels.
- Métodos úteis:
  - Componentes: `getPreferredSize()` e `setBounds()`
  - Painel: `getInsets()`

Layout nulo

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

public class ExLayoutNulo extends JFrame
  {
  public ExLayoutNulo()
    {
    super("Exemplo de Layout Nulo");
    Container c = getContentPane();
    c.setLayout(null);
    JLabel l1 = new JLabel("Label 1");
    JLabel l2 = new JLabel("Label 2");
    JLabel l3 = new JLabel("Label 3");
    l1.setBackground(Color.WHITE); l1.setOpaque(true);
    l1.setBounds(0,0,100,20);
    l2.setBackground(Color.YELLOW); l2.setOpaque(true);
    l2.setBounds(20,20,100,20);
    l3.setBackground(Color.GREEN); l3.setOpaque(true);
    l3.setBounds(40,20,100,20);
    c.add(l1);     c.add(l2);     c.add(l3);
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new ExLayoutNulo();
    }
  }
```



Layout nulo

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

public class ExLayoutNulo extends JFrame
  {
  public ExLayoutNulo()
    {
    super("Exemplo de Layout Nulo");
    Container c = getContentPane();
    c.setLayout(null);
    JLabel l1 = new JLabel("Label 1");
    JLabel l2 = new JLabel("Label 2");
    JLabel l3 = new JLabel("Label 3");
    l1.setBackground(Color.WHITE); l1.setOpaque(true);
    l1.setBounds(0,0,100,20);
    l2.setBackground(Color.YELLOW); l2.setOpaque(true);
    l2.setBounds(20,20,100,20);
    l3.setBackground(Color.GREEN); l3.setOpaque(true);
    l3.setBounds(40,20,100,20);
    c.add(l1);     c.add(l2);     c.add(l3);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(140,40);
    }

  public static void main(String[] args)
    {
    new ExLayoutNulo();
    }
  }
```

Layout nulo

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

public class ExLayoutNulo extends JFrame
  {
  public ExLayoutNulo()
    {
    super("Exemplo de Layout Nulo");
    Container c = getContentPane();
    c.setLayout(null);
    JLabel l1 = new JLabel("Label 1");
    JLabel l2 = new JLabel("Label 2");
    JLabel l3 = new JLabel("Label 3");
    l1.setBackground(Color.WHITE); l1.setOpaque(true);
    l1.setBounds(0,0,100,20);
    l2.setBackground(Color.YELLOW); l2.setOpaque(true);
    l2.setBounds(20,20,100,20);
    l3.setBackground(Color.GREEN); l3.setOpaque(true);
    l3.setBounds(40,20,100,20);
    c.add(l1);     c.add(l2);     c.add(l3);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Insets i = getInsets();
    setSize(140+i.left+i.right,40+i.bottom+i.top);
    }
  public static void main(String[] args)
    {
    new ExLayoutNulo();
    }
  }
```

Outros componentes

- Vimos `JLabels`.
- Controles simples como `JButton`, `JComboBox`, `JList`, `JMenu`, `JSlider`, `JSpinner`, `JTextField`, etc.
- Controles complexos como `JColorChooser`, `JFileChooser`, `JTable`, `JTree`.
- *Containers top-level* como `JApplet`, `JDialog`, JFrame.
- Outros containers como `JPanel`, `JScrollPane`, `JSplitPane`, `JTabbedPane`, `JInternalFrame`.
- Não dá para ver exemplos de uso de todos...

Componentes e Eventos

- Componentes são realmente úteis para interação com usuário.
- Quando alguma interação for feita com um componente, devemos executar parte do código.
- Problema com código procedural: interações podem ocorrer a qualquer momento!
- Solução: uso de **eventos**.
  - Criamos os componentes, registramos eventos que podem ocorrer, criamos métodos para processar estes eventos.
  - Existem vários tipos de eventos...

Componentes e eventos: JButton

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

public class ExJButton1 extends JFrame
  {
  private JButton j1,j2;
  private int contador = 0;
  private JLabel lContador;

  public ExJButton1()
    {
    super("Exemplo de JButtons e Eventos");
    Container c = getContentPane();
    c.setLayout(new GridLayout(1,3));
    j1 = new JButton("Adiciona");
    j2 = new JButton("Subtrai");
    lContador = new JLabel(""+contador);
    c.add(j1); c.add(j2); c.add(lContador);
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new ExJButton1();
    }
  }
```

Componentes e eventos: JButton

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ExJButton2 extends JFrame implements ActionListener
  {
  private JButton j1,j2;
  private int contador = 0;
  private JLabel lContador;
  public ExJButton2()
    {
    super("Exemplo de JButtons e Eventos");
    Container c = getContentPane();
    c.setLayout(new GridLayout(1,3));
    j1 = new JButton("Adiciona");
    j2 = new JButton("Subtrai");
    lContador = new JLabel(""+contador);
    c.add(j1); c.add(j2); c.add(lContador);
    j1.addActionListener(this); j2.addActionListener(this);
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
  public void actionPerformed(ActionEvent e)
    {
    if (e.getSource() == j1) contador++;
    else if (e.getSource() == j2) contador--;
    lContador.setText(""+contador);
    }
  public static void main(String[] args) { new ExJButton2(); }
  }
```

Componentes e eventos: JButton e JTextArea

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

public class ExTextField extends JFrame implements ActionListener
  {
  private JTextField valor;
  private JButton calcula;
  private JLabel resultado;

  public ExTextField()
    {
    super("Exemplo de JTextFields e Eventos");
    Container c = getContentPane();
    c.setLayout(new FlowLayout());
    c.add(new JLabel("Fatorial de "));
    valor = new JTextField("    1");
    valor.addActionListener(this);
    c.add(valor);
    calcula = new JButton("=");
    calcula.addActionListener(this);
    c.add(calcula);
    resultado = new JLabel("    1");
    c.add(resultado);
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

Componentes e eventos: JButton e JTextArea

```
public void actionPerformed(ActionEvent e)
  {
  int val = Integer.parseInt(valor.getText().trim());
  double fat = 1;
  for(int v=1;v<=val;v++) fat *= v;
  resultado.setText(""+fat);
  }

public static void main(String[] args)
  {
  new ExTextField();
  }

}
```

Componentes e eventos: JRadioButton

- Botões que podem ser combinados em um `ButtonGroup`.
- Somente um botão pode ser selecionado.

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

public class ExJRadioButton extends JFrame implements ActionListener
  {
  private String[] imagens = {"eclipse01.png","eclipse02.png","eclipse03.png",
                              "eclipse04.png","eclipse05.png","eclipse06.png",
                              "eclipse07.png","eclipse08.png","eclipse09.png",};
  private JLabel imagem;
```

Componentes e eventos: JRadioButton

```
public ExJRadioButton()
  {
  super("Exemplo de JRadioButtons");
  // Painel com os botões
  JPanel painel = new JPanel(new GridLayout(3,3));
  ButtonGroup grupo = new ButtonGroup();
  JRadioButton[] botões = new JRadioButton[9];
  for(int b=0;b<9;b++)
    {
    botões[b] = new JRadioButton(imagens[b]);
    botões[b].addActionListener(this);
    grupo.add(botões[b]);
    painel.add(botões[b]);
    }
  // UI
  Container c = getContentPane();
  c.add(painel,BorderLayout.SOUTH);
  imagem = new JLabel();
  imagem.setPreferredSize(new Dimension(100,100));
  imagem.setHorizontalAlignment(SwingConstants.CENTER);
  c.add(imagem,BorderLayout.CENTER);
  pack();
  setVisible(true);
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
```

Componentes e eventos: `JRadioButton`

```java
public void actionPerformed(ActionEvent e)
  {
  JRadioButton rb = (JRadioButton)e.getSource();
  ImageIcon ícone = new ImageIcon(rb.getActionCommand());
  imagem.setIcon(ícone);
  }

public static void main(String[] args)
  {
  new ExJRadioButton();
  }
}
```

Componentes e eventos: JSlider

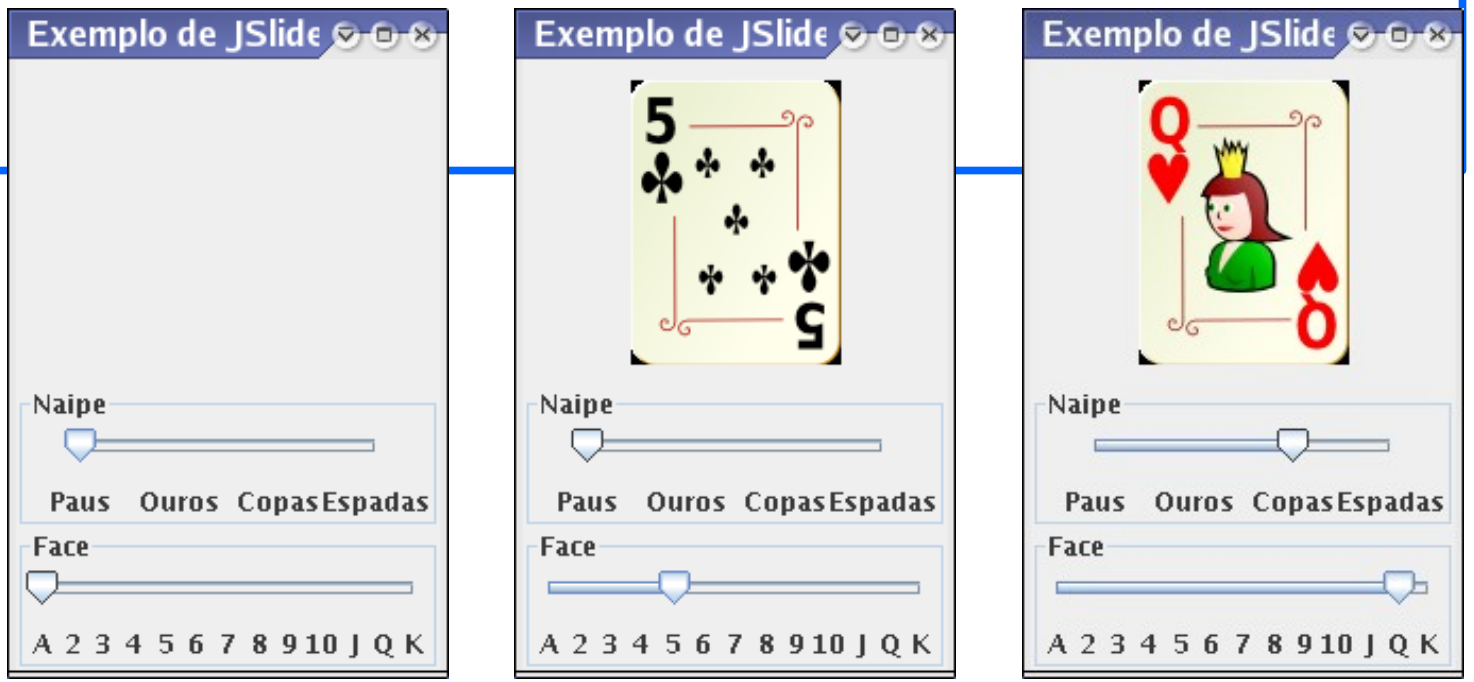
```
import java.awt.*;
import java.util.Hashtable;
import javax.swing.*;
import javax.swing.event.*;

public class ExSliders extends JFrame implements ChangeListener
  {
  private JSlider naipe,face;
  private JLabel carta;

  public ExSliders()
    {
    super("Exemplo de JSliders");
    Container c = getContentPane();
    criaSliderNaipes();    criaSliderFaces();
    JPanel controle = new JPanel(new GridLayout(2,1));
    controle.add(naipe); controle.add(face);
    carta = new JLabel();
    carta.setPreferredSize(new Dimension(99+20,134+20));
    carta.setHorizontalAlignment(SwingConstants.CENTER);
    c.add(controle,BorderLayout.SOUTH);
    c.add(carta,BorderLayout.CENTER);
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

Componentes e eventos: JSlider

```
private void criaSliderNaipes()
  {
  naipe = new JSlider(0,3,0);
  Hashtable<Integer,JLabel> labels = new Hashtable<Integer,JLabel>();
  labels.put(new Integer(0),new JLabel("Paus"));
  labels.put(new Integer(1),new JLabel("Ouros"));
  labels.put(new Integer(2),new JLabel("Copas"));
  labels.put(new Integer(3),new JLabel("Espadas"));
  naipe.setLabelTable(labels);
  naipe.setPaintLabels(true); naipe.setPaintTicks(true); naipe.setSnapToTicks(true);
  naipe.setBorder(BorderFactory.createTitledBorder("Naipe"));
  naipe.addChangeListener(this);
  }

private void criaSliderFaces()
  {
  face = new JSlider(0,12,0);
  Hashtable<Integer,JLabel> labels = new Hashtable<Integer,JLabel>();
  for(int l=2;l<11;l++) labels.put(new Integer(l-1),new JLabel(""+l));
  labels.put(new Integer(0),new JLabel("A"));
  labels.put(new Integer(10),new JLabel("J"));
  labels.put(new Integer(11),new JLabel("Q"));
  labels.put(new Integer(12),new JLabel("K"));
  face.setLabelTable(labels);
  face.setPaintLabels(true); face.setPaintTicks(true); face.setSnapToTicks(true);
  face.setBorder(BorderFactory.createTitledBorder("Face"));
  face.addChangeListener(this);
  }
```

Componentes e eventos: JSlider

```
public void stateChanged(ChangeEvent e)
  {
  String nome = String.format("%d-%02d.png",
                              new Object[]{naipe.getValue()+1,face.getValue()+1});
  ImageIcon ícone = new ImageIcon(nome);
  carta.setIcon(ícone);
  }

public static void main(String[] args)
  {
  new ExSliders();
  }

}
```



Formato dos nomes dos arquivos das cartas é {0,1,2,3}-{00,01,02..12}.png

Componentes e eventos: JSpinner

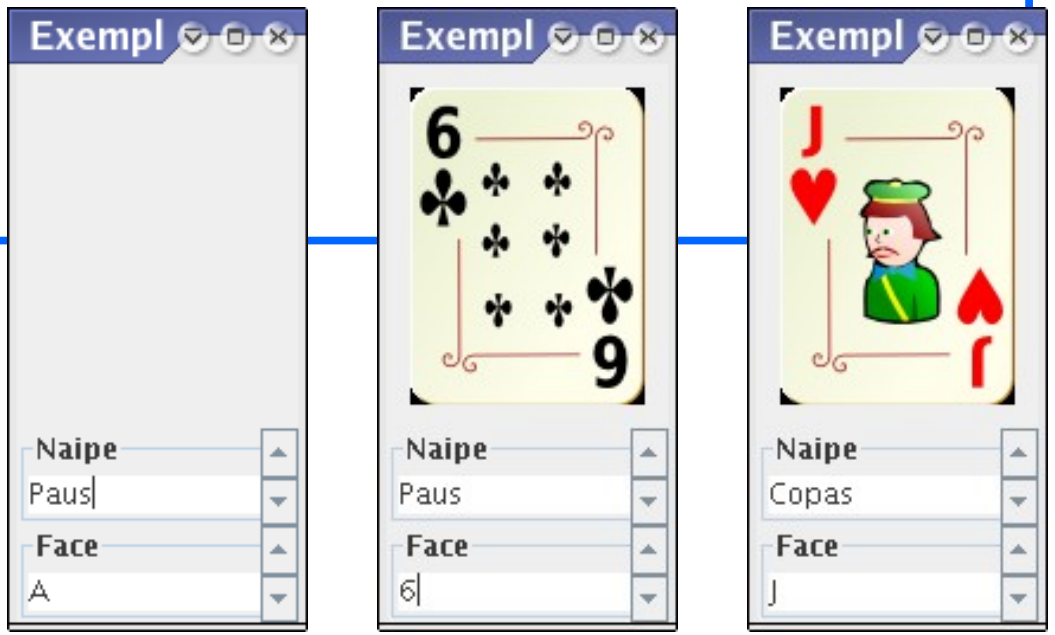
```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

public class ExSpinners extends JFrame implements ChangeListener
  {
  private JSpinner naipe,face;
  private JLabel carta;

  public ExSpinners()
    {
    super("Exemplo de JSpinners");
    Container c = getContentPane();
    criaSpinnerNaipes();
    criaSpinnerFaces();
    JPanel controle = new JPanel(new GridLayout(2,1));
    controle.add(naipe); controle.add(face);
    carta = new JLabel();
    carta.setPreferredSize(new Dimension(99+20,134+20));
    carta.setHorizontalAlignment(SwingConstants.CENTER);
    c.add(controle,BorderLayout.SOUTH);
    c.add(carta,BorderLayout.CENTER);
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

Componentes e eventos: JSpinner

```
private void criaSpinnerNaipes()
  {
  String[] naipes = {"Paus","Ouros","Copas","Espadas"};
  SpinnerListModel modelo = new SpinnerListModel(naipes);
  naipe = new JSpinner(modelo);
  naipe.setBorder(BorderFactory.createTitledBorder("Naipe"));
  naipe.addChangeListener(this);
  }

private void criaSpinnerFaces()
  {
  String[] faces = {"A","2","3","4","5","6","7","8","9","10","J","Q","K"};
  SpinnerListModel modelo = new SpinnerListModel(faces);
  face = new JSpinner(modelo);
  face.setBorder(BorderFactory.createTitledBorder("Face"));
  face.addChangeListener(this);
  }
```

Componentes e eventos: JSpinner

```
public void stateChanged(ChangeEvent e)
  {
  int iNaipe = -1;
  char sNaipe = naipe.getValue().toString().charAt(0);
  iNaipe = "POCE".indexOf(sNaipe);
  int iFace = -1;
  char sFace = face.getValue().toString().charAt(0);
  iFace = "A23456789JQK".indexOf(sFace);
  String nome = String.format("%d-%02d.png",new Object[]{iNaipe+1,iFace+1});
  ImageIcon ícone = new ImageIcon(nome);
  carta.setIcon(ícone);
  }

public static void main(String[] args)
  {
  new ExSpinners();
  }

}
```

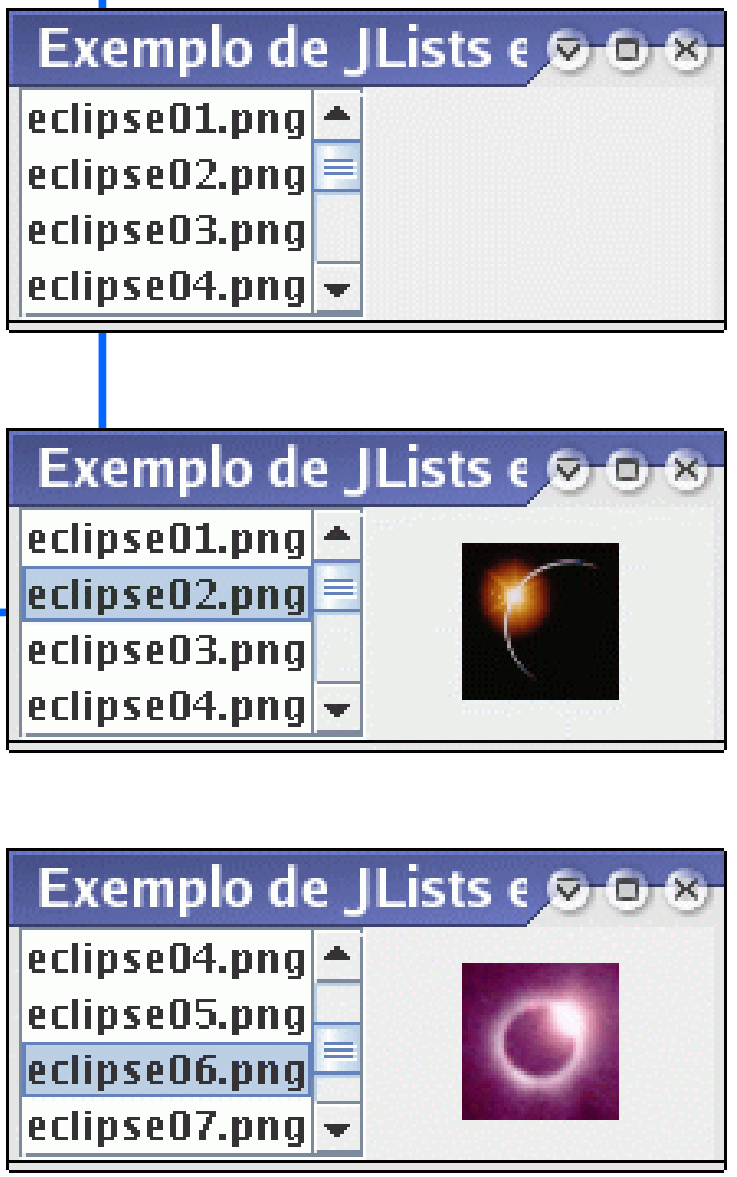


Componentes e eventos: JList

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

public class ExList extends JFrame implements ListSelectionListener
  {
  private String[] imagens = {"eclipse01.png","eclipse02.png","eclipse03.png",
                              "eclipse04.png","eclipse05.png","eclipse06.png",
                              "eclipse07.png","eclipse08.png","eclipse09.png"};
  private JList lista;
  private JLabel imagem;
  public ExList()
    {
    super("Exemplo de JLists e Eventos");
    Container c = getContentPane();       c.setLayout(new GridLayout(1,2));
    lista = new JList(imagens);
    lista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    lista.setLayoutOrientation(JList.VERTICAL);
    lista.setVisibleRowCount(4);
    JScrollPane painelParaLista = new JScrollPane(lista);
    lista.addListSelectionListener(this);
    imagem = new JLabel();
    imagem.setHorizontalAlignment(SwingConstants.CENTER);
    c.add(painelParaLista); c.add(imagem);
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

Componentes e eventos: JList

```
public void valueChanged(ListSelectionEvent e)
  {
  if (e.getValueIsAdjusting()) return;
  int qual = lista.getSelectedIndex();
  ImageIcon ícone = new ImageIcon(imagens[qual]);
  imagem.setIcon(ícone);
  }

public static void main(String[] args)
  {
  new ExList();
  }

}
```



Componentes e eventos: JMenu

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

public class ExFileChooser extends JFrame implements ActionListener
  {
  private JMenuItem abrePNG,abreJPG,abreGIF;
  private JLabel imagem;
  private JScrollPane scrollImagem;
  public ExFileChooser()
    {
    super("Exemplo de JMenus e JFileChooser");
    Container c = getContentPane();
    JMenuBar menuBar = new JMenuBar();
    // Criamos o menu "Abre"...
    JMenu menuAbre = new JMenu("Abre");
    abrePNG = new JMenuItem("Abre PNG"); menuAbre.add(abrePNG); abrePNG.addActionListener(this);
    abreJPG = new JMenuItem("Abre JPG"); menuAbre.add(abreJPG); abreJPG.addActionListener(this);
    abreGIF = new JMenuItem("Abre GIF"); menuAbre.add(abreGIF); abreGIF.addActionListener(this);
    menuBar.add(menuAbre);
    setJMenuBar(menuBar);
    imagem = new JLabel();  imagem.setHorizontalAlignment(SwingConstants.CENTER);
    scrollImagem = new JScrollPane(imagem,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                                          JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
    scrollImagem.setPreferredSize(new Dimension(300,300));
    c.add(scrollImagem);
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

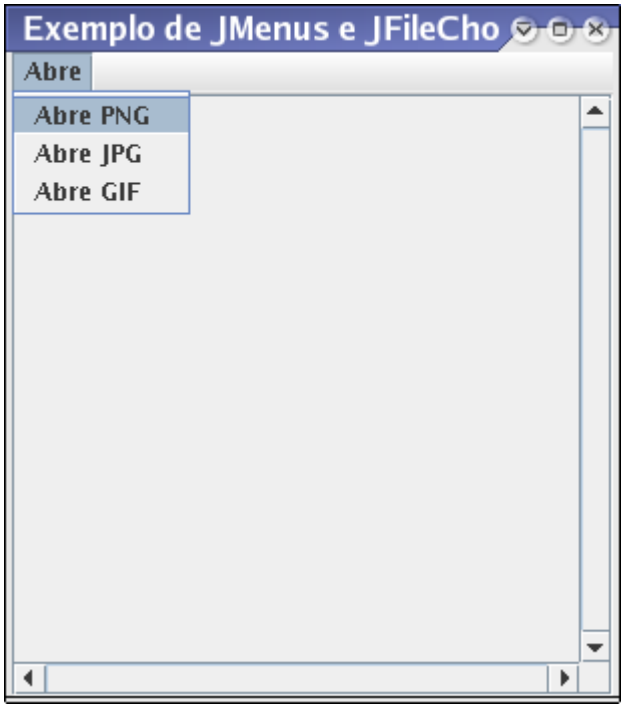
Componentes e eventos: JMenu

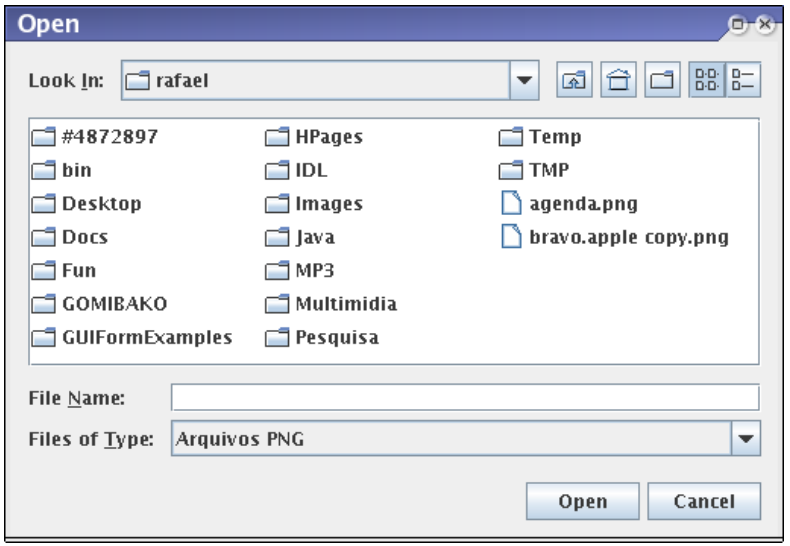
```
public void actionPerformed(ActionEvent e)
  {
  JFileChooser fc = new JFileChooser();
  if (e.getSource() == abrePNG) fc.setFileFilter(new FiltroPNG());
  if (e.getSource() == abreJPG) fc.setFileFilter(new FiltroJPG());
  if (e.getSource() == abreGIF) fc.setFileFilter(new FiltroGIF());
  int retorno = fc.showOpenDialog(this);
  if (retorno == JFileChooser.APPROVE_OPTION)
    {
    ImageIcon ícone = new ImageIcon(fc.getSelectedFile().toString());
    imagem.setIcon(ícone);
    scrollImagem.setPreferredSize(new Dimension(300,300));
    scrollImagem.revalidate();
    }
  }

public static void main(String[] args)
  {
  new ExFileChooser();
  }
}
```

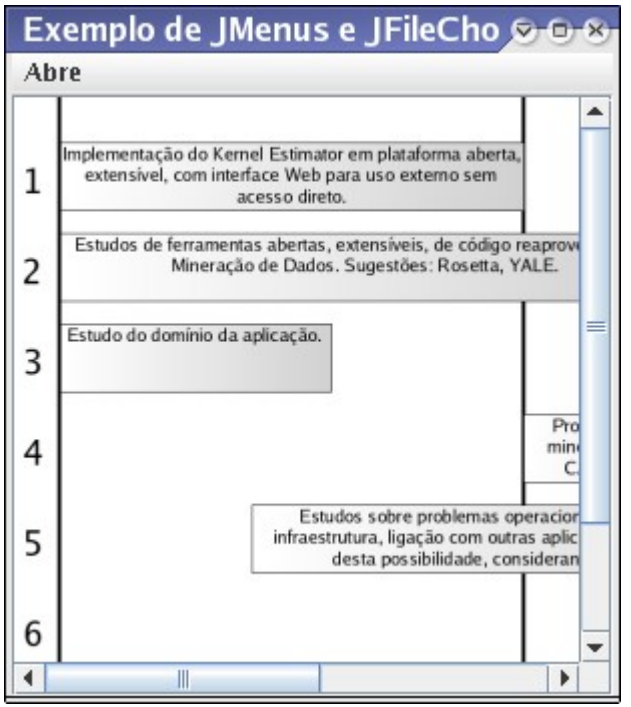
Componentes e eventos: JMenu

```
import java.io.File;
import javax.swing.filechooser.FileFilter;

public class FiltroJPG extends FileFilter
  {

  public boolean accept(File f)
    {
    if (f.isDirectory()) return true;
    if (f.toString().toUpperCase().endsWith(".JPG")) return true;
    if (f.toString().toUpperCase().endsWith(".JPEG")) return true;
    return false;
    }

  public String getDescription()
    {
    return "Arquivos JPEG";
    }
  }
```

Componentes e eventos: JMenu







JInternalFrame e MDI

- Podemos ter várias *frames* internas em uma mesma aplicação: *Multiple Document Interface*.

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

public class ImagemIF extends JInternalFrame
  {
  public ImagemIF(String name,float escala,ImageIcon ícone)
    {
    // Resizable, closable, maximizable e iconifiable.
    super(name,true,true,true,true);
    // Vamos mudar a escala da imagem?
    float width = ícone.getIconWidth();
    float height = ícone.getIconHeight();
    width *= escala; height *= escala;
    ícone = new ImageIcon(ícone.getImage().getScaledInstance((int)width,(int)height,
                                                             Image.SCALE_SMOOTH));
    // Mostra em um JLabel.
    getContentPane().add(new JScrollPane(new JLabel(ícone)));
    pack();
    setVisible(true);
    }
  }
```

JInternalFrame e MDI

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.net.*;
import javax.imageio.ImageIO;
import javax.swing.*;

public class MostraMultiplasImagens extends JFrame implements ActionListener
  {
  private JDesktopPane desktop;
  private JTextField url;
  private JComboBox escala;
  private String[] escalas = {"0.01","0.05","0.1","0.2","0.5","1","2","5","10","20"};
```

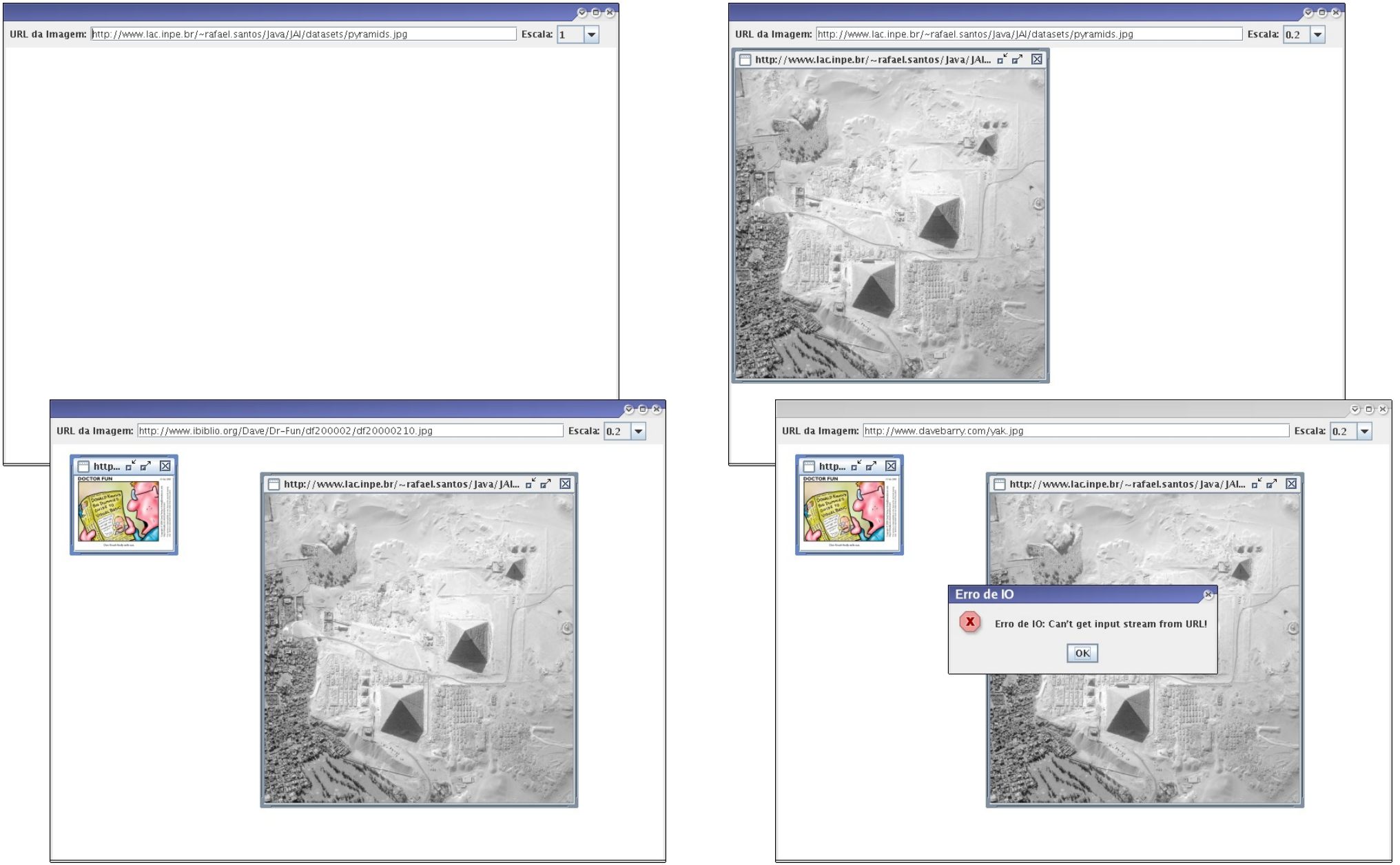
JInternalFrame e MDI

```
  public MostraMultiplasImagens()
    {
    desktop = new JDesktopPane();
    JPanel controle = new JPanel(new FlowLayout(FlowLayout.LEFT));
    controle.add(new JLabel("URL da Imagem:"));
    url = new JTextField(50);
    url.addActionListener(this);

url.setText("http://www.lac.inpe.br/~rafael.santos/Java/JAI/datasets/pyramids.jpg");
    controle.add(url);
    controle.add(new JLabel("Escala:"));
    escala = new JComboBox(escalas);
    escala.setSelectedIndex(5);
    controle.add(escala);
    getContentPane().add(controle,BorderLayout.NORTH);
    getContentPane().add(desktop,BorderLayout.CENTER);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(800,600);
    setVisible(true);
    }

  public static void main(String[] args)
    {
    new MostraMultiplasImagens();
    }

  }
```

JInternalFrame e MDI

```
public void actionPerformed(ActionEvent e)
  {
  BufferedImage imagem = null;      boolean carregada = true;
  try
    {
    imagem = ImageIO.read(new URL(url.getText()));
    }
  catch (MalformedURLException e1)
    {
    JOptionPane.showMessageDialog(this,"Erro na URL: "+url.getText(),
                                       "Erro na URL",JOptionPane.ERROR_MESSAGE); carregada = false;
    }
  catch (IOException e1)
    {
    JOptionPane.showMessageDialog(this,"Erro de IO: "+e1.getMessage(),
                                       "Erro de IO",JOptionPane.ERROR_MESSAGE);  carregada = false;
    }
  if (carregada)
    {
    if (imagem == null)
      JOptionPane.showMessageDialog(this,"Não pode ler "+url.getText(),
                                         "Não pode ler",JOptionPane.ERROR_MESSAGE);
    else
      {
      float usaEscala = Float.parseFloat(escalas[escala.getSelectedIndex()]);
      ImagemIF i = new ImagemIF(url.getText(),usaEscala,new ImageIcon(imagem));
      desktop.add(i);
      }
    }
  }
```

JInternalFrame e MDI



Applets

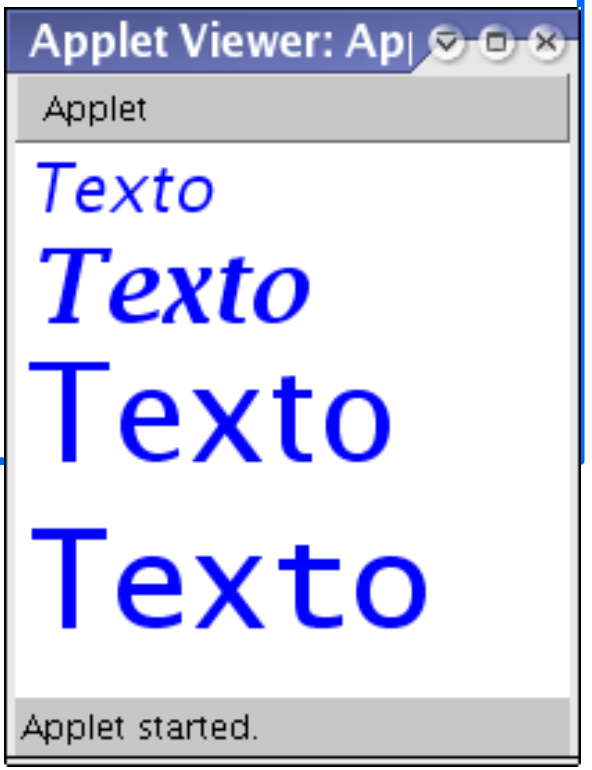
- Aplicações com interface gráfica que são executadas em um navegador.
- Mais seguras (para o cliente) do que aplicações.
- Menos flexíveis do que aplicações (*sandbox*).
- Idéia: apresentação de dados que são obtidos do servidor.
- Têm métodos que devem ser sobreescritos, em particular:
  - `init()`: inicializa a *applet*.
  - `paint(Graphics g)`: faz com que a *applet* seja pintada/desenhada.

Applets

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import javax.swing.JApplet;

public class Applet1 extends JApplet
  {
  public void paint(Graphics g)
    {
    Graphics2D g2d = (Graphics2D)g;
    g2d.setColor(Color.BLUE);
    g2d.drawString("Olá, Mundo",5,15);
    }
  }
```

Applets

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

public class Applet2 extends JApplet
  {
  public void paint(Graphics g)
    {
    Graphics2D g2d = (Graphics2D)g;
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                         RenderingHints.VALUE_ANTIALIAS_ON);
    g2d.setColor(Color.BLUE);
    g2d.setFont(new Font("SansSerif",Font.ITALIC,24));
    g2d.drawString("Texto",5,25);
    g2d.setFont(new Font("Serif",Font.ITALIC|Font.BOLD,36));
    g2d.drawString("Texto",5,65);
    g2d.setFont(new Font("Dialog",Font.PLAIN,48));
    g2d.drawString("Texto",5,115);
    g2d.setFont(new Font("DialogInput",Font.PLAIN,48));
    g2d.drawString("Texto",5,175);
    }
  }
```



Applets

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

public class AppletSliders extends JApplet implements ChangeListener
  {
  private JSlider naipe,face;
  private JLabel carta;

  public void init()
    {
    Container c = getContentPane();
    criaSliderNaipes();
    criaSliderFaces();
    JPanel controle = new JPanel(new GridLayout(2,1));
    controle.add(naipe); controle.add(face);
    carta = new JLabel();
    carta.setPreferredSize(new Dimension(99+20,134+20));
    carta.setHorizontalAlignment(SwingConstants.CENTER);
    c.add(controle,BorderLayout.SOUTH);
    c.add(carta,BorderLayout.CENTER);
    setSize(200,300);
    }
```

Applets

```
private void criaSliderNaipes()
  {
  naipe = new JSlider(0,3,0);
  Hashtable<Integer,JLabel> labels = new Hashtable<Integer,JLabel>();
  labels.put(new Integer(0),new JLabel("Paus"));
  labels.put(new Integer(1),new JLabel("Ouros"));
  labels.put(new Integer(2),new JLabel("Copas"));
  labels.put(new Integer(3),new JLabel("Espadas"));
  naipe.setLabelTable(labels);
  naipe.setPaintLabels(true); naipe.setPaintTicks(true); naipe.setSnapToTicks(true);
  naipe.setBorder(BorderFactory.createTitledBorder("Naipe"));
  naipe.addChangeListener(this);
  }

private void criaSliderFaces()
  {
  face = new JSlider(0,12,0);
  Hashtable<Integer,JLabel> labels = new Hashtable<Integer,JLabel>();
  for(int l=2;l<11;l++) labels.put(new Integer(l-1),new JLabel(""+l));
  labels.put(new Integer(0),new JLabel("A"));
  labels.put(new Integer(10),new JLabel("J"));
  labels.put(new Integer(11),new JLabel("Q"));
  labels.put(new Integer(12),new JLabel("K"));
  face.setLabelTable(labels);
  face.setPaintLabels(true); face.setPaintTicks(true); face.setSnapToTicks(true);
  face.setBorder(BorderFactory.createTitledBorder("Face"));
  face.addChangeListener(this);
  }
```

Applets

```
  public void stateChanged(ChangeEvent e)
    {
    String nome = String.format("%d-%02d.png",
                                new Object[]{naipe.getValue()+1,face.getValue()+1});
    ImageIcon ícone = new ImageIcon(nome);
    carta.setIcon(ícone);
    }

  }
```

Onde está a imagem? Deveria estar em uma URL!

```
Image i1 =
  new ImageIcon(getClass().getResource(nome)).getImage();
```

Criando Novos Componentes

Criando novos componentes

- Pode ser necessário criar novos componentes para exibição ou entrada de informações especializadas ou para exibir comportamento diferente dos componentes já existentes.
- Duas abordagens:
  - Criar componentes que herdam de outros, já existentes.
  - Criar novos componentes a partir de um componente genérico.

Criando Novos Componentes Gráficos

- Passos Usando Herança (nem todos são obrigatórios):
  - Herdar de classe que tem comportamento semelhante.
  - No construtor, chamar construtor ancestral, inicializar atributos relevantes e modificar comportamento através de métodos.
  - Sobreescrever métodos `get{Maximum,Minimum,Preferred}Size()`.
  - Sobreescrever `paintComponent()`.

Criando Novos Componentes Gráficos

- Exemplo: peça para Reversi.
  - Botão com aparência e comportamento diferente.
  - Existem várias maneiras de implementar...

```
package reversi;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;

import javax.swing.JButton;

public class Peca extends JButton
  {
  private static int tamanho = 64;
  private Estado estado;

  public Peca()
    {
    super();
    estado = Estado.VAZIO;
    }
```

```
package reversi;

public enum Estado { VAZIO, PRETO, BRANCO }
```

Criando Novos Componentes Gráficos

```
public Dimension getMaximumSize() { return getPreferredSize(); }
public Dimension getMinimumSize() { return getPreferredSize(); }
public Dimension getPreferredSize() { return new Dimension(tamanho,tamanho); }

protected void paintComponent(Graphics g)
  {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;
  // Não preenchemos botões vazios.
  if (estado != Estado.VAZIO)
    {
    if (estado == Estado.BRANCO) g2d.setColor(Color.WHITE);
    else if (estado == Estado.PRETO) g2d.setColor(Color.BLACK);
    g2d.fillOval(6,6,getWidth()-12,getHeight()-12);
    }
  // Pintamos a borda da peça independente do estado.
  g2d.setColor(Color.GRAY);
  g2d.drawOval(6,6,getWidth()-12,getHeight()-12);
  }
}
```

Criando Novos Componentes Gráficos

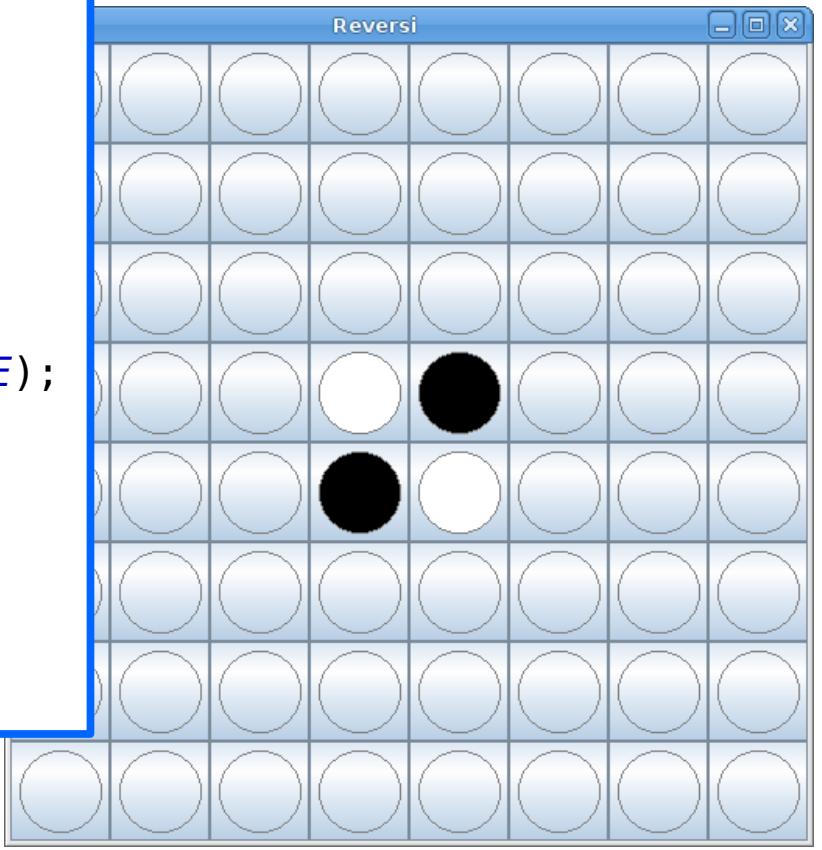
- Para mostrar a interface gráfica precisamos de:
  - Classe `Peca`, que representa botões para o jogo.
  - Classe `Tabuleiro`, que é um conjunto 8x8 de peças.
  - Classe `Jogo`, que é a aplicação que usa `Tabuleiro`.

Criando Novos Componentes Gráficos

```
package reversi;

import java.awt.GridLayout;
import javax.swing.JPanel;

public class Tabuleiro extends JPanel
  {
  private Peca[][] tabuleiro;

  public Tabuleiro()
    {
    setLayout(new GridLayout(8,8));
    tabuleiro = new Peca[8][8];
    for(int l=0;l<8;l++)
      for(int c=0;c<8;c++)
        {
        tabuleiro[c][l] = new Peca();
        add(tabuleiro[c][l]);
        }
    tabuleiro[3][3].setEstado(Estado.BRANCO);
    tabuleiro[4][4].setEstado(Estado.BRANCO);
    tabuleiro[3][4].setEstado(Estado.PRETO);
    tabuleiro[4][3].setEstado(Estado.PRETO);
    }

  }
```

Criando Novos Componentes Gráficos

```
package reversi;

import javax.swing.JFrame;

public class Jogo extends JFrame
  {

  public Jogo()
    {
    super("Reversi");
    getContentPane().add(new Tabuleiro());
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

  public static void main(String[] args)
    {
    new Jogo();
    }
  }
```



Dá pra melhorar a aparência?

Criando Novos Componentes Gráficos

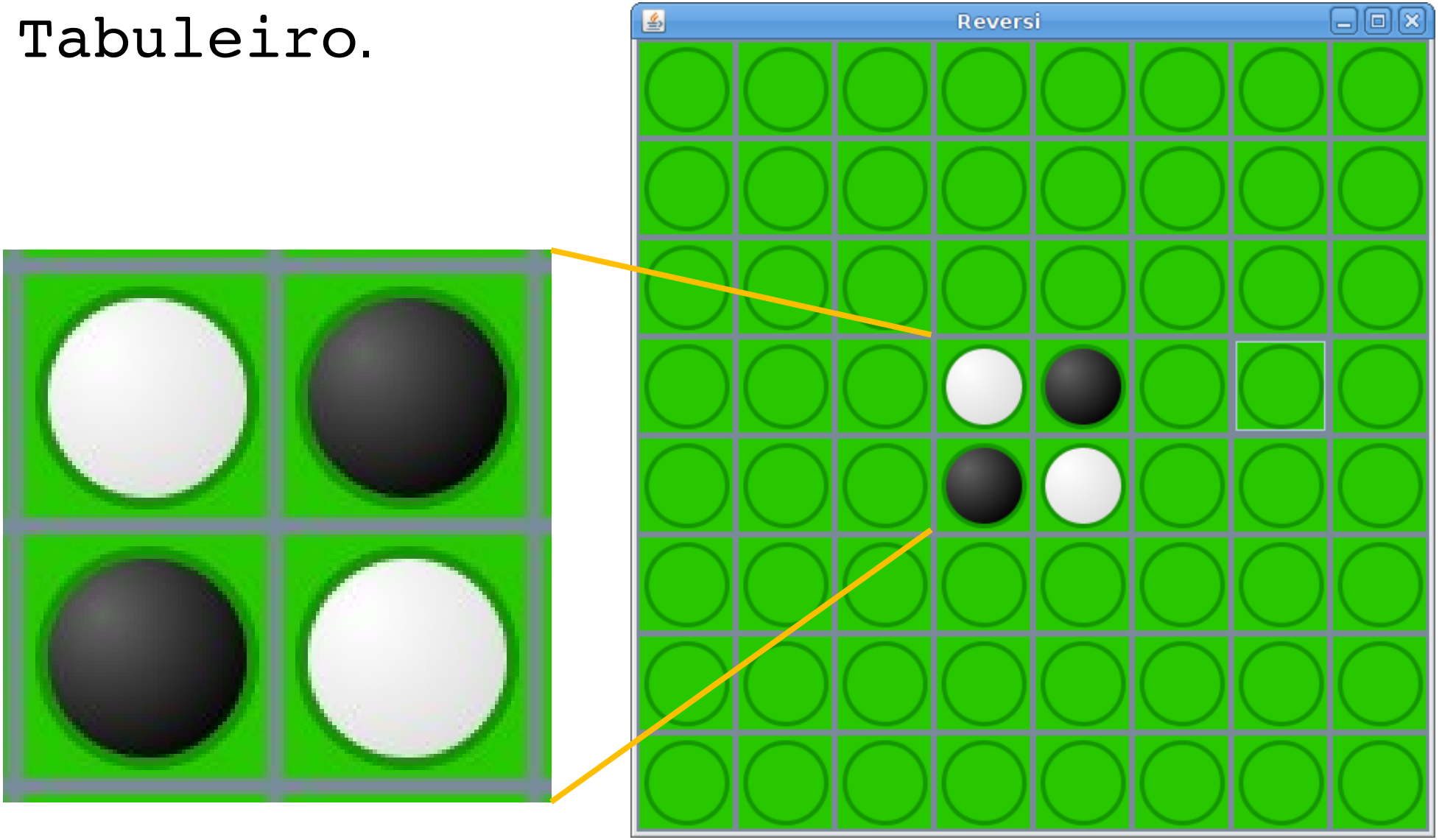
```
package reversi;

import java.awt.*;
import java.awt.geom.Point2D;

import javax.swing.JButton;

public class PecaMelhor extends JButton
  {
  private static int tamanho = 64;
  private Estado estado;

  public PecaMelhor()
    {
    super();
    setBackground(new Color(40,200,0));
    estado = Estado.VAZIO;
    }

  public void setEstado(Estado e) { estado = e; }

  public Dimension getMaximumSize() { return getPreferredSize(); }
  public Dimension getMinimumSize() { return getPreferredSize(); }
  public Dimension getPreferredSize() { return new Dimension(tamanho,tamanho); }
```

Criando Novos Componentes Gráficos

```
protected void paintComponent(Graphics g)
  {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                       RenderingHints.VALUE_ANTIALIAS_ON);
  // Não preenchemos botões vazios.
  if (estado != Estado.VAZIO)
    {
    Color[] cores = new Color[2];
    if (estado == Estado.BRANCO)
      { cores[0] = Color.WHITE; cores[1] = new Color(220,220,220); }
    else if (estado == Estado.PRETO)
      { cores[0] = new Color(100,100,100); cores[1] = Color.BLACK; }
    RadialGradientPaint paint =
      new RadialGradientPaint(new Point2D.Double(tamanho/3,tamanho/3),
                              2*tamanho/3,new float[]{0f,1f},cores);
    g2d.setPaint(paint);
    g2d.fillOval(6,6,getWidth()-12,getHeight()-12);
    }
  // Pintamos a borda da peça independente do estado.
  g2d.setColor(new Color(20,150,0));
  g2d.setStroke(new BasicStroke(3f));
  g2d.drawOval(6,6,getWidth()-12,getHeight()-12);
  }
}
```

Criando Novos Componentes Gráficos

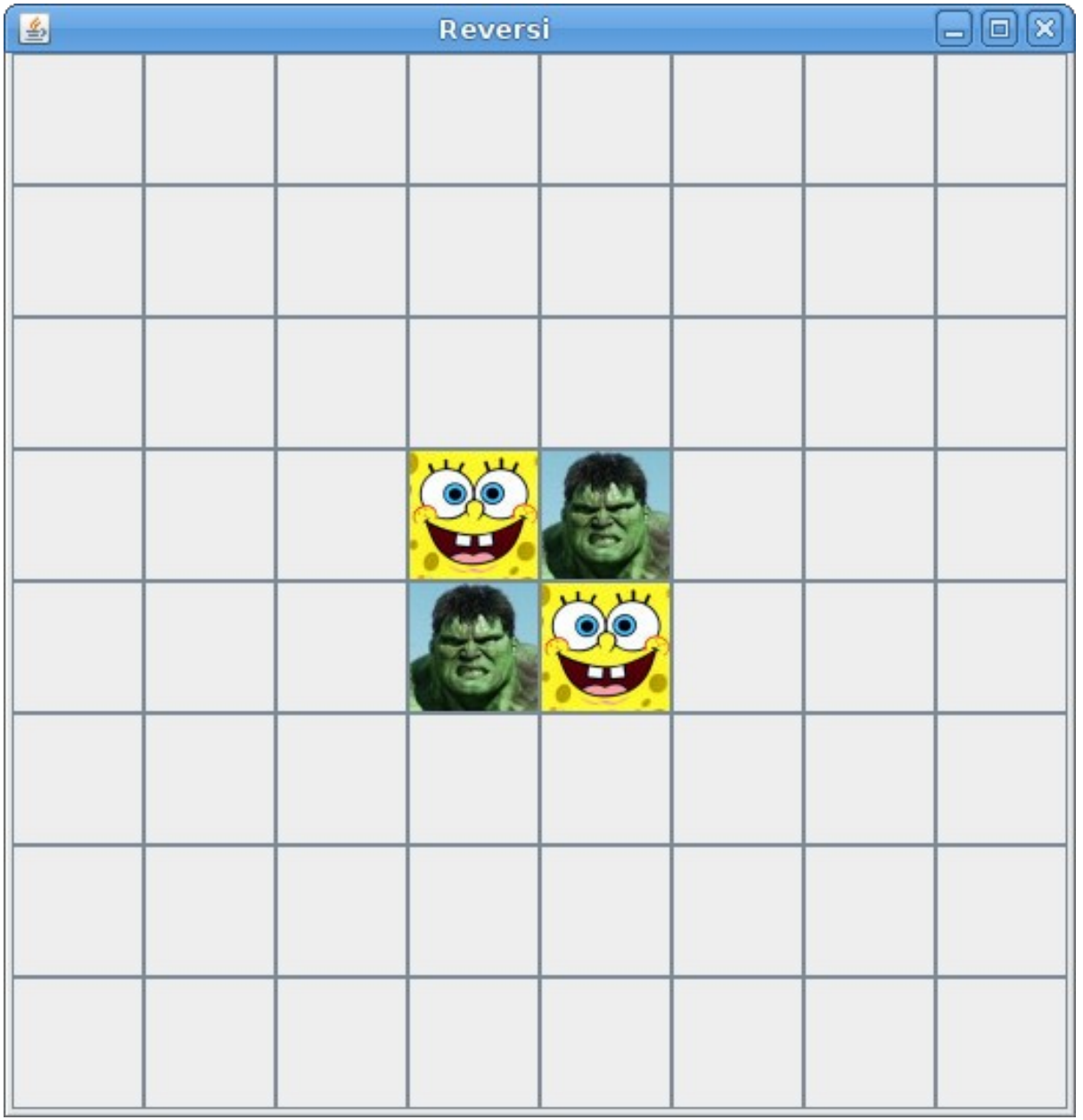
- Basta usar `PecaMelhor` no lugar de `Peca` em `Tabuleiro`.



Criando Novos Componentes Gráficos

```
...
public class PecaIcone extends JButton
  {
  private static int tamanho = 64;
  private Estado estado;
  private Image i1,i2;

  public PecaIcone()
    {
    super();
    setContentAreaFilled(false);
    estado = Estado.VAZIO;
    i1 = new ImageIcon(getClass().getResource("/Sprites/sbob.jpg")).getImage();
    i2 = new ImageIcon(getClass().getResource("/Sprites/hulk.jpg")).getImage();
    } ...
  protected void paintComponent(Graphics g)
    {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D)g;
    // Não preenchemos botões vazios.
    if (estado != Estado.VAZIO)
      {
      if (estado == Estado.BRANCO) g2d.drawImage(i1,0,0,null);
      else if (estado == Estado.PRETO) g2d.drawImage(i2,0,0,null);
      }
    }...
```

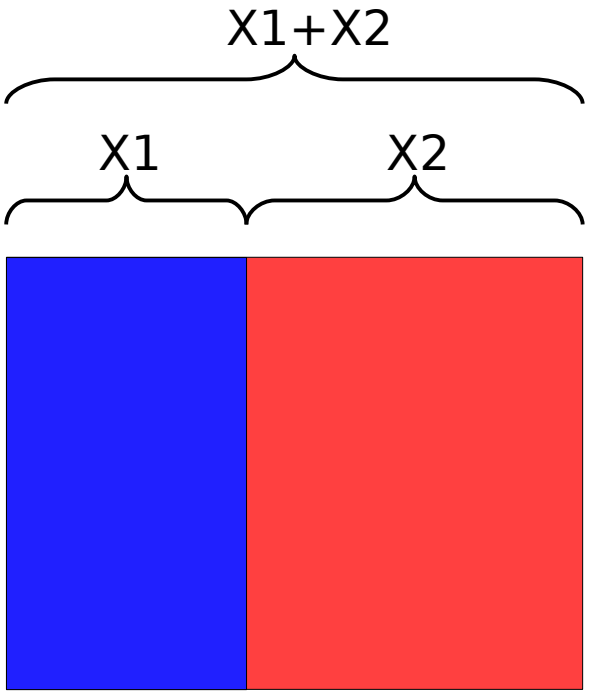
Criando Novos Componentes Gráficos



Criando novos componentes

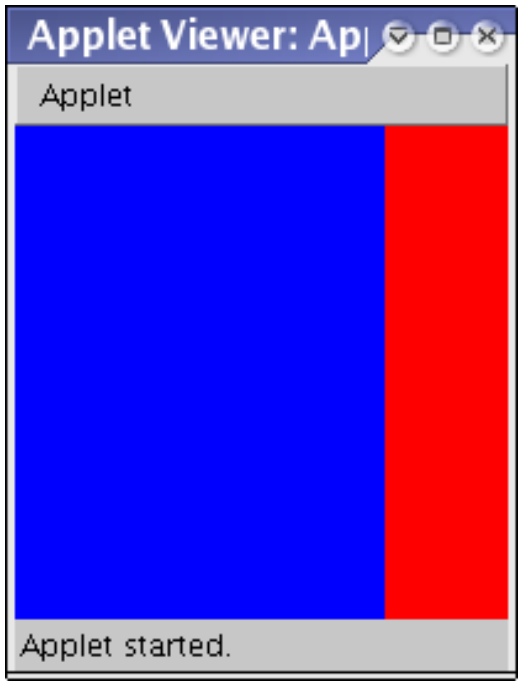
- Herdando de `JComponent`:
- Mostra duas barras de larguras proporcionais.

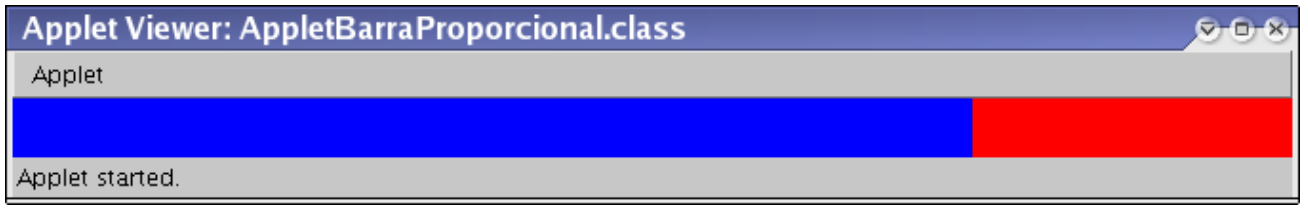
```
import java.awt.*;
import javax.swing.JComponent;

public class BarraProporcional extends JComponent
  {
  private int x1,x2;

  public BarraProporcional(int x1,int x2)
    {
    this.x1 = x1; this.x2 = x2;
    }
  protected void paintComponent(Graphics g)
    {
    float w = getWidth()*x1/(x1+x2);
    g.setColor(Color.BLUE);
    g.fillRect(0,0,(int)w,getHeight());
    g.setColor(Color.RED);
    g.fillRect((int)w,0,getWidth(),getHeight());
    }

  }
```

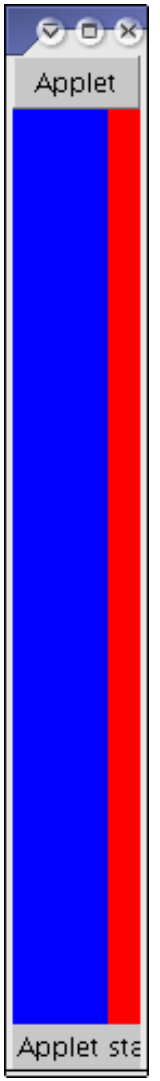


Criando novos componentes

```
import javax.swing.JApplet;

public class AppletBarraProporcional extends JApplet
  {
  public void init()
    {
    BarraProporcional b = new BarraProporcional(75,25);
    getContentPane().add(b);
    }
  }
```

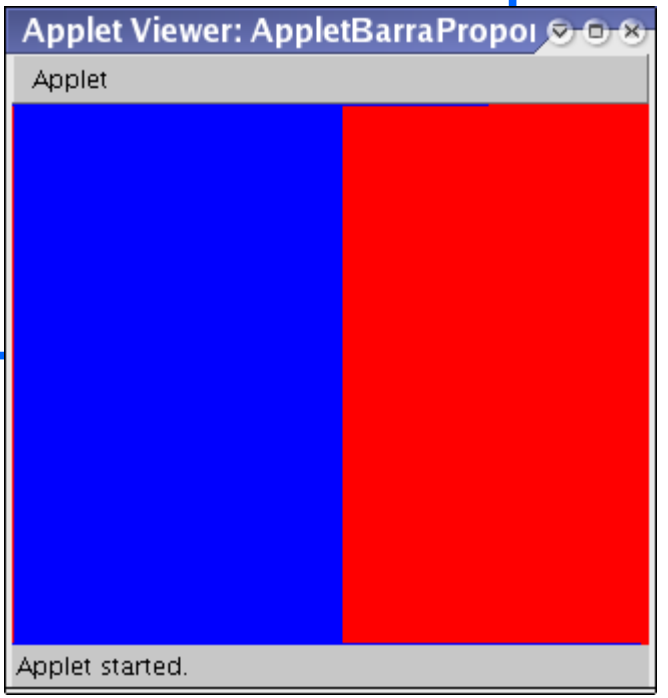






Criando novos componentes

```
import java.awt.BorderLayout;
import javax.swing.JApplet;

public class AppletBarraProporcional2 extends JApplet
  {
  public void init()
    {
    BarraProporcional bN = new BarraProporcional(75,25);
    BarraProporcional bS = new BarraProporcional(99,1);
    BarraProporcional bE = new BarraProporcional(99,98);
    BarraProporcional bW = new BarraProporcional(3,2);
    BarraProporcional bC = new BarraProporcional(120,110);
    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(bN,BorderLayout.NORTH);
    getContentPane().add(bS,BorderLayout.SOUTH);
    getContentPane().add(bE,BorderLayout.EAST);
    getContentPane().add(bW,BorderLayout.WEST);
    getContentPane().add(bC,BorderLayout.CENTER);
    }
  }
```



Problemas com dimensionamento de componentes!

Devemos garantir dimensões preferidas, mínimas e/ou máximas.

Criando novos componentes

- Componentes podem processar seus próprios eventos.

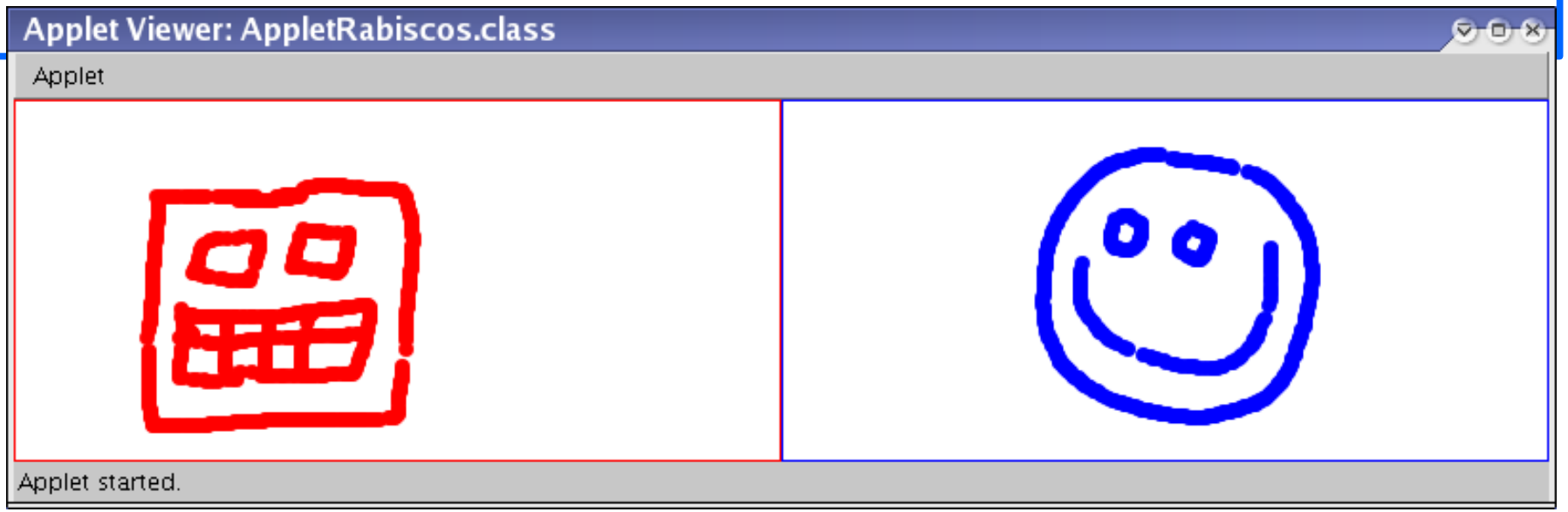
```
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import javax.swing.JComponent;

public class ComponenteParaRabiscos extends JComponent
                                    implements MouseListener,MouseMotionListener
  {
  private ArrayList<Point> pontos;
  private int size = 8; private int halfsize = size/2;
  private Color cor;

  public ComponenteParaRabiscos(Color cor)
    {
    this.cor = cor;
    pontos = new ArrayList<Point>(1024);
    addMouseListener(this);
    addMouseMotionListener(this);
    }
```

Criando novos componentes

```
protected void paintComponent(Graphics g)
  {
  Graphics2D g2d = (Graphics2D)g;
  g2d.setColor(Color.WHITE);
  g2d.fillRect(0,0,getWidth(),getHeight());
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                       RenderingHints.VALUE_ANTIALIAS_ON);
  g2d.setColor(cor);
  for(Point p:pontos)
    g2d.fillOval(p.x-halfsize,p.y-halfsize,size,size);
  }

public void mousePressed(MouseEvent e)
  {
  pontos.add(e.getPoint());   repaint();
  }

public void mouseDragged(MouseEvent e)
  {
  pontos.add(e.getPoint());   repaint();
  }

public void mouseReleased(MouseEvent e) { } // NOP
public void mouseClicked(MouseEvent e) { } // NOP
public void mouseEntered(MouseEvent e) { } // NOP
public void mouseExited(MouseEvent e) { } // NOP
public void mouseMoved(MouseEvent e) { } // NOP
}
```

Criando novos componentes

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

public class AppletRabiscos extends JApplet
  {
  public void init()
    {
    ComponenteParaRabiscos c1 = new ComponenteParaRabiscos(Color.RED);
    c1.setBorder(BorderFactory.createLineBorder(Color.RED));
    ComponenteParaRabiscos c2 = new ComponenteParaRabiscos(Color.BLUE);
    c2.setBorder(BorderFactory.createLineBorder(Color.BLUE));
    getContentPane().setLayout(new GridLayout(1,2));
    getContentPane().add(c1);
    getContentPane().add(c2);
    }
  }
```



Criando novos componentes

- Componentes podem produzir e consumir seus próprios eventos.

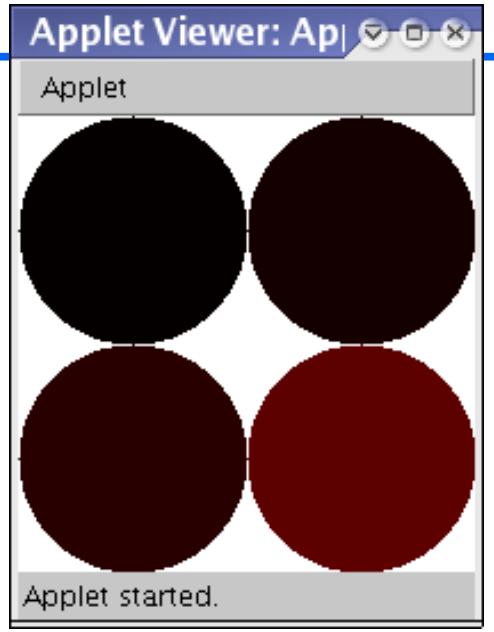
```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JComponent;
import javax.swing.Timer;

public class ComponenteLuzVermelha extends JComponent implements ActionListener
  {
  private int nível,passo;
  private Timer timer;

  public ComponenteLuzVermelha(int passo)
    {
    this.passo = passo;
    nível = 0;
    timer = new Timer(50,this);
    timer.setCoalesce(true);
    timer.start();
    }
```

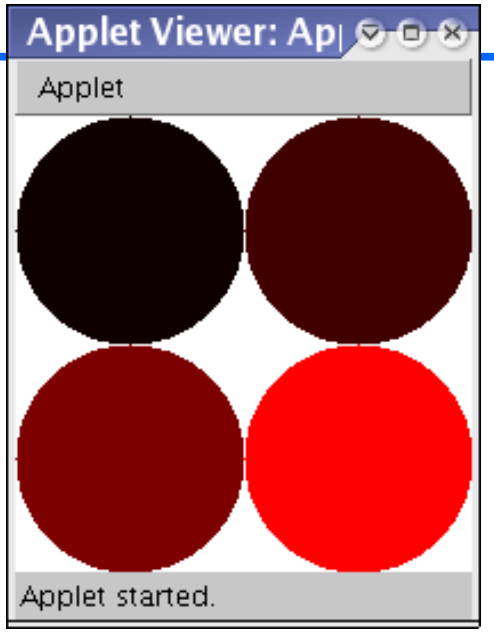
Criando novos componentes

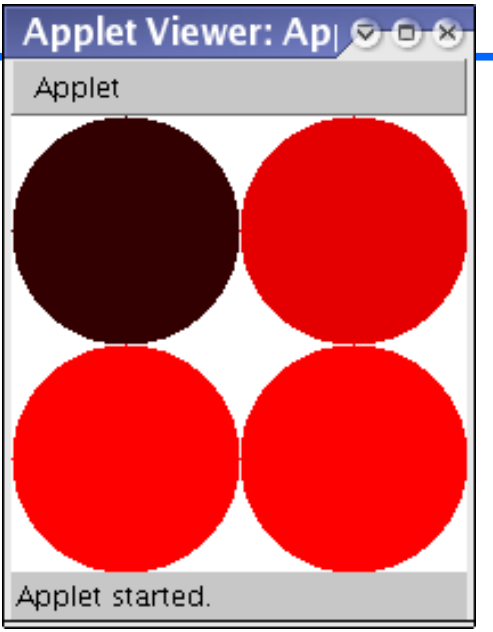
```
protected void paintComponent(Graphics g)
  {
  g.setColor(Color.WHITE);
  g.fillRect(0,0,getWidth(),getHeight());
  // Calculamos a cor de acordo com o passo.
  g.setColor(new Color(nível/100,0,0));
  g.fillArc(0,0,getWidth(),getHeight(),0,360);
  }

public void actionPerformed(ActionEvent e)
  {
  if (nível < 25500) nível += passo;
  repaint();
  }

}
```

Criando novos componentes

```
import java.awt.GridLayout;
import javax.swing.JApplet;

public class AppletLuzVermelha extends JApplet
  {
  public void init()
    {
    getContentPane().setLayout(new GridLayout(2,2));
    ComponenteLuzVermelha c1,c2,c3,c4;
    c1 = new ComponenteLuzVermelha(10);        c2 = new ComponenteLuzVermelha(50);
    c3 = new ComponenteLuzVermelha(100);       c4 = new ComponenteLuzVermelha(250);
    getContentPane().add(c1); getContentPane().add(c2);
    getContentPane().add(c3); getContentPane().add(c4);
    }
  }
```







Exemplo: Tanques

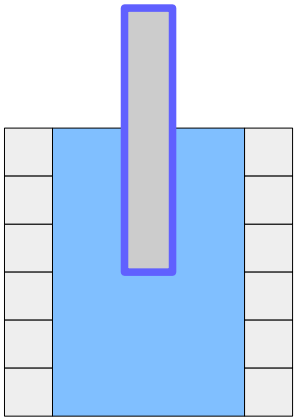
Tanques

- Primeiros passos em uma simulação muito simples.
- **Tanques** podem andar para a frente, modificar a velocidade, girar nos sentidos horário e anti-horário.
- **Arena** comporta vários tanques e permite a manipulação dos mesmos através do *mouse*.
- **Aplicação** cria instância da Arena.
- Arena é um componente bastante específico, Tanque não.

Tanques

- Classe `Tanque`: Declarações

```
package tanques;

import java.awt.*;
import java.awt.geom.AffineTransform;

public class Tanque
  {
  private double x,y;
  private double ângulo;
  private double velocidade;
  private Color cor;
  private boolean estáAtivo;
  public Tanque(int x,int y,int a,Color c)
    {
    this.x = x; this.y = y; ângulo = 90-a; cor = c;
    velocidade = 0;
    estáAtivo = false;
    }
```



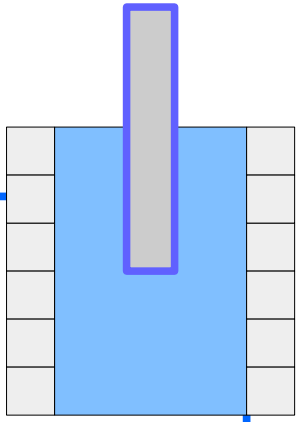
Tanques

- Classe `Tanque`: Modificadores

```
public void aumentaVelocidade()
  {
  velocidade++;
  }

public void giraHorário(int a)
  {
  ângulo += a;
  }

public void giraAntiHorário(int a)
  {
  ângulo -= a;
  }

public void move()
  {
  x = x + Math.sin(Math.toRadians(ângulo))*velocidade;
  y = y - Math.cos(Math.toRadians(ângulo))*velocidade;
  }

public void setEstáAtivo(boolean estáAtivo)
  {
  this.estáAtivo = estáAtivo;
  }
```

Tanques

- Classe `Tanque`: Desenho (1)

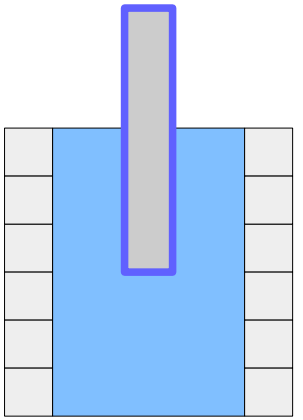


```
public void draw(Graphics2D g2d)
  {
  // Armazenamos o sistema de coordenadas original.
  AffineTransform antes = g2d.getTransform();
  // Criamos um sistema de coordenadas para o robô.
  AffineTransform at = new AffineTransform();
  at.translate(x,y);
  at.rotate(Math.toRadians(ângulo));
  // Aplicamos o sistema de coordenadas.
  g2d.transform(at);
  // Desenhamos o tanque na posição 0,0. Primeiro o corpo:
  g2d.setColor(cor);
  g2d.fillRect(-10,-12,20,24);
```

Tanques

- Classe `Tanque`: Desenho (2)

```
    // Agora as esteiras
    for(int e=-12;e<=8;e+=4)
      {
      g2d.setColor(Color.LIGHT_GRAY);
      g2d.fillRect(-15,e,5,4);
      g2d.fillRect(10,e,5,4);
      g2d.setColor(Color.BLACK);
      g2d.drawRect(-15,e,5,4);
      g2d.drawRect(10,e,5,4);
      }
    // Finalmente o canhão.
    g2d.setColor(Color.LIGHT_GRAY);
    g2d.fillRect(-3,-25,6,25);
    g2d.setColor(cor);
    g2d.drawRect(-3,-25,6,25);
```



Tanques

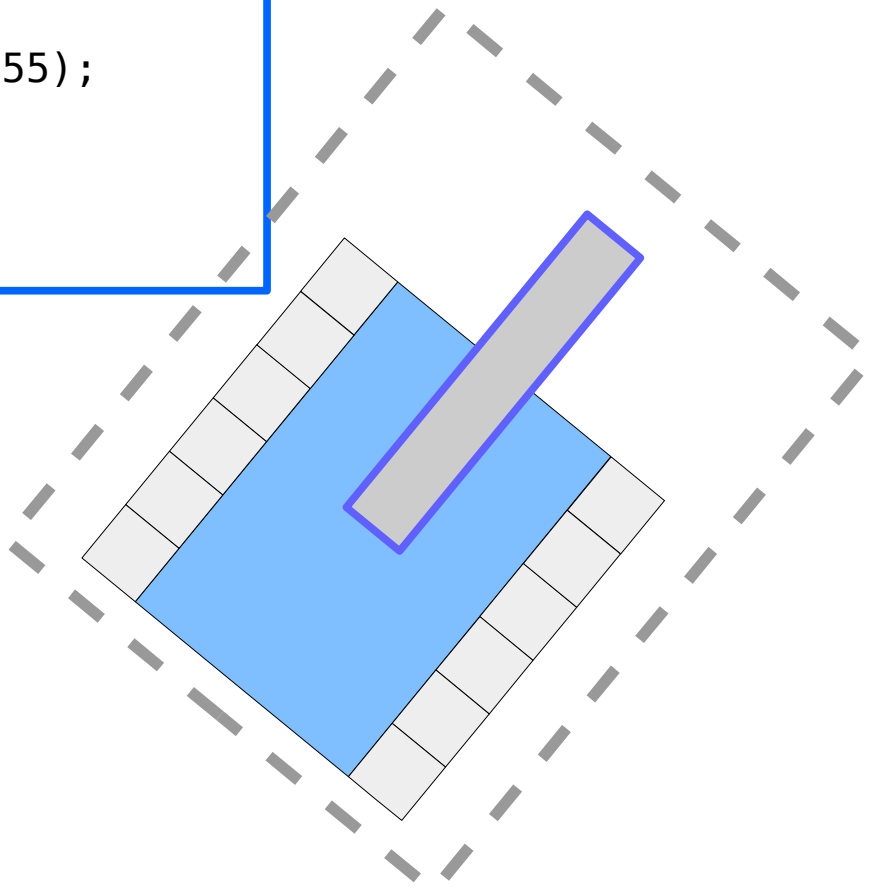
- Classe `Tanque`: Desenho (3)

```
    // Se o tanque estiver ativo, desenhamos uma margem nele.
    if (estáAtivo)
      {
      g2d.setColor(new Color(120,120,120));
      Stroke linha = g2d.getStroke();
      g2d.setStroke(new BasicStroke(1f,BasicStroke.CAP_ROUND,
                        BasicStroke.JOIN_ROUND,0,
                        new float[]{8},0));
      g2d.drawRect(-24,-32,48,55);
      g2d.setStroke(linha);
      }
    // Aplicamos o sistema de coordenadas original.
    g2d.setTransform(antes);
    }
```

Tanques

- Classe `Tanque`: Outros

```
  public Shape getRectEnvolvente()
    {
    AffineTransform at = new AffineTransform();
    at.translate(x,y);
    at.rotate(Math.toRadians(ângulo));
    Rectangle rect = new Rectangle(-24,-32,48,55);
    return at.createTransformedShape(rect);
    }

  }
```



Tanques

- Classe `Arena`: Declarações

```
package tanques;

import java.awt.*;
import java.awt.event.*;
import java.util.HashSet;
import javax.swing.*;

public class Arena extends JComponent implements MouseListener, ActionListener
  {
  private int w,h;
  private HashSet<Tanque> tanques;
  private Timer timer;

  public Arena(int w,int h)
    {
    this.w = w; this.h = h;
    tanques = new HashSet<Tanque>();
    addMouseListener(this);
    timer = new Timer(500,this);
    timer.start();
    }
```

Tanques

- Classe `Arena`: Modificadores e métodos “mágicos”

```
public void adicionaTanque(Tanque t)
  {
  tanques.add(t);
  }

public Dimension getMaximumSize()
  {
  return getPreferredSize();
  }

public Dimension getMinimumSize()
  {
  return getPreferredSize();
  }

public Dimension getPreferredSize()
  {
  return new Dimension(w,h);
  }
```

Tanques

- Classe `Arena`: Desenho

```
protected void paintComponent(Graphics g)
  {
  super.paintComponent(g);
  Graphics2D g2d = (Graphics2D)g;
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                       RenderingHints.VALUE_ANTIALIAS_ON);
  g2d.setColor(new Color(245,245,255));
  g2d.fillRect(0,0,w,h);
  g2d.setColor(new Color(220,220,220));
  for(int _w=0;_w<=w;_w+=20) g2d.drawLine(_w,0,_w,h);
  for(int _h=0;_h<=h;_h+=20) g2d.drawLine(0,_h,w,_h);
  // Desenhamos todos os tanques
  for(Tanque t:tanques) t.draw(g2d);
  }
```

Tanques

- Classe `Arena`: Eventos (1)

```
public void mouseClicked(MouseEvent e)
  {
  for(Tanque t:tanques) t.setEstáAtivo(false);
  for(Tanque t:tanques)
    {
    boolean clicado = t.getRectEnvolvente().contains(e.getX(),e.getY());
    if (clicado)
      {
      t.setEstáAtivo(true);
      switch(e.getButton())
        {
        case MouseEvent.BUTTON1: t.giraAntiHorário(3); break;
        case MouseEvent.BUTTON2: t.aumentaVelocidade(); break;
        case MouseEvent.BUTTON3: t.giraHorário(3); break;
        }
      break;
      }
    }
  repaint();
  }
```

Tanques

- Classe `Arena`: Eventos (2)

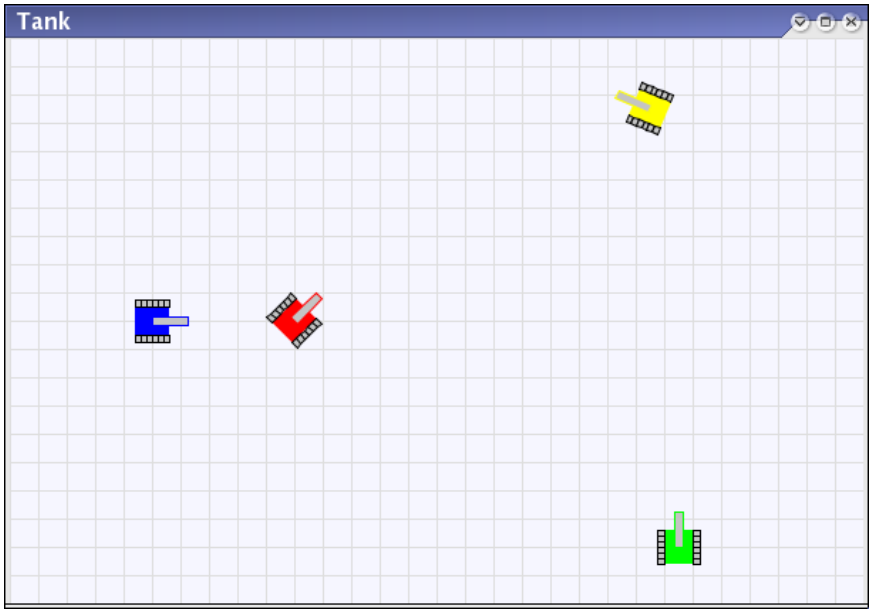
```
public void mouseEntered(MouseEvent e) { }

public void mouseExited(MouseEvent e) { }

public void mousePressed(MouseEvent e) { }

public void mouseReleased(MouseEvent e) { }

public void actionPerformed(ActionEvent e)
  {
  for(Tanque t:tanques) t.move();
  repaint();
  }

}
```

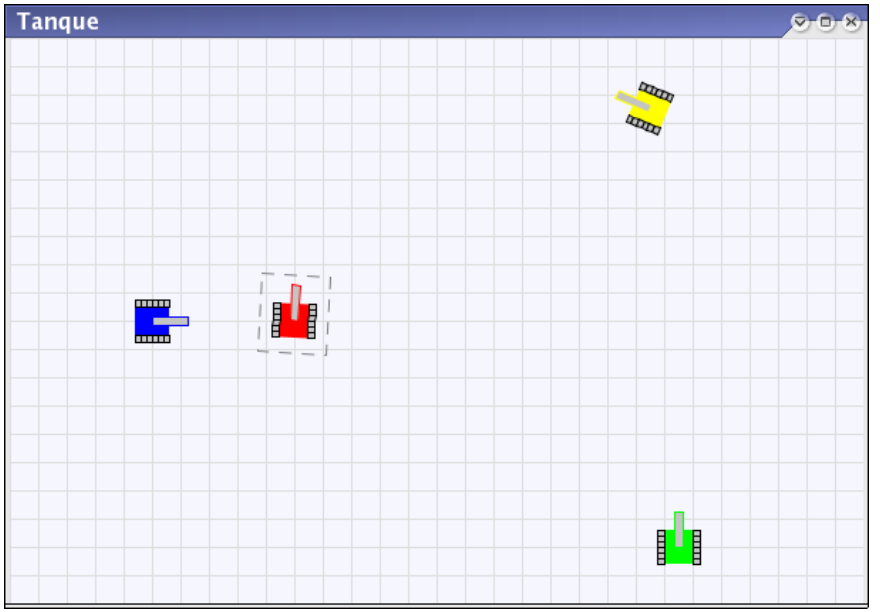
Tanques

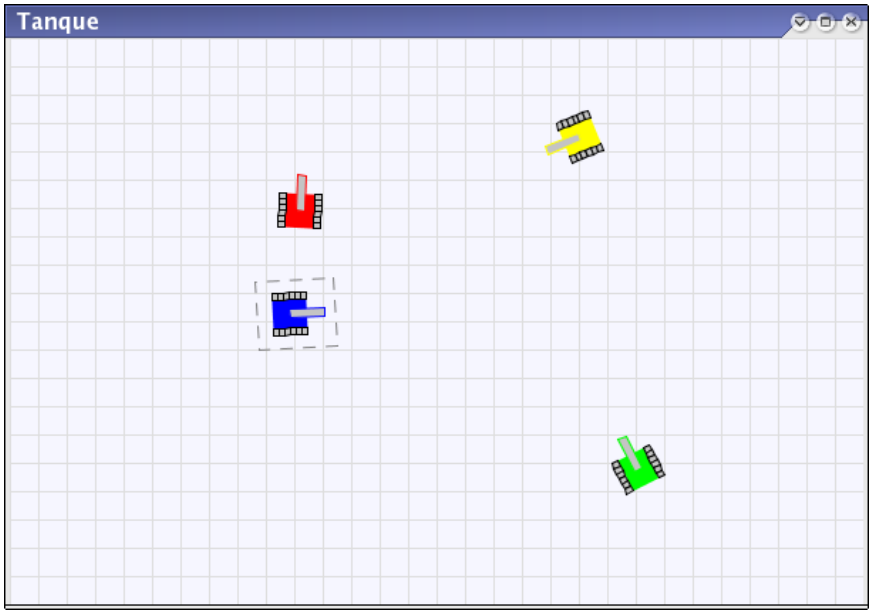
- Classe `App`

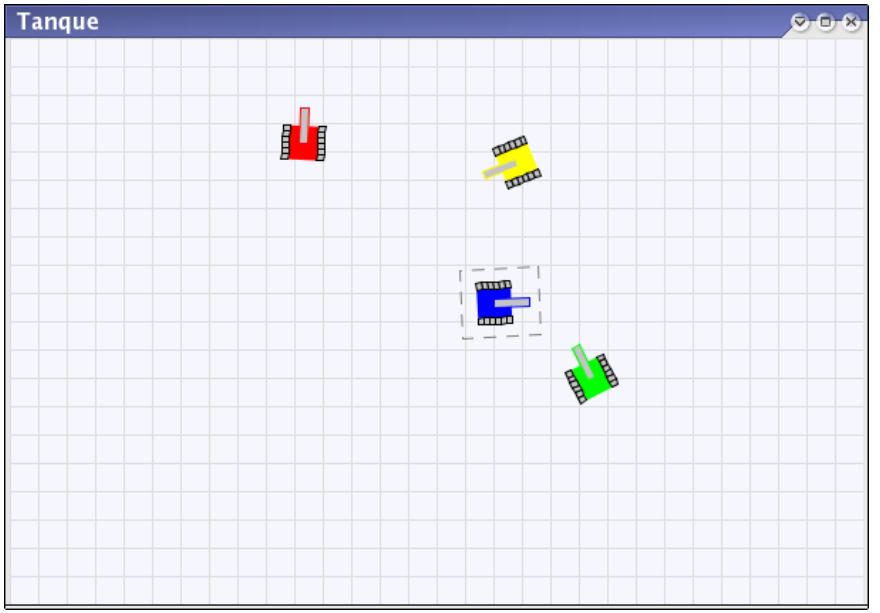
```
package tanques;

import java.awt.Color;
import javax.swing.JFrame;

public class App
  {
  public static void main(String[] args)
    {
    Arena arena = new Arena(600,400);
    arena.adicionaTanque(new Tanque(100,200,  0,Color.BLUE));
    arena.adicionaTanque(new Tanque(200,200, 45,Color.RED));
    arena.adicionaTanque(new Tanque(470,360, 90,Color.GREEN));
    arena.adicionaTanque(new Tanque(450, 50,157,Color.YELLOW));

    JFrame f = new JFrame("Tanques");
    f.getContentPane().add(arena);
    f.pack();
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
  }
```

Tanques









Tanques

- Como está, não é interessante, mas...
- É simples criar um ambiente onde podemos interagir com vários objetos.
- Como fazer objetos interagir com outros?
  - Colisão: detectável com métodos que verificam intersecção entre retângulos envolventes.
  - Limites da arena: verificável de forma simples.
- Sugestões de projetos

Exemplo: Celofane

Celofane

- Primeiros passos em um editor gráfico interativo.
- Objetos podem ser desenhados, coloridos e modificados.
- Regras para desenhar objetos estão em suas classes, que herdam da classe abstrata `ObjetoGeometrico`.
- A classe `AreaDeDesenho` mostra desenhos e permite interação.
- A classe `Celofane` é uma aplicação que usa `AreaDeDesenho`.

Celofane

- Classe abstrata `ObjetoGeometrico`

```
package celofane;

import java.awt.Color;
import java.awt.Graphics2D;

public abstract class ObjetoGeometrico
  {
  protected int x,y;
  protected Color cor;

  public int getX() { return x; }
  public int getY() { return y; }
  public void setX(int x) { this.x = x; } // poderíamos ter condições...
  public void setY(int y) { this.y = y; }
  public void setCor(Color c) { cor = c; }
  public Color getCor() { return cor; }
  public abstract void desenha(Graphics2D g2d,boolean selecionado);
  public abstract boolean contém(int px,int py);
  }
```

- Classe `ObjetoQuadrado` (1)

```
package celofane;

import java.awt.Color;
import java.awt.Graphics2D;

public class ObjetoQuadrado extends ObjetoGeometrico
  {
  private int lado;
  public ObjetoQuadrado(int x,int y,int t)
    {
    setX(x); setY(y);
    lado = t;
    setCor(Color.BLACK);
    }

  public boolean contém(int px, int py)
    {
    if ((px >= x-lado/2) && (px <= x+lado/2) &&
        (py >= y-lado/2) && (py <= y+lado/2)) return true;
    else return false;
    }
```

- Classe `ObjetoQuadrado` (2)

```
  public void desenha(Graphics2D g2d,boolean selecionado)
    {
    if (selecionado)
      {
      g2d.setColor(Color.WHITE);
      g2d.fillRect(x-lado/2,y-lado/2,lado,lado);
      g2d.setColor(cor);
      g2d.drawRect(x-lado/2,y-lado/2,lado,lado);
      }
    else
      {
      g2d.setColor(cor);
      g2d.fillRect(x-lado/2,y-lado/2,lado,lado);
      }
    }

  }
```

- Classe `ObjetoCirculo` (1)

```
package celofane;

import java.awt.Color;
import java.awt.Graphics2D;

public class ObjetoCirculo extends ObjetoGeometrico
  {
  private int raio;
  public ObjetoCirculo(int x,int y,int r)
    {
    setX(x); setY(y);
    raio = r;
    setCor(Color.BLACK);
    }

  public boolean contém(int px, int py)
    {
    if ((px >= x-raio) && (px <= x+raio) &&
        (py >= y-raio) && (py <= y+raio)) return true;
    else return false;
    }
```

- Classe `ObjetoCirculo` (2)

```
public void desenha(Graphics2D g2d,boolean selecionado)
  {
  if (selecionado)
    {
    g2d.setColor(Color.WHITE);
    g2d.fillOval(x-raio,y-raio,2*raio,2*raio);
    g2d.setColor(cor);
    g2d.drawOval(x-raio,y-raio,2*raio,2*raio);
    }
  else
    {
    g2d.setColor(cor);
    g2d.fillOval(x-raio,y-raio,2*raio,2*raio);
    }
  }
}
```

- Classe `ObjetoTriangulo` (1)

```
package celofane;

import java.awt.*;

public class ObjetoTriangulo extends ObjetoGeometrico
  {
  private int base;
  private Polygon forma;
  public ObjetoTriangulo(int x,int y,int b)
    {
    setX(x); setY(y);
    base = b;
    calculaForma();
    setCor(Color.BLACK);
    }
  public void setX(int x)
    {
    super.setX(x);     calculaForma();
    }
  public void setY(int y)
    {
    super.setY(y);     calculaForma();
    }
```

- Classe `ObjetoTriangulo` (2)

```
  private void calculaForma()
    {
    int[] xp = new int[3];
    int[] yp = new int[3];
    xp[0] = x;         yp[0] = y-base/2;
    xp[1] = x-base/2; yp[1] = y+base/2;
    xp[2] = x+base/2; yp[2] = y+base/2;
    forma = new Polygon(xp,yp,3);
    }

  public boolean contém(int px, int py)
    {
    if ((px >= x-base) && (px <= x+base) &&
        (py >= y-base) && (py <= y+base)) return true;
    else return false;
    }
```

- Classe `ObjetoTriangulo` (3)

```
public void desenha(Graphics2D g2d,boolean selecionado)
  {
  if (selecionado)
    {
    g2d.setColor(Color.WHITE);
    g2d.fill(forma);
    g2d.setColor(cor);
    g2d.draw(forma);
    }
  else
    {
    g2d.setColor(cor);
    g2d.fill(forma);
    }
  }

}
```

- Classe `AreaDeDesenho` (1)

```
package celofane;

import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import javax.swing.*;

public class AreaDeDesenho extends JComponent
                           implements MouseListener, MouseMotionListener,\
                                      ActionListener
  {
  private int w,h;
  private int x,y; // posições atuais do mouse
  private int selecionado;
  private JPopupMenu popupMenu;
  private JMenuItem quadrado,círculo,triângulo;
  private ArrayList<ObjetoGeometrico> objetos;
```

- Classe `AreaDeDesenho` (2)

```
public AreaDeDesenho(int w,int h)
  {
  this.w = w; this.h = h;
  selecionado = -1;
  objetos = new ArrayList<ObjetoGeometrico>();
  // Criamos o menu de objetos
  popupMenu = new JPopupMenu();
  quadrado = new JMenuItem("Quadrado");
  quadrado.addActionListener(this);
  popupMenu.add(quadrado);
  círculo = new JMenuItem("Círculo");
  círculo.addActionListener(this);
  popupMenu.add(círculo);
  triângulo = new JMenuItem("Triângulo");
  triângulo.addActionListener(this);
  popupMenu.add(triângulo);
  addMouseListener(this);
  addMouseMotionListener(this);
  }

public Dimension getMaximumSize() { return getPreferredSize(); }
public Dimension getMinimumSize() { return getPreferredSize(); }
public Dimension getPreferredSize() { return new Dimension(w,h);  }
```

- Classe `AreaDeDesenho` (3)

```
protected void paintComponent(Graphics g)
  {
  Graphics2D g2d = (Graphics2D)g;
  g2d.setColor(Color.WHITE);
  g2d.fillRect(0,0,w,h);
  for(int o=0;o<objetos.size();o++)
    {
    if (o == selecionado) objetos.get(o).desenha(g2d,true);
    else objetos.get(o).desenha(g2d,false);
    }
  }
```

- Classe `AreaDeDesenho` (4)

```
public void mouseClicked(MouseEvent e)
  {
  if (e.getButton() == MouseEvent.BUTTON1)
    {
    // Procuramos algum objeto sob a posição do mouse.
    selecionado = -1;
    for(int o=0;o<objetos.size();o++)
      if (objetos.get(o).contém(e.getX(),e.getY())) selecionado = o;
    }
  else if (e.getButton() == MouseEvent.BUTTON3)
    {
    x = e.getX(); y = e.getY();
    if (selecionado >= 0)
      {
      if (objetos.get(selecionado).contém(x,y)) // Estamos sobre um objeto selecionado?
        {
        // Mudamos a cor deste objeto.
        Color novaCor = JColorChooser.showDialog(this,"Escolha uma cor",
                                                 objetos.get(selecionado).getCor());
        objetos.get(selecionado).setCor(novaCor);
        selecionado = -1;
        }
      else popupMenu.show(e.getComponent(),x,y); // Clicamos fora do selecionado
      }
    else popupMenu.show(e.getComponent(),x,y); // Não temos nada selecionado
    }
  repaint();
  }
```

- Classe `AreaDeDesenho` (5)

```
public void mouseEntered(MouseEvent e) { }

public void mouseExited(MouseEvent e) { }

public void mousePressed(MouseEvent e) { }

public void mouseReleased(MouseEvent e)
  {
  if (popupMenu.isVisible()) popupMenu.setVisible(false);
  }

public void mouseDragged(MouseEvent e)
  {
  x = e.getX(); y = e.getY();
  if (selecionado >= 0)
    {
    objetos.get(selecionado).setX(x);
    objetos.get(selecionado).setY(y);
    }
  repaint();
  }

public void mouseMoved(MouseEvent e) { }
```

- Classe `AreaDeDesenho` (6)

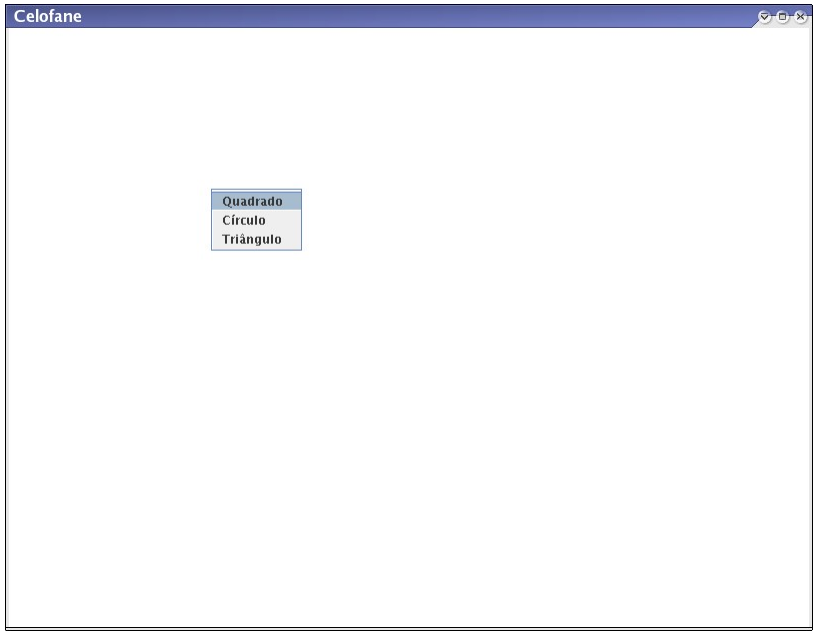
```
public void actionPerformed(ActionEvent e)
  {
  if (e.getSource() instanceof JMenuItem)
    {
    JMenuItem item = (JMenuItem)e.getSource();
    if (item == quadrado) objetos.add(new ObjetoQuadrado(x,y,80));
    if (item == círculo) objetos.add(new ObjetoCirculo(x,y,40));
    if (item == triângulo) objetos.add(new ObjetoTriangulo(x,y,80));
    repaint();
    }
  }

}
```

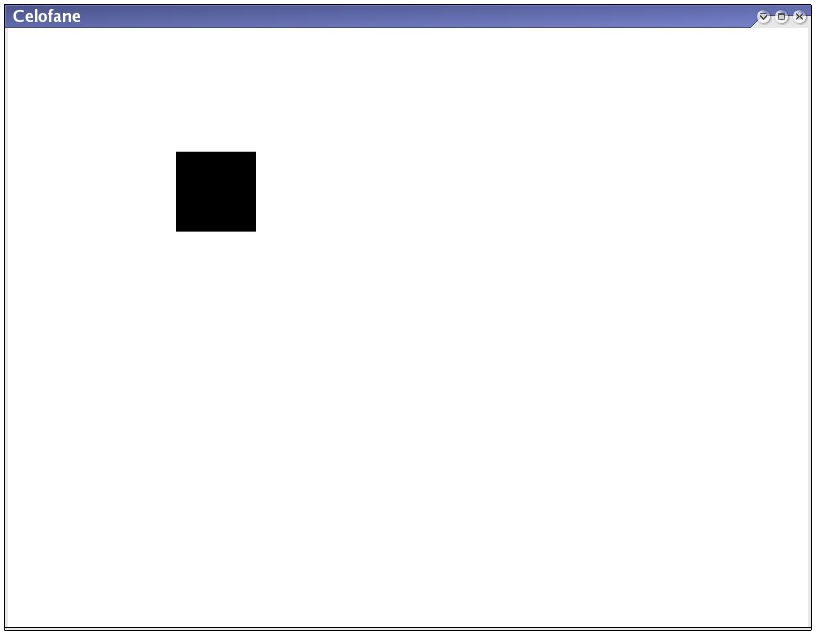
Celofane

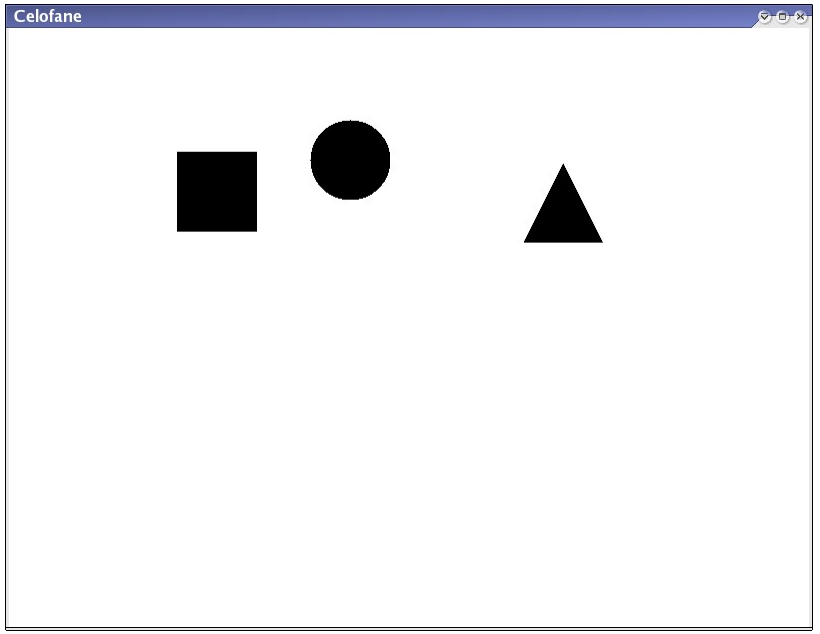
- Classe `Celofane`

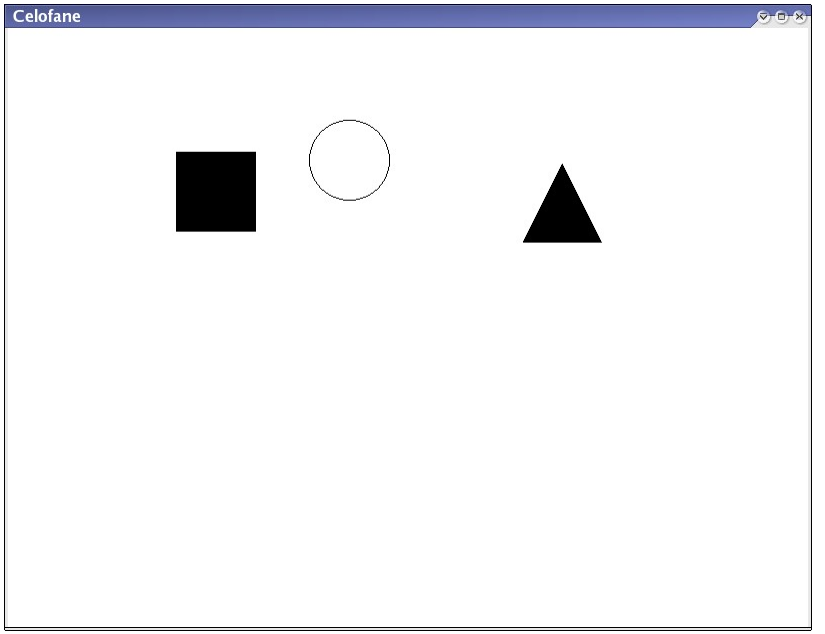
```
package celofane;

import javax.swing.JFrame;

public class Celofane
  {
  public static void main(String[] args)
    {
    AreaDeDesenho área = new AreaDeDesenho(800,600);
    JFrame f = new JFrame("Celofane");
    f.getContentPane().add(área);
    f.pack();
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
  }
```

Celofane

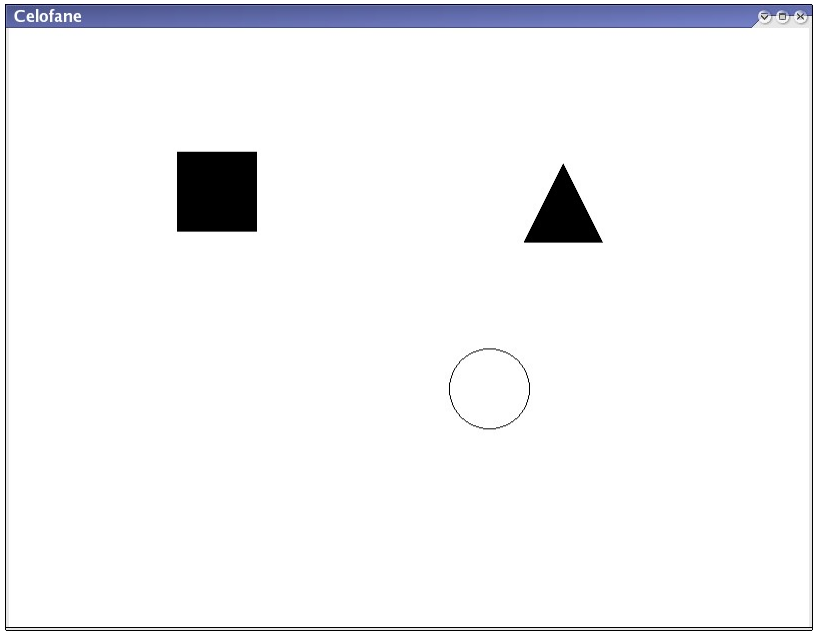

Celofane
Quadrado
Círculo
Triângulo

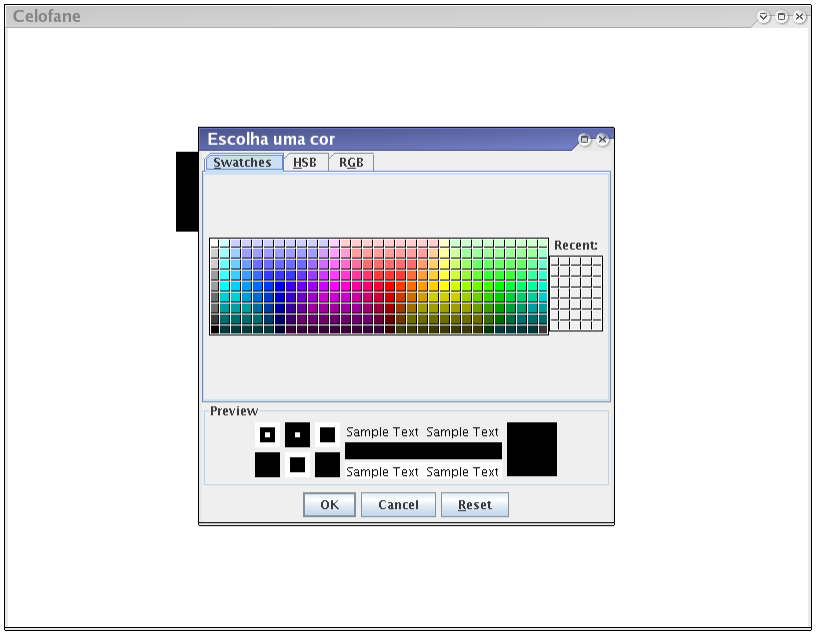

Celofane

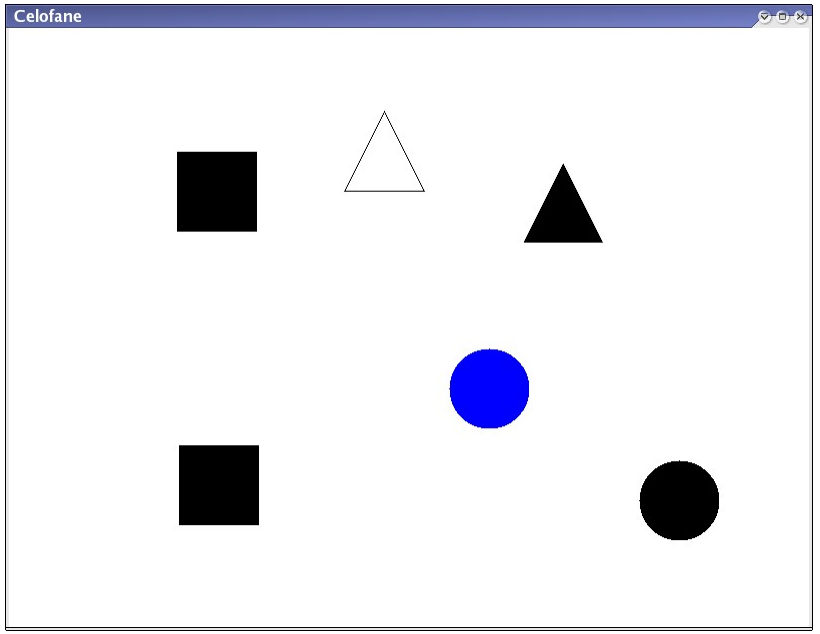

Celofane

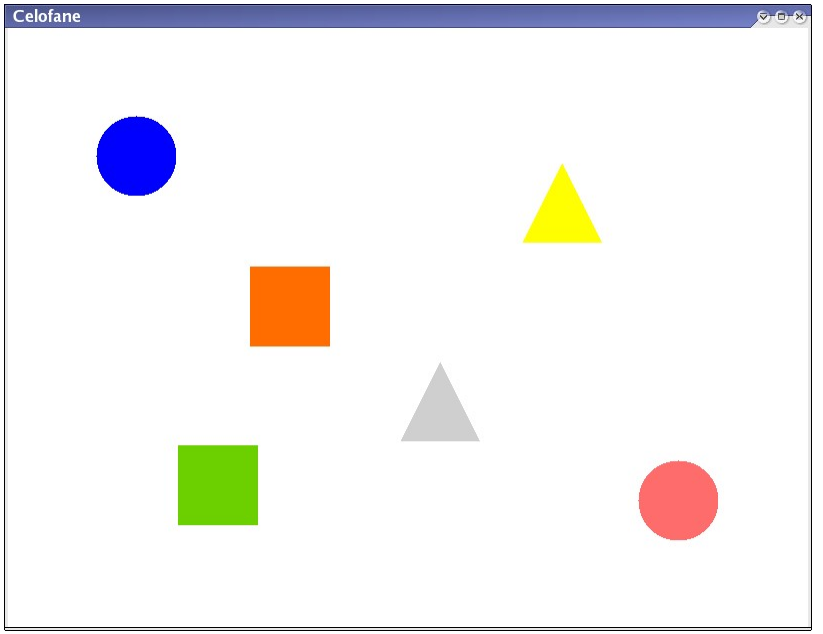

Celofane

Celofane


Celofane


Celofane
Escolha uma cor
Swatches
HSB
RGB
Recent:
Preview
Sample Text Sample Text
Sample Text Sample Text
OK
Cancel
Reset


Celofane


Celofane

Celofane

- Como está, não é interessante, mas...
- É simples criar um ambiente com mais interatividade (mesmo que usando somente mouse!)
- Sugestões de projetos

Exemplo: Aquário

Aquário

- Exemplo simples de *applet* com animação e *sprites*.
- Classe `Peixe` contém imagem e coordenadas e métodos para modificar as coordenadas.
- Classe `ComponenteAquario` contém um componente para desenhar um fundo, desenhar peixes e movimentá-los.
- Classe `Aquario` é uma *applet* que usa este componente.

Aquário

- Classe `Peixe`

```
package aquario;

import java.awt.Dimension;
import java.awt.Graphics2D;
import javax.swing.ImageIcon;

public class Peixe
  {
  private ImageIcon fish;
  private int x,y; // posição
  private int wf,hf; // tamanho peixe
  private int wa,ha; // tamanho aquário
  private float speed;

  public Peixe(String filename,Dimension aq,float s)
    {
    fish = new ImageIcon(filename);
    wf = fish.getIconWidth(); hf = fish.getIconHeight();
    wa = aq.width; ha = aq.height;
    speed = s;
    x = (int)(Math.random()*wa);
    y = (int)(Math.random()*(ha-hf));
    }
```

Veja o Slide 71!

Aquário

- Classe `Peixe`

```
public void move()
  {
  x -= speed;
  if (x < -wf)
    {
    x = wa;
    y = (int)(Math.random()*(ha-hf));
    }
  }

protected void paint(Graphics2D g2d)
  {
  g2d.drawImage(fish.getImage(),x,y,null);
  }

}
```

Aquário

- Classe `ComponenteAquario`

```
package aquario;

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

public class ComponenteAquario extends JComponent implements ActionListener
  {
  private Peixe vermelho;
  private Peixe amarelo;
  private ImageIcon background;
  private Timer tt;
  public ComponenteAquario()
    {
    background = new ImageIcon("aquarium_bg.jpg");
    Dimension area = new Dimension(background.getIconWidth(),
                                   background.getIconHeight());
    vermelho = new Peixe("red_fish.png",area,2);
    amarelo = new Peixe("yellow_fish.png",area,5.5f);
    tt = new Timer(20,this);
    tt.setCoalesce(true);
    tt.start();
    }
```

Veja o Slide 71!

Aquário

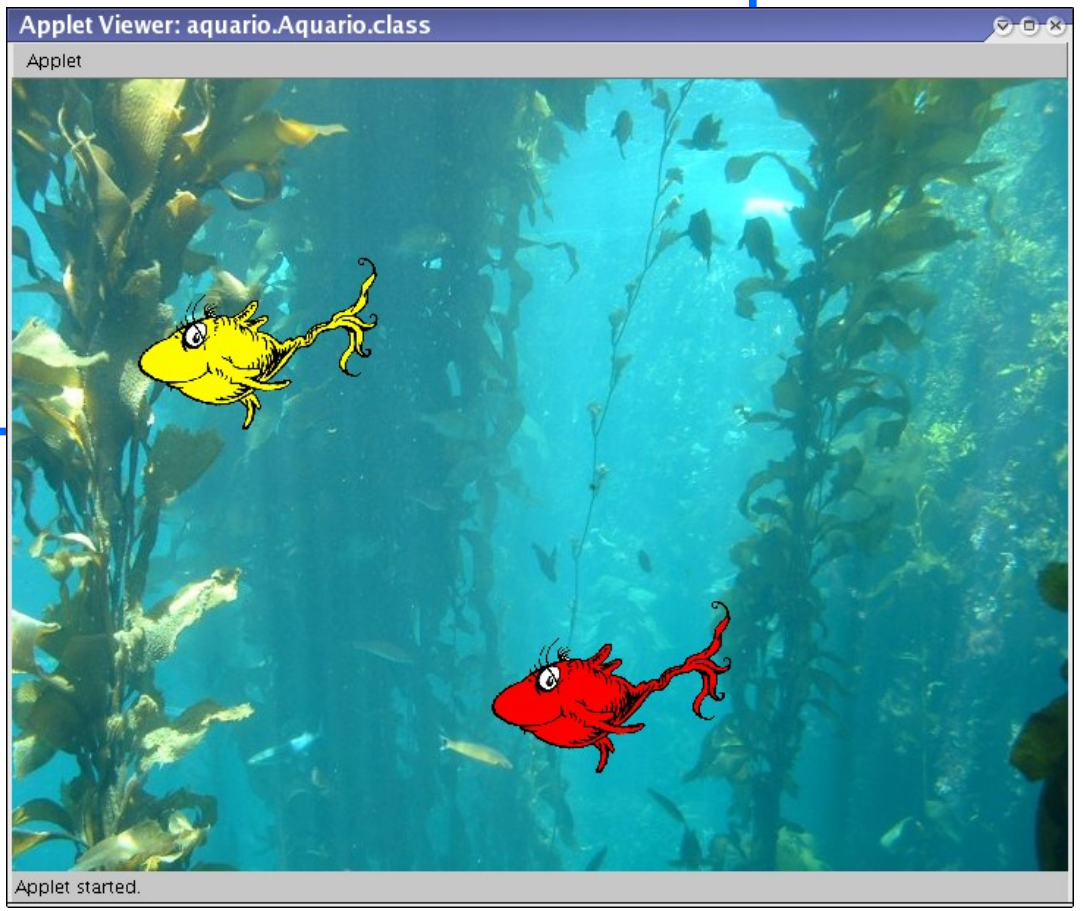
- Classe `ComponenteAquario`

```
protected void paintComponent(Graphics g)
  {
  Graphics2D g2d = (Graphics2D)g;
  g2d.drawImage(background.getImage(),0,0,null);
  vermelho.paint(g2d);
  amarelo.paint(g2d);
  }

public void actionPerformed(ActionEvent e)
  {
  vermelho.move(); amarelo.move(); repaint();
  }

}
```

Aquário

- Classe `Aquario`

```
package aquario;

import javax.swing.JApplet;

public class Aquario extends JApplet
  {
  private ComponenteAquario c;
  public void init()
    {
    c = new ComponenteAquario();
    add(c);
    resize(720,540);
    }
  }
```



Aquário

- Problemas com a classe `ComponenteAquario`: Desenho de todo o componente pode ser custoso!
- Solução: desenhar somente as áreas modificadas.
  - Na classe `Peixe`:

```
public Rectangle getEnvolvente()
  {
  return new Rectangle(x,y,wf,hf);
  }
```

  - Na classe `ComponenteAquario`:

```
public void actionPerformed(ActionEvent e)
  {
  vermelho.move(); amarelo.move();
  repaint(vermelho.getEnvolvente());
  repaint(amarelo.getEnvolvente());
  }
```

Finalizando...

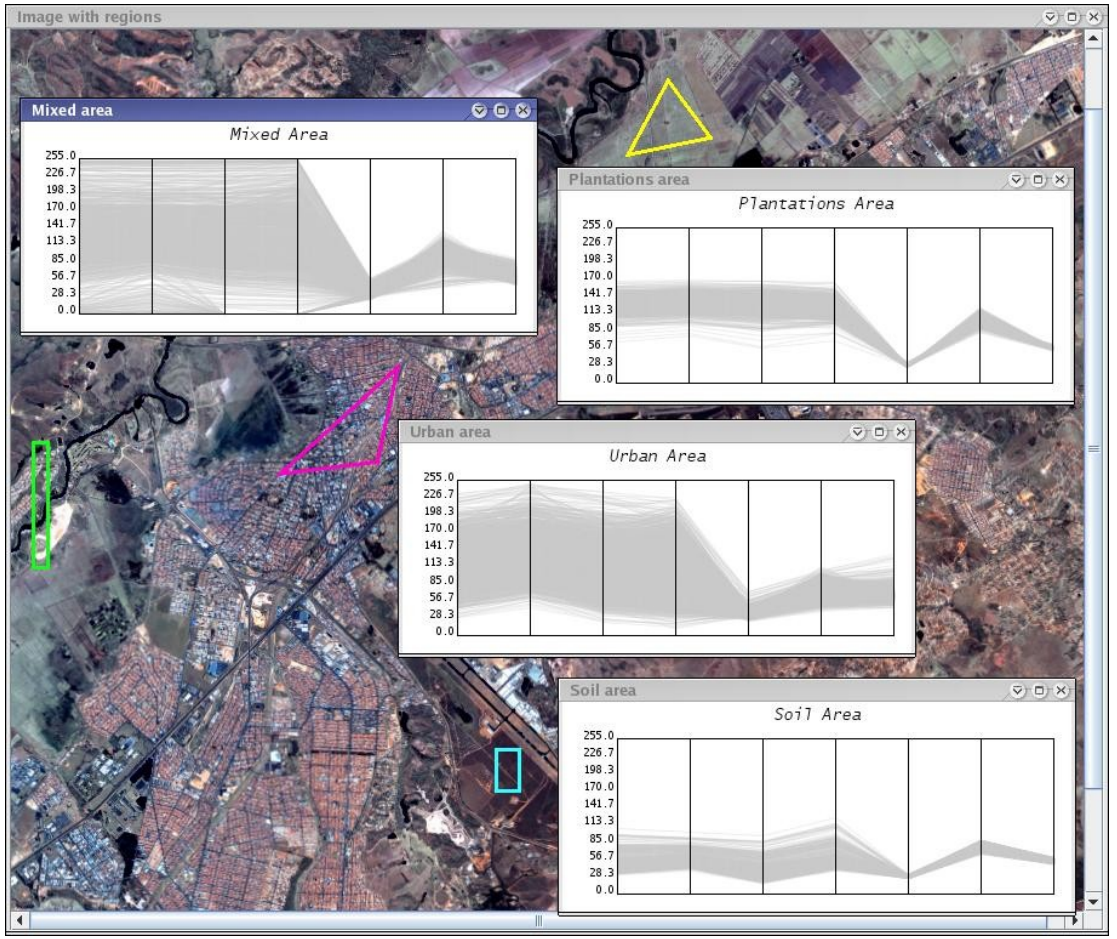
Vimos

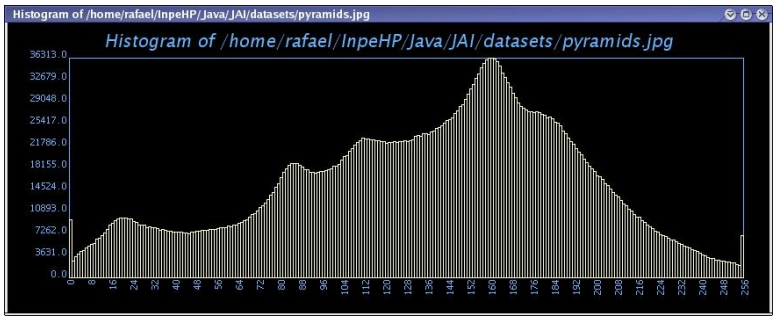
- Algumas noções de programação com interfaces gráficas em Java.
- Como usar alguns componentes comuns.
- Como usar eventos em suas aplicações.

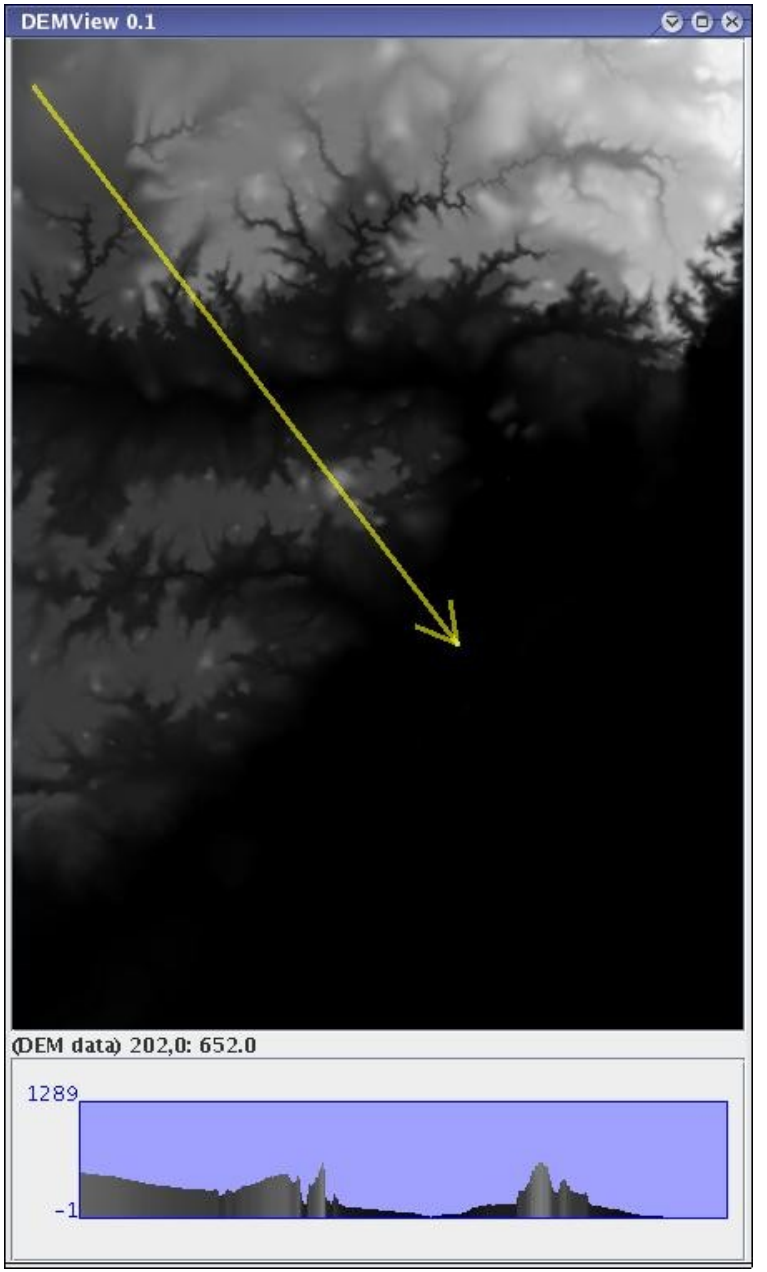
Não vimos

- **Muita** coisa!
- Formalização do modelo **MVC**.
- Exemplos de componentes para visualização e interação complexas.
- Aplicações com estes componentes
  - Alguns estão espalhados por http://www.lac.inpe.br/~rafael.santos
- Componentes baseados em *Filthy Rich Clients*.

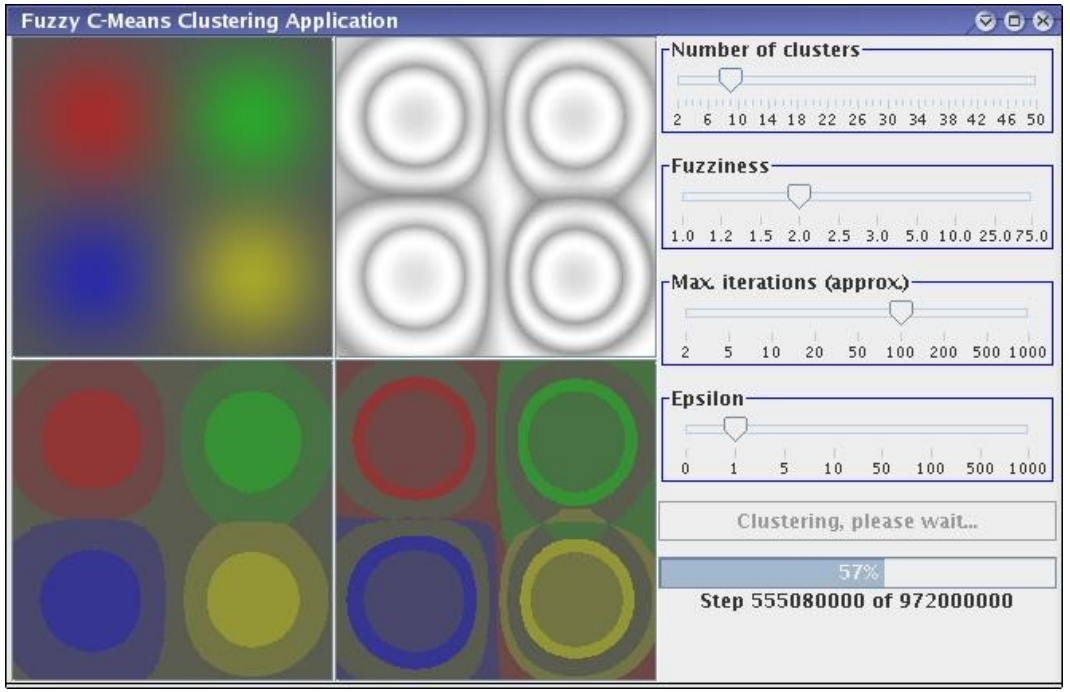
Exemplos de componentes (para PI)

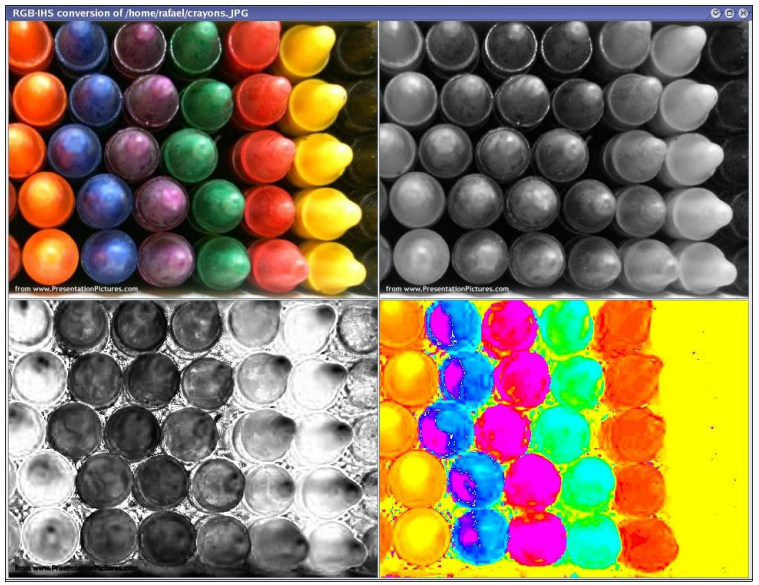


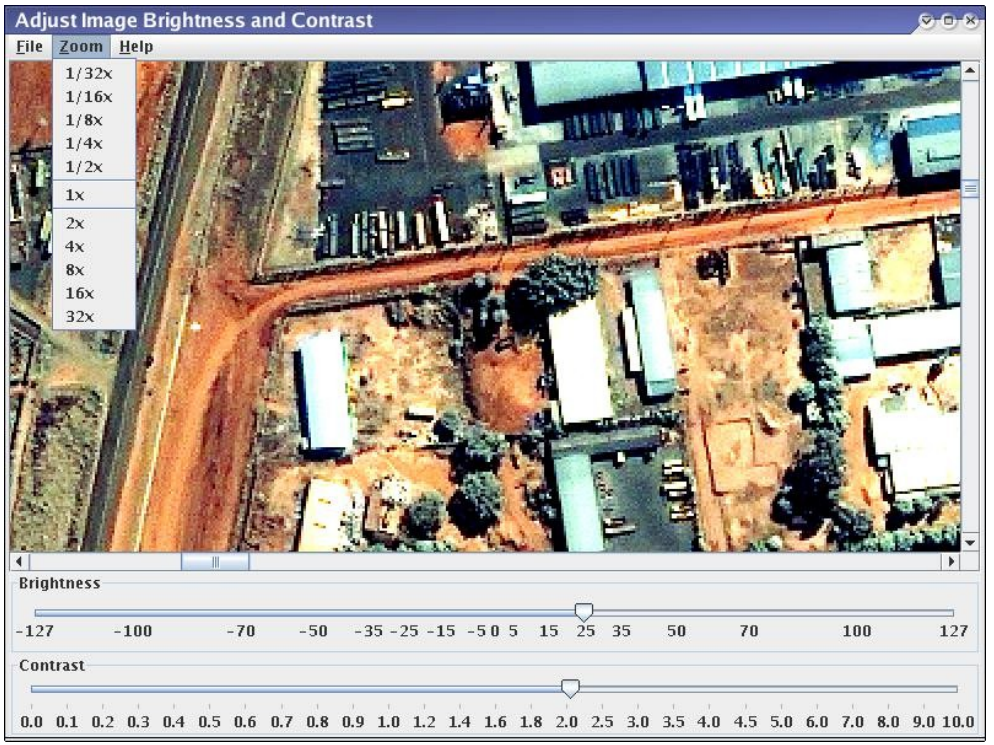




Exemplos de componentes (para PI)



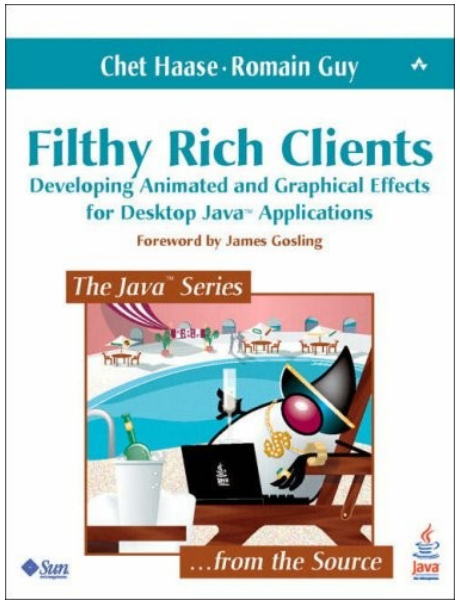




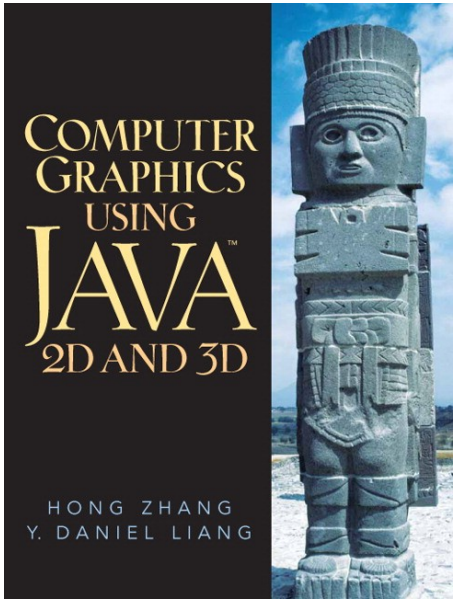
Referências

- Java Tutorial: http://java.sun.com/docs/books/tutorial/index.html
- *Java SE Desktop Articles*: http://java.sun.com/javase/technologies/desktop/articles.jsp
- Material em http://www.lac.inpe.br/~rafael.santos

Referências

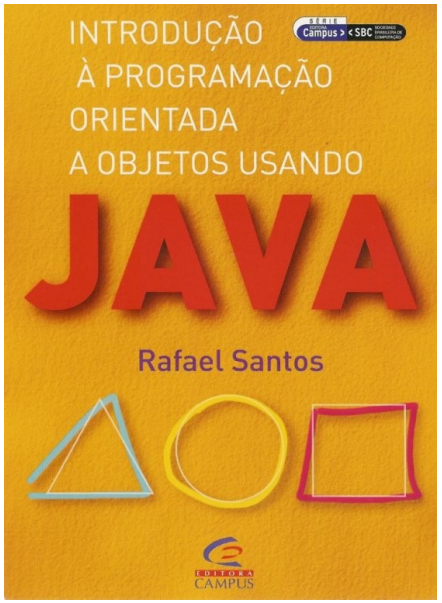


- *Filthy Rich Clients*; Chet Haase, Romain Guy; Prentice-Hall/Sun, 2007, 608pp.
  - Interfaces ricas em Swing, *threads*, animação de interfaces, temporização, processamento de imagens (para tela).



- *Computer Graphics Using Java 2D and 3D*; Hong Zhang, Y. Daniel Liang; Prentice-Hall, 2007, 632pp.
  - Gráficos, Java2D, renderização, Java3D.

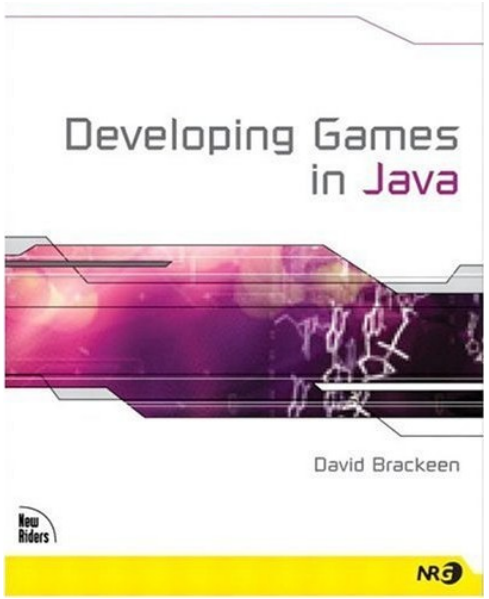
Referências

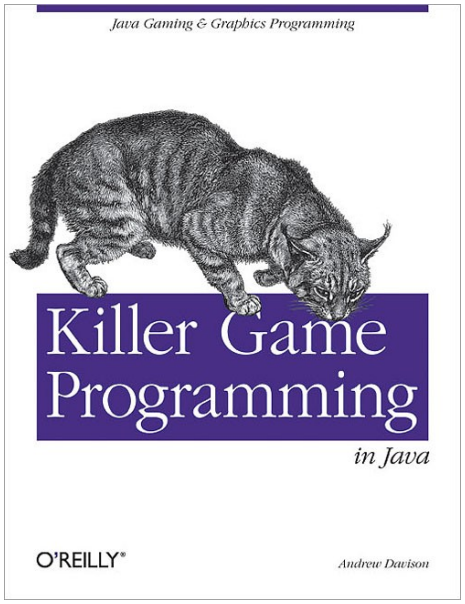


- *Introdução à Programação Orientada a Objetos Usando Java*; Rafael Santos; Campus/SBC, 2003, 352pp.
  - Conceitos básicos: programação, orientação a objetos, modelagem, lógica.

- *Java: Como Programar*; H. M. Deitel, P. J. Deitel; Prentice-Hall, 2005, 1152pp.
  - Muita informação sobre Java e sobre APIs principais.

Referências





- *Developing Games in Java*; David Brackeen; New Riders, 2004, 996pp.
  - *Threads*, 2D e animação, interatividade, áudio, 3D, IA, otimização.
  - Exemplos: *scroller*.
- *Killer Game Programming in Java*; Andrew Davison; O'Reilly, 2005, 970pp.
  - Animação, imagens, sons, sprites, Java3D, sprites 3D, sistemas de partículas, etc.
  - Exemplos: worms, side-scroller, isométrico, labirinto 3D, FPS.
  - fivedots.coe.psu.ac.th/~ad/jg/

Obrigado!

Perguntas?