Introdução à Programação de Aplicações Científicas em Java

Escola de Verão do Laboratório Associado de Computação e Matemática Aplicada do INPE

Rafael Santos

Sobre o curso

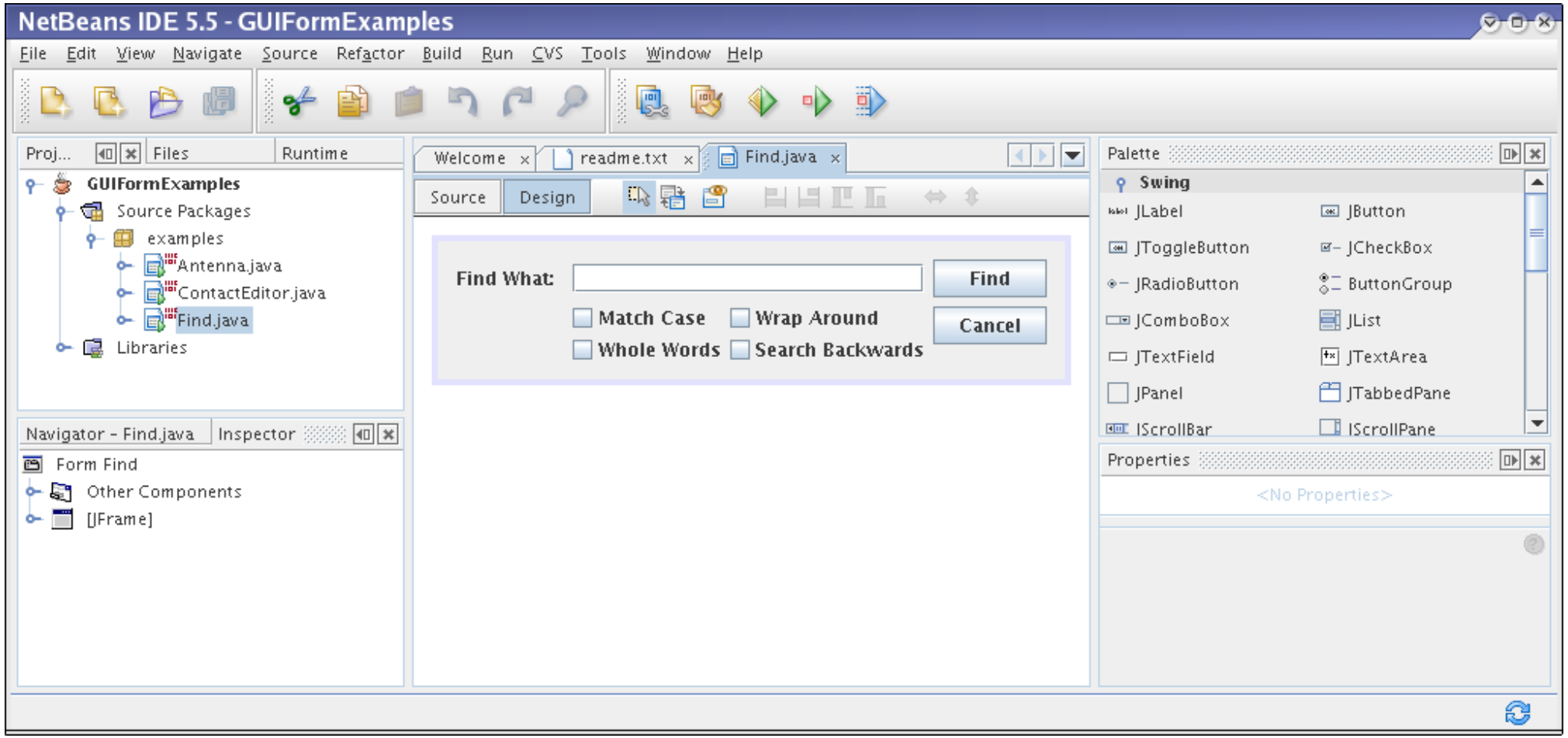
- Dividido em quatro partes:
  1. Tecnologia, Linguagem e Orientação a Objetos.
  2. *API*s comuns.
  3. **Programação com Interfaces Gráficas.**
  4. *API*s para Processamento Científico (aplicações diversas).
- Cada parte tem aproximadamente 3 horas.
- O curso é somente teórico, sem laboratório.
  - Exemplos, *links*, etc. serão disponibilizados em http://www.lac.inpe.br/~rafael.santos
- Muitos exemplos formatados como *how-to*.

Por que usar interfaces gráficas?

- Interfaces gráficas ≈ Interfaces com Usuário (IUs).
- Por que usar interfaces gráficas?
  - Permitem maior interatividade com usuário, ex. modificação de parâmetros e condições de execução.
  - Visualização de resultados.
  - Familiaridade das ferramentas.
- Por que *não usar* interfaces gráficas?
  - Complexidade adicional.
  - Podem não ser adequadas, ex. cliente/servidor sobre WWW, processamento em lote, execução remota por terminal, *scripts*.

Interfaces gráficas e IDEs

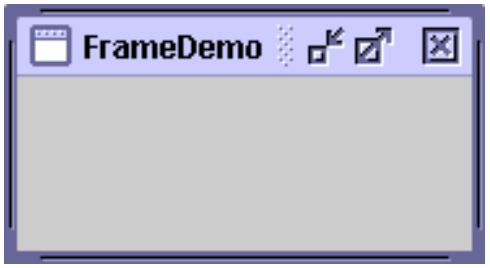
- A maioria das IDEs permite a criação de interfaces gráficas (aplicações, *applets*, diálogos) por composição visual.
  - Realmente útil para IUs complexas.



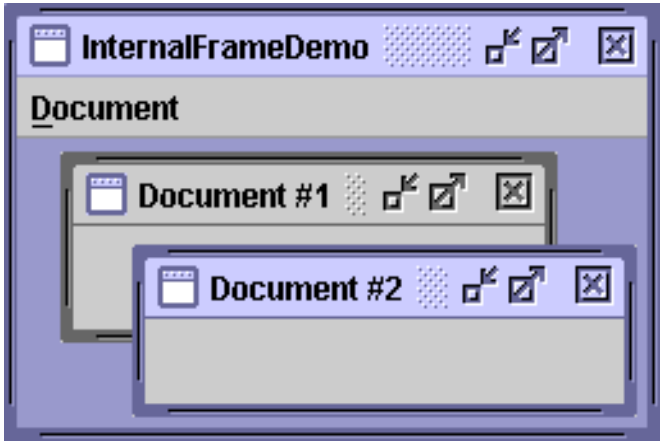
Interfaces gráficas e IDEs

- É **importante** saber como código relacionado funciona para compreender o que a IDE escreve!
- Escreveremos código “na mão”: componentes da IU serão instanciados e posicionados no código.
  - Em IDEs posicionamos visualmente e o código correspondente é gerado pela IDE.
- Abordagem manual, mais trabalhosa, mas independente de IDEs.
  - Até editores de texto comum podem ser usados!

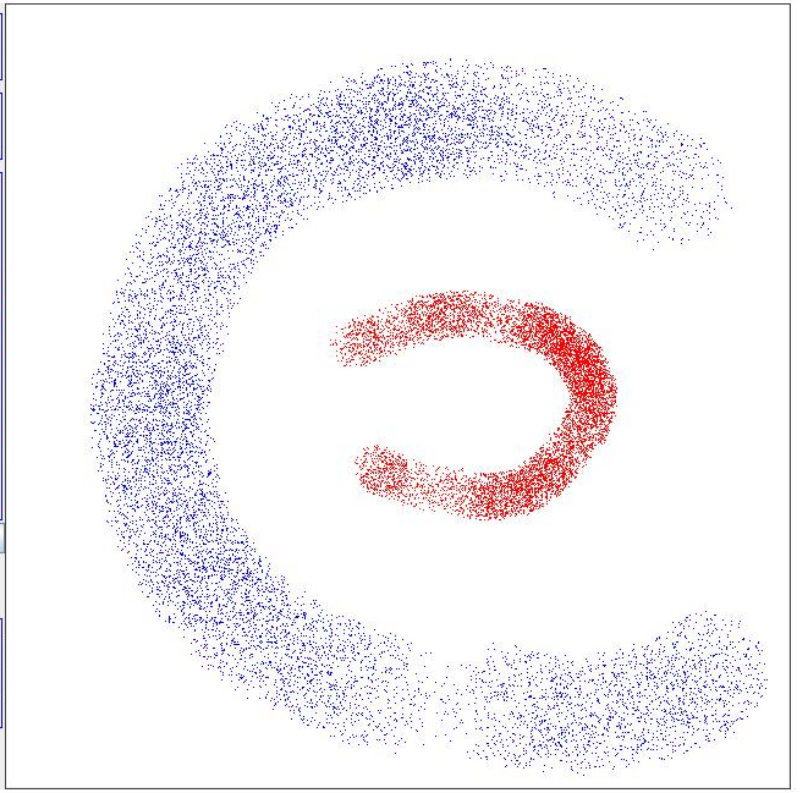
Principais tipos de IUs



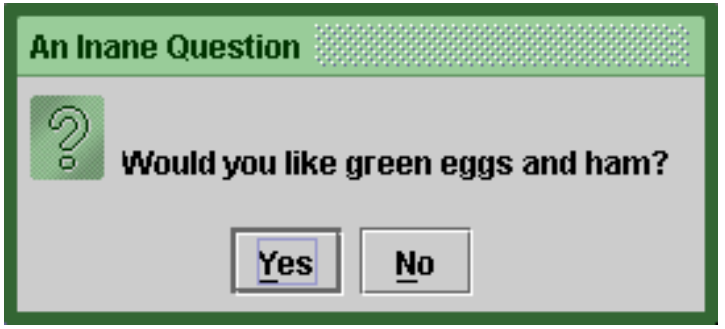
Frame



Frame Interna (MDI)



Applet



Diálogo

Adaptado do *Java Tutorial*: http://java.sun.com/docs/books/tutorial/index.html

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 para JFrames podem ser executados pela nossa classe.
  - Podemos sobreescrever métodos com comportamento específico.

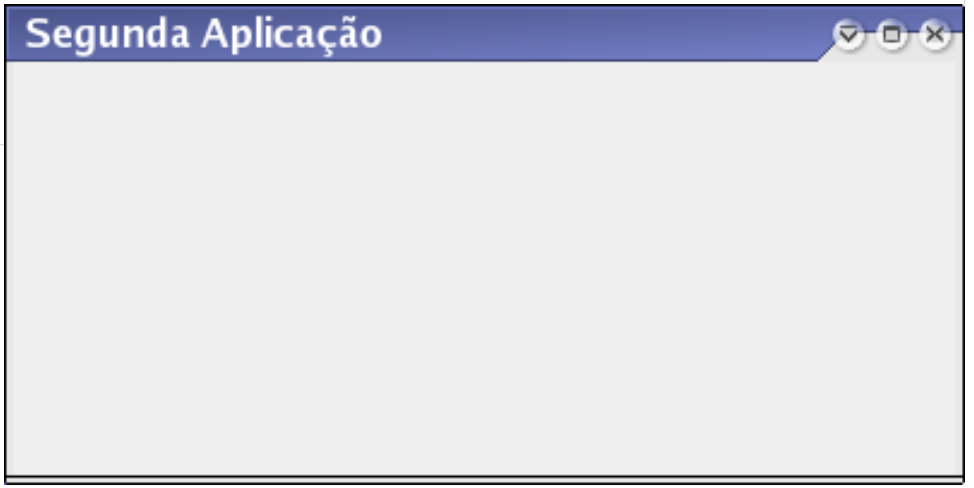
Criando uma janela gráfica (*Frame*)

```
import javax.swing.JFrame;

public class PrimeiraAplicacao extends JFrame
   {

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

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

- Ao executar o código, nada aparece!
- Faltou executar métodos que definem aparência e comportamento básico da aplicação!

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();
     }
   }
```

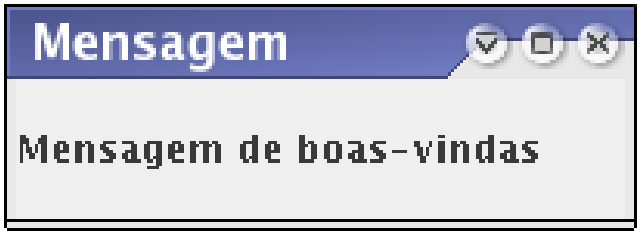


Criando uma janela gráfica (*Frame*)

```
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*)

```
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

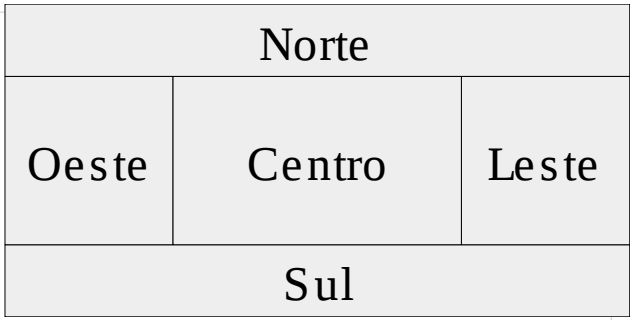
- Existem vários tipos de componentes de interfaces com usuários
  - Customizáveis por chamada a métodos de instâncias ou herança.
- Para criar aplicações com interfaces gráficas:
  - Criamos instâncias das classes dos componentes.
  - Modificamos atributos destas instâncias.
  - Adicionamos estas instâncias 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 um *layout* diferente ou mesmo 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.

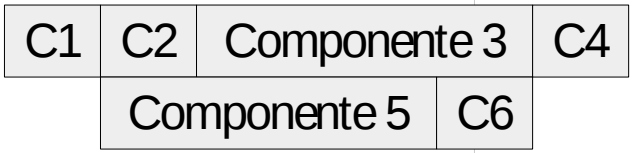
Alguns *layouts* gráficos: *BorderLayout*

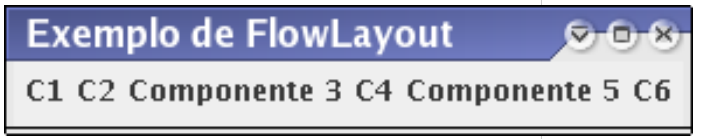
```
import java.awt.*;
import javax.swing.*;

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();
    }
  }
```

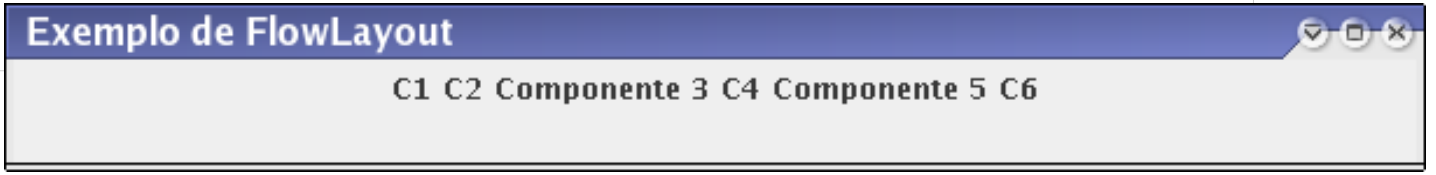


Alguns *layouts* gráficos: *FlowLayout*

```
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();
     }
   }
```



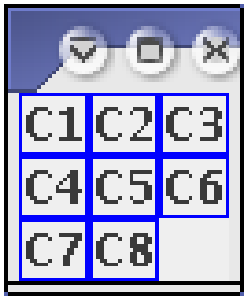


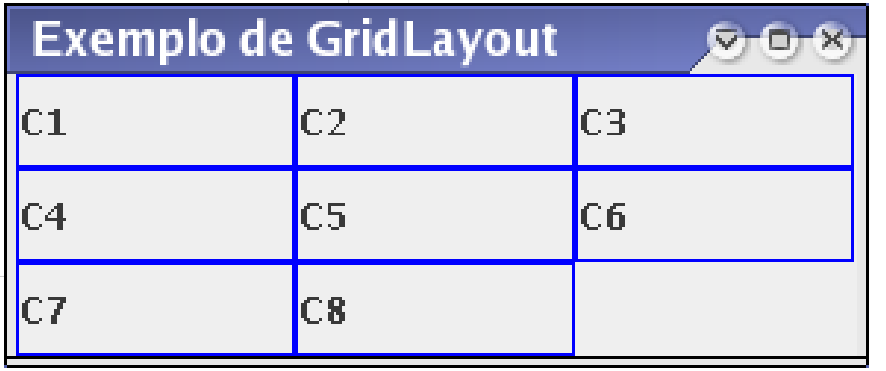


Alguns *layouts* gráficos: *GridLayout*

```
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

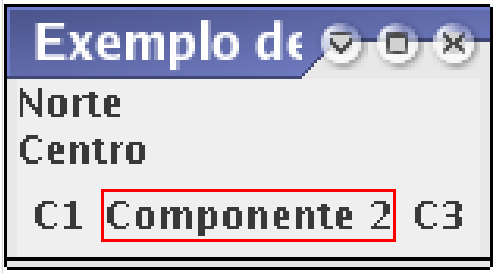
- *BoxLayout*: permite arranjar componentes em uma única linha ou coluna.
- *CardLayout*: permite empilhar conjuntos de componentes na direção Z.
- *GridBagLayout*: permite arranjo de componentes com proporções diferentes.
- *SpringLayout*: permite arranjar componentes relativamente uns aos outros.

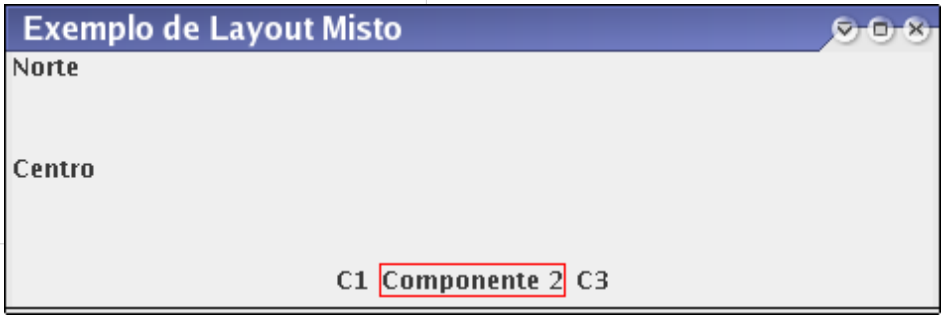
Layouts podem ser combinados!

- 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();
    }
  }
```



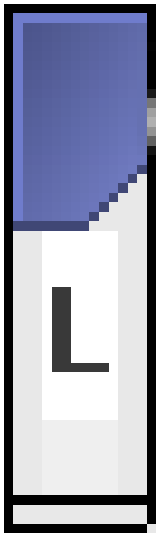


Layout nulo

- 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

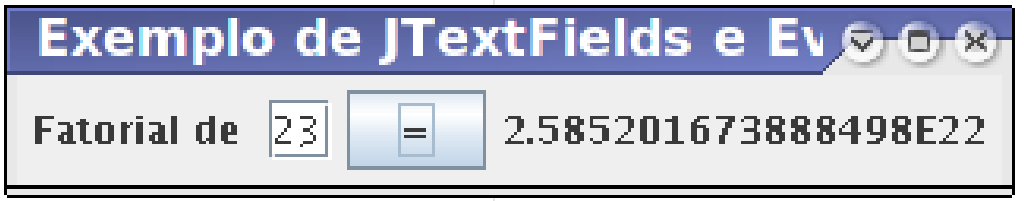
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ExJButton2 extends JFrame implements ActionListener
  {
  private JButton j1,j2;
  private int contador = 0;
  private JLabel lContador;
  public ExJButton2()
    {
    super("Exemplo de JButtons e Eventos");
    Container 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 JTextField

```
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 JTextField

```
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);
   pack();
   }

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

```
  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();
    }
  }
```

Exemplo de JRadioButtons

○ eclipse01.png ○ eclipse02.png ○ eclipse03.png
○ eclipse04.png ○ eclipse05.png ○ eclipse06.png
○ eclipse07.png ○ eclipse08.png ○ eclipse09.png

Exemplo de JRadioButtons

○ eclipse01.png ○ eclipse02.png ○ eclipse03.png
○ eclipse04.png ○ eclipse05.png ○ eclipse06.png
○ eclipse07.png ● eclipse08.png ○ eclipse09.png

Exemplo de JRadioButtons

○ eclipse01.png ○ eclipse02.png ○ eclipse03.png
○ eclipse04.png ○ eclipse05.png ● eclipse06.png
○ eclipse07.png ○ eclipse08.png ○ eclipse09.png

Componentes e eventos: JSlider

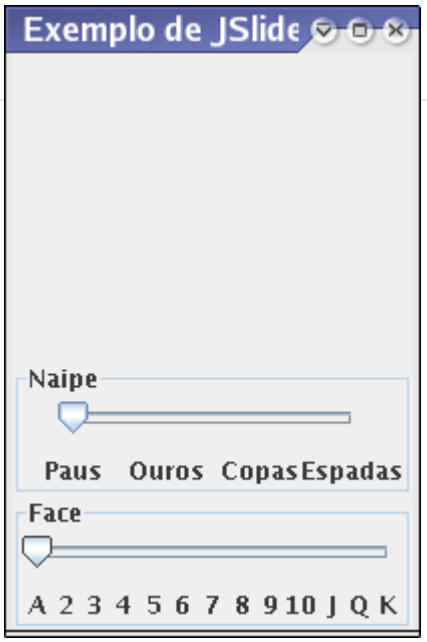
```
import java.awt.*;
import java.util.Hashtable;
import javax.swing.*;
import javax.swing.event.*;

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);
    }
```

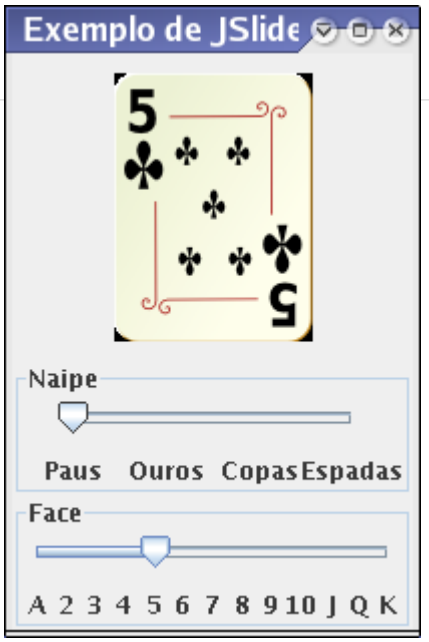
Componentes e eventos: JSlider

```
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

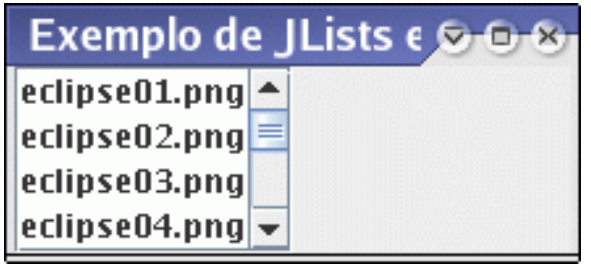
```
public void stateChanged(ChangeEvent e)
   {
   int iNaipe = -1;
   char sNaipe = naipe.getValue().toString().charAt(0);
   iNaipe = "POCE".indexOf(sNaipe);
   int iFace = -1;
   char sFace = face.getValue().toString().charAt(0);
   iFace = "A234567891JQK".indexOf(sFace);
   String nome = String.format("%d-%02d.png",new 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);
```

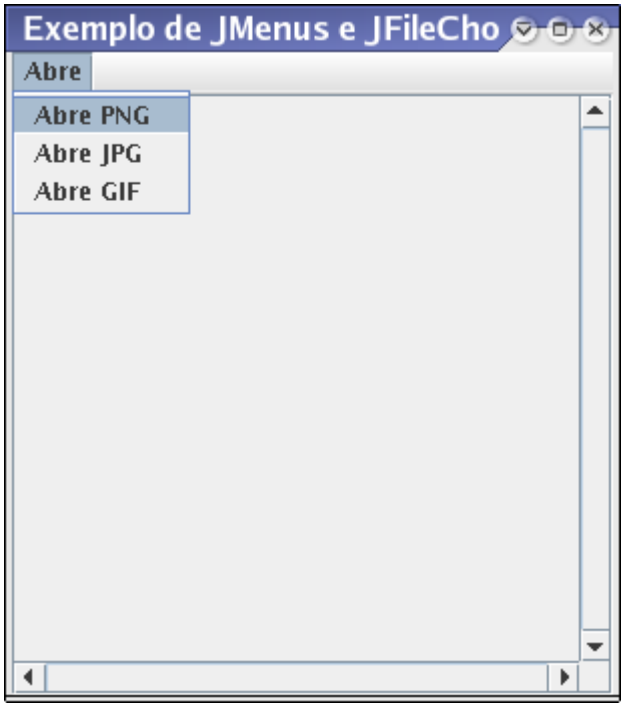
Componentes e eventos: JMenu

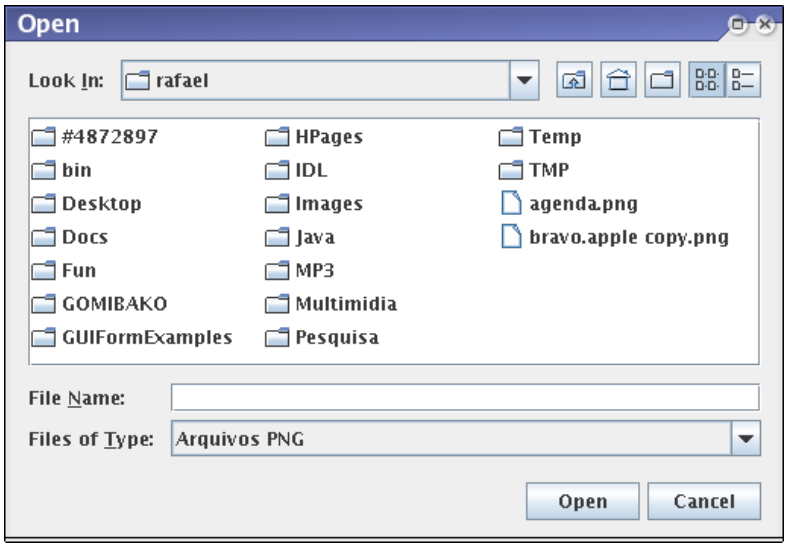
```
    imagem = new JLabel();  imagem.setHorizontalAlignment(SwingConstants.CENTER);
    scrollImagem = new JScrollPane(imagem,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                                          JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
    scrollImagem.setPreferredSize(new Dimension(300,300));
    c.add(scrollImagem);
    pack();     setVisible(true);     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
  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();
    }
  }
```

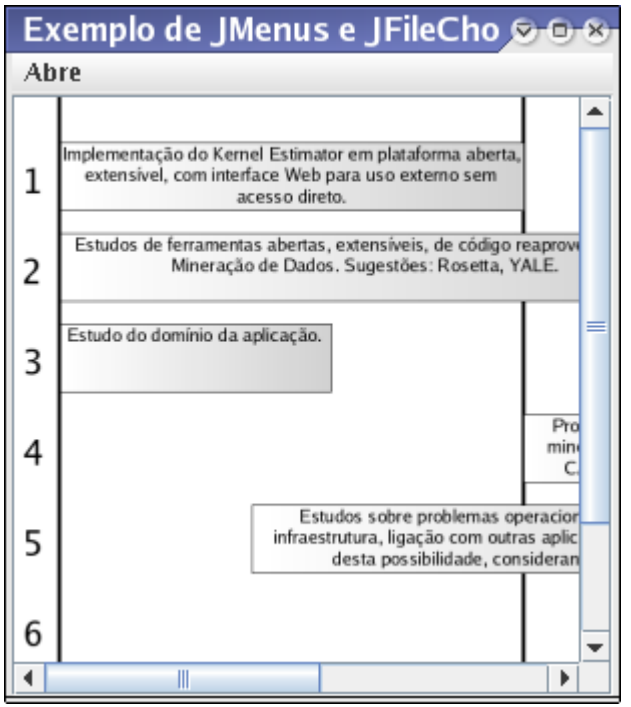
Componentes e eventos: JMenu

```
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);
  }
```

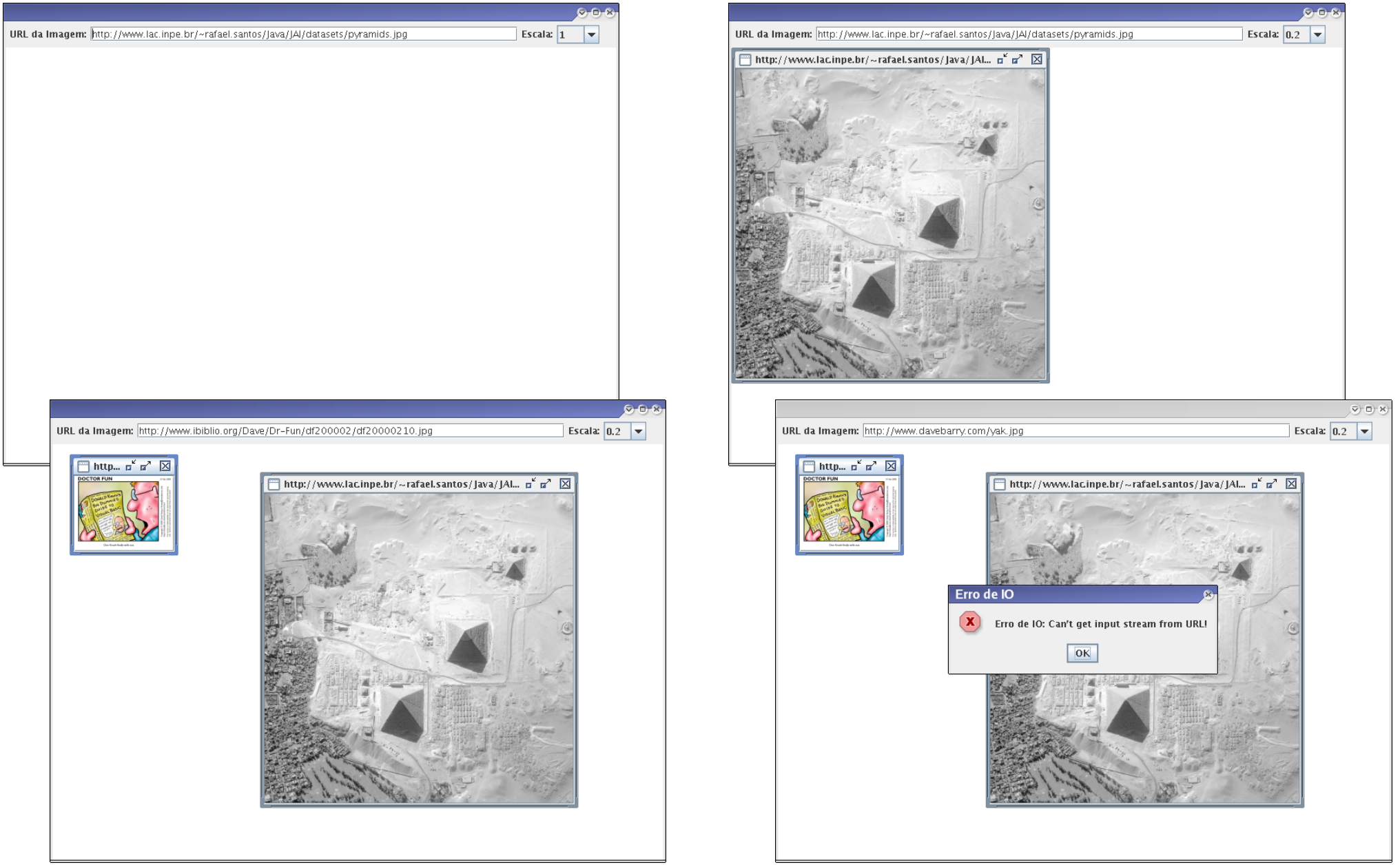
JInternalFrame e MDI

```
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;
    }
```

JInternalFrame e MDI

```
    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);
        }
      }
    }

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

  }
```

JInternalFrame e MDI

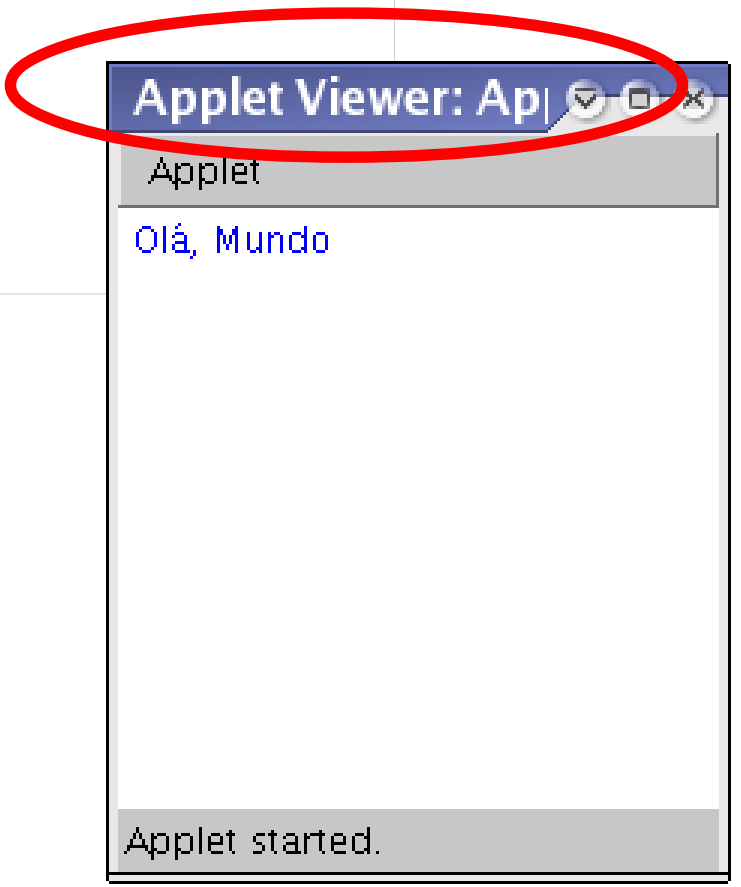


Applets

- 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.

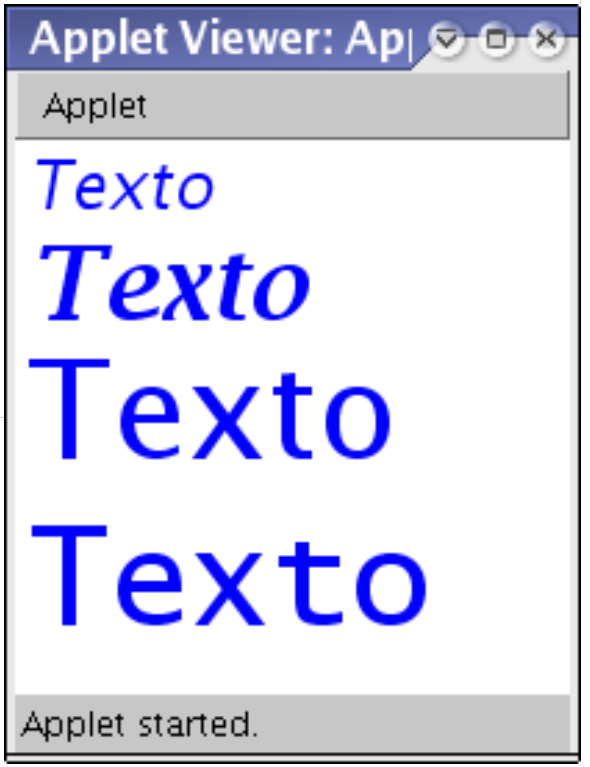
Applets

```
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);
   }

}
```

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(getImage(getCodeBase(),nome));
   carta.setIcon(ícone);
   }
}
```

Criando Novos Componentes

Criando novos componentes

- Pode ser necessário ou interessante criar novos componentes:
  - Para exibição ou entrada de informações especializadas.
  - 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

- Através de herança.
- Exemplo: botão com comportamento ligeiramente diferente.
  - Devemos sobreescrever alguns métodos.
  - Podemos criar novos métodos específicos.

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

public class JButtonPedra extends JButton
  {
  private int tipo;

  public JButtonPedra()
    {
    super();
    tipo = 0;
    }

  public Dimension getMaximumSize() { return getPreferredSize(); }
  public Dimension getMinimumSize() { return getPreferredSize(); }
  public Dimension getPreferredSize() { return new Dimension(40,40); }
```

Criando novos componentes

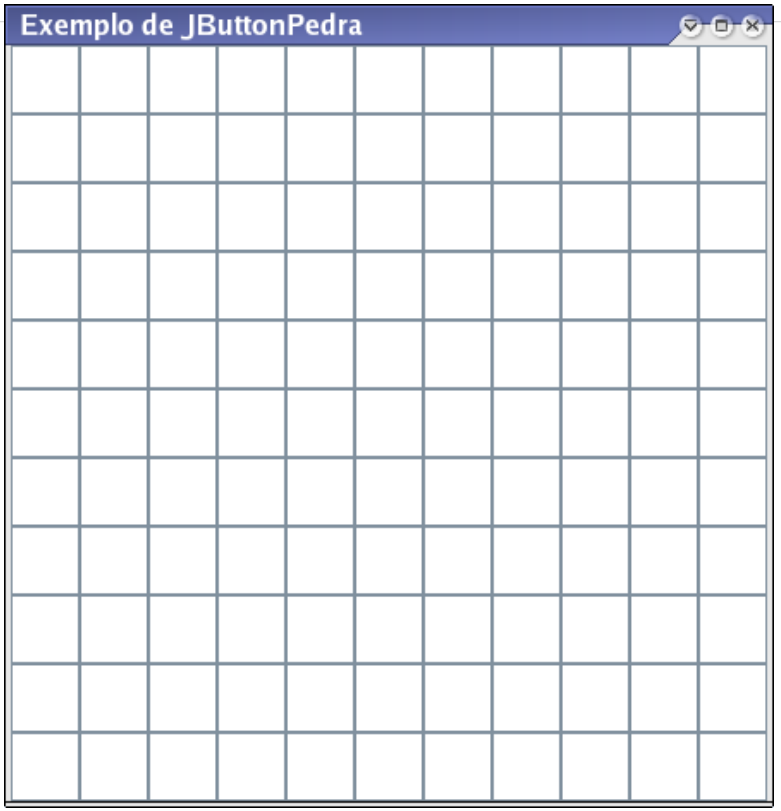
```
public void próximoTipo()
   {
   tipo++; if (tipo > 2) tipo = 0;
   }

protected void paintComponent(Graphics g)
   {
   super.paintComponent(g);
   Graphics2D g2d = (Graphics2D)g;
   g2d.setColor(Color.WHITE);
   g2d.fillRect(0,0,getWidth(),getHeight());
   g2d.setColor(Color.BLACK);
   switch(tipo)
     {
     case 1: g2d.drawOval(6,6,getWidth()-12,getHeight()-12); break;
     case 2: g2d.fillOval(6,6,getWidth()-12,getHeight()-12); break;
     }
   }
}
```

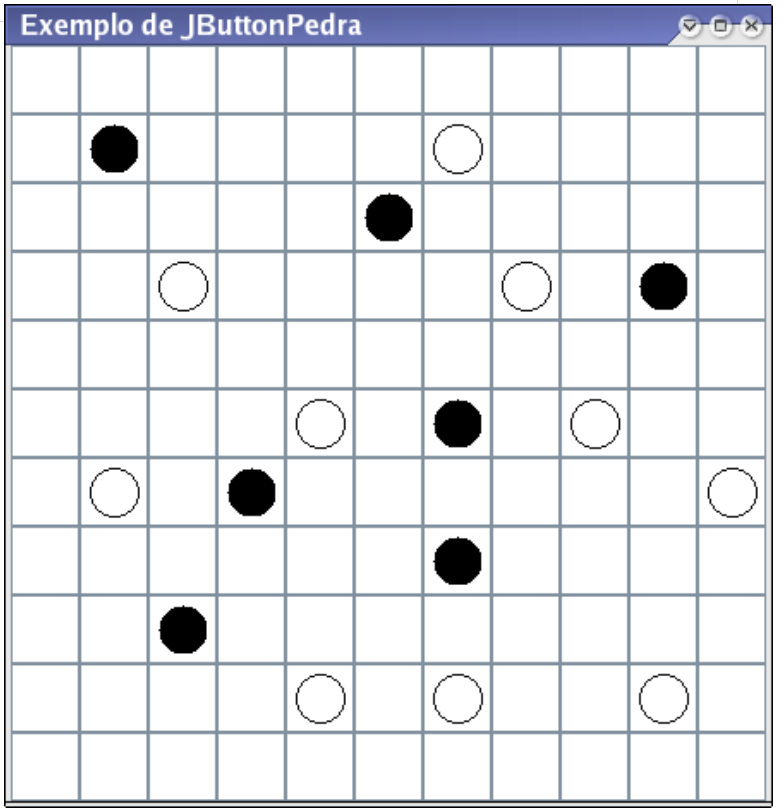
Criando novos componentes

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

public class ExButtonPedra extends JFrame implements ActionListener
  {
  private JButtonPedra[][] jogo;

  public ExButtonPedra()
    {
    super("Exemplo de JButtonPedra");
    Container c = getContentPane();
    c.setLayout(new GridLayout(11,11));
    jogo = new JButtonPedra[11][11];
    for(int col=0;col<11;col++)
      for(int row=0;row<11;row++)
        {
        jogo[col][row] = new JButtonPedra();
        jogo[col][row].addActionListener(this);
        c.add(jogo[col][row]);
        }
    pack();
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

Criando novos componentes

```
public void actionPerformed(ActionEvent e)
   {
   JButtonPedra b = (JButtonPedra)e.getSource();
   b.próximoTipo();
   }
public static void main(String[] args)
   {
   new ExButtonPedra();
   }
}
```





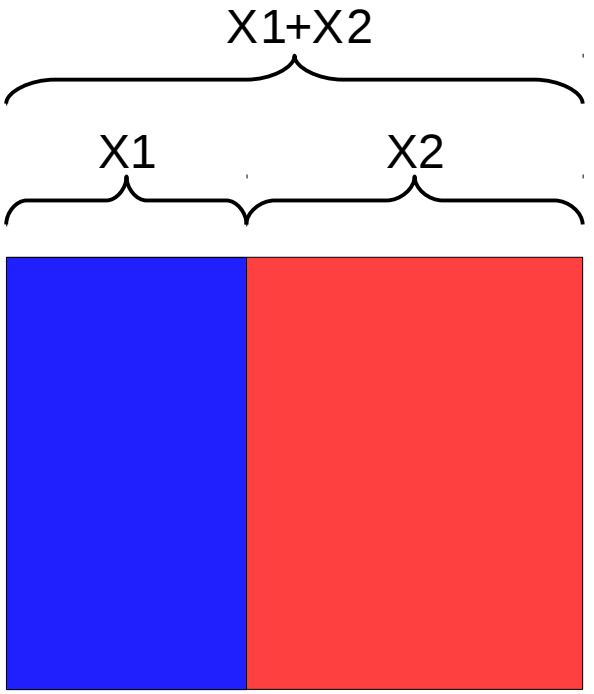
Criando novos componentes

- Geralmente, sobrepor métodos
  - paintComponent(Graphics g): **sempre**!
  - getMaximumSize(), getMinimumSize(), getPreferredSize(): sempre que quisermos impor regras sobre tamanhos.
- Usar construtor para passar atributos adicionais.
- Usar classe/componente já existente quando comportamento for similar.

Criando novos componentes

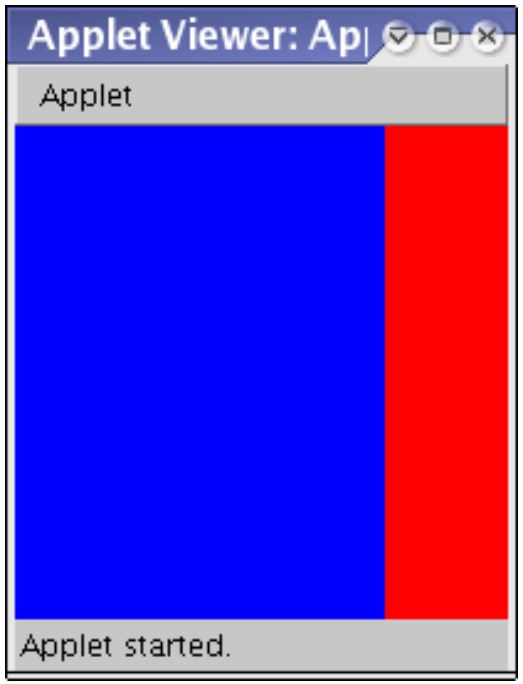
- Herdando de JComponent:
- Mostra duas barras de larguras proporcionais.

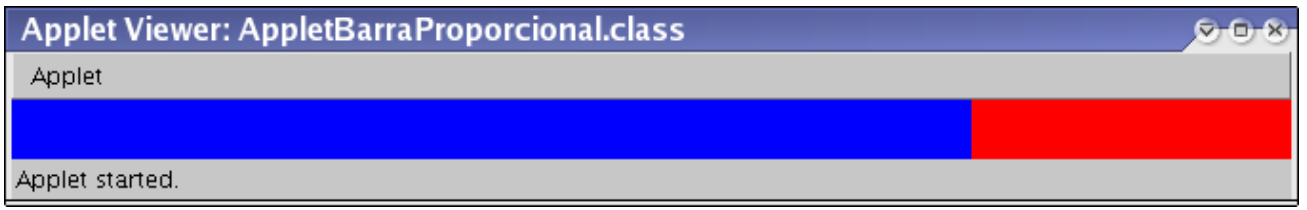
```
import java.awt.*;
import javax.swing.JComponent;

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());
     }

   }
```

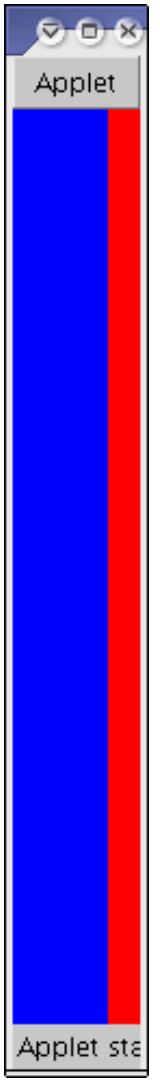


Criando novos componentes

```
import javax.swing.JApplet;

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

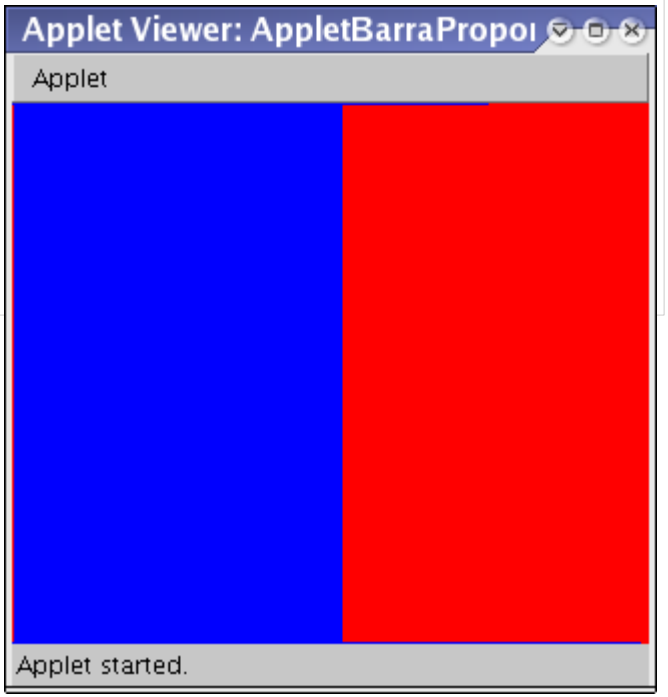






Criando novos componentes

```
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!

Criando novos componentes

- Componentes podem processar seus próprios eventos.

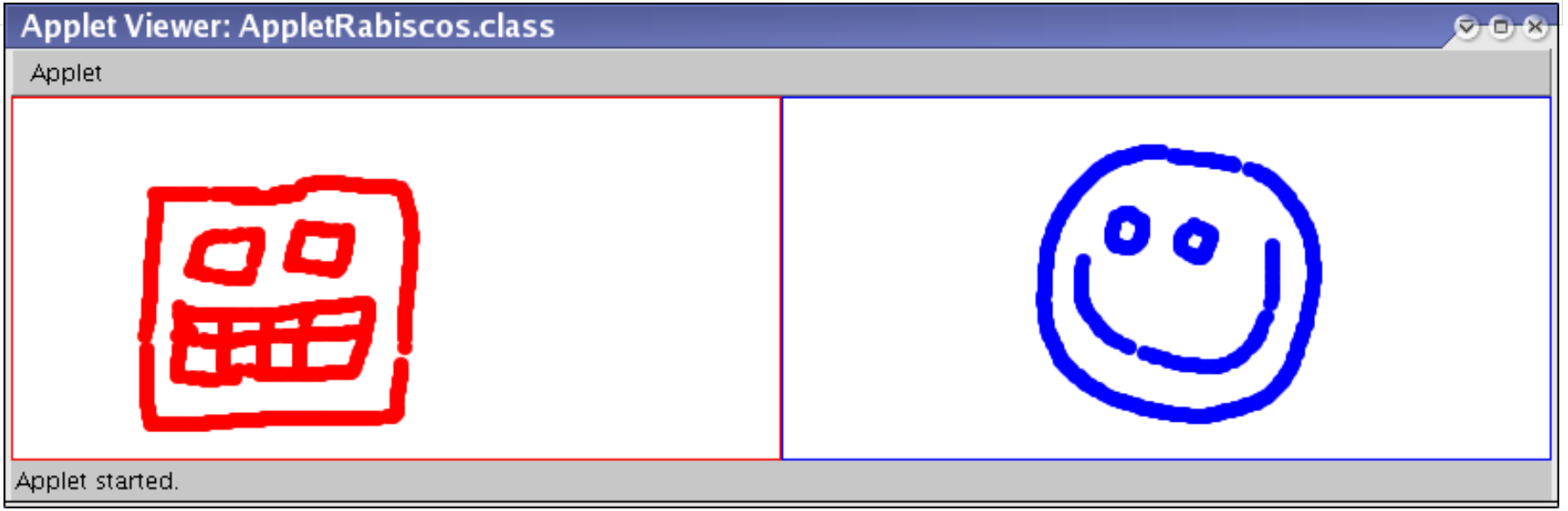
```
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import javax.swing.JComponent;

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.

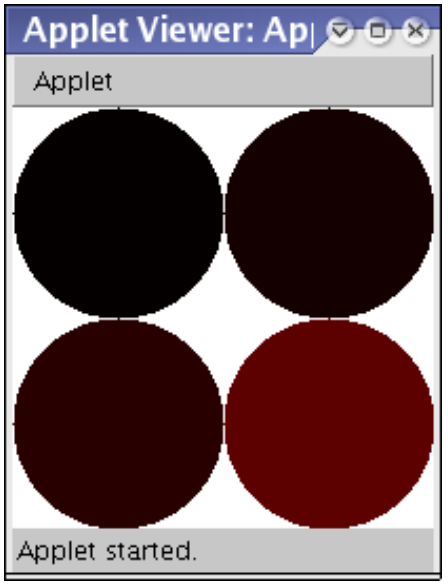
```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JComponent;
import javax.swing.Timer;

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();
    }
```

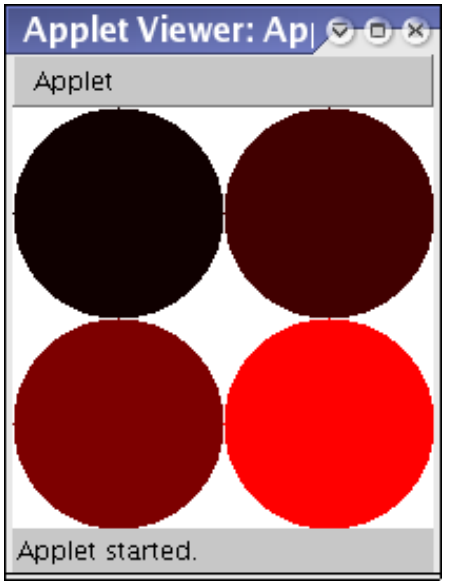
Criando novos componentes

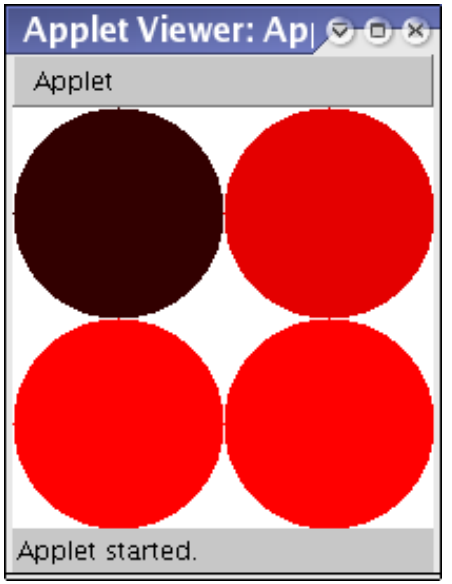
```
protected void paintComponent(Graphics g)
   {
   g.setColor(Color.WHITE);
   g.fillRect(0,0,getWidth(),getHeight());
   // Calculamos a cor de acordo com o passo.
   g.setColor(new Color(nível/100,0,0));
   g.fillArc(0,0,getWidth(),getHeight(),0,360);
   }

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);
    }
  }
```







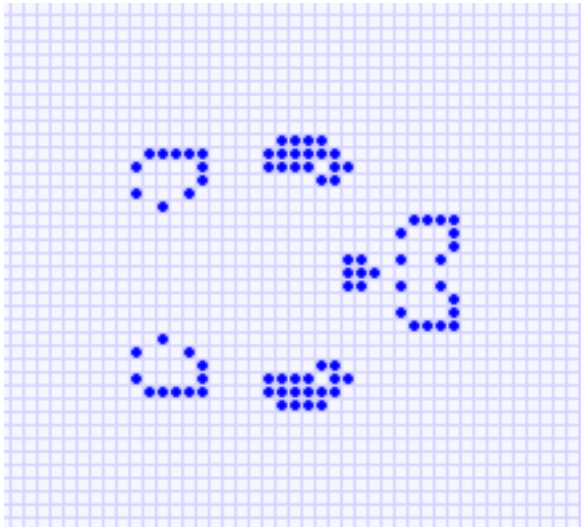
Exemplo: Jogo da Vida

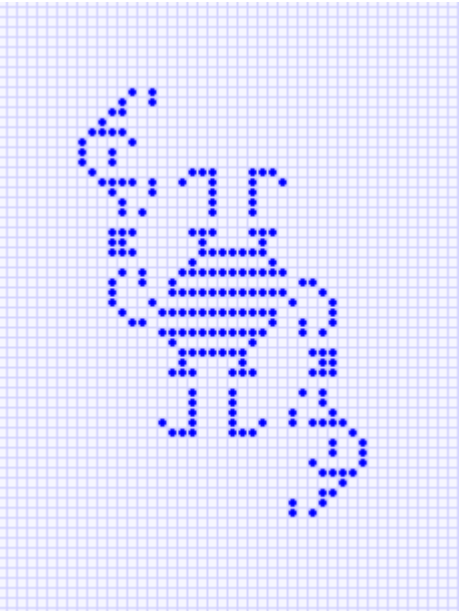
Jogo da Vida

- Simulação bem conhecida.
- Matriz de células “vivas” ou não = matriz de booleanos.
- A cada iteração...
  - Se a célula não estiver viva mas tiver três vizinhos vivos → nasce.
  - Se a célula estiver viva e tiver dois ou três vizinhos vivos → continua viva.
  - Em outros casos → morre.
- Separação de lógica de apresentação e manipulação: padrão **MVC** (*Model*, *View*, *Controller*).

Jogo da Vida: classe Life

- Existem muitas situações iniciais já prontas que podem ser usadas para iniciar o jogo da vida.
  - Ex. em http://www.radicaleye.com/lifepage/
- Instâncias da classe Life podem ser criadas usando estas situações.
  - Um construtor poderá ler as situações de arquivos.





Jogo da Vida: classe **Life**

```
P1
# Max
49 63
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
```

Jogo da Vida: classe Life

```
package life;
import java.io.*;
import java.util.*;

public class Life
  {
  private int largura, altura;
  private int iteração, númeroDeCélulas;
  private String comentário;
  private boolean[][] células,célulasOrig;
  private boolean éToroidal;
  private static final boolean VIVA = true;
  private static final boolean VAZIA = false;
  // Construtor, inicializa campos e células aleatoriamente.
  public Life(int larg,int alt,double probabilidade)
    {
    largura = larg;         altura = alt;
    iteração = 0;           númeroDeCélulas = 0;
    comentário = "Dimensões: "+largura+" x "+altura+", probabilidade:"+probabilidade;
    células = new boolean[largura][altura];
    for(int a=0;a<altura;a++)
      for(int l=0;l<largura;l++)
        if (Math.random() < probabilidade) { células[l][a] = VIVA; númeroDeCélulas++; }
        else células[l][a] = VAZIA;
    éToroidal = false;
    backup();
    }
```

Jogo da Vida: classe Life

```
// Construtor, inicializa campos a partir de um arquivo PBM
public Life(File inicial)
  {
  iteração = 0;
  númeroDeCélulas = 0;
  String linha;
  try
    {
    BufferedReader br = new BufferedReader(new FileReader(inicial));
    // Lemos a primeira linha do arquivo, que deverá ser obrigatoriamente P1.
    linha = br.readLine();
    if (!linha.startsWith("P1")) throw new IOException();
    // Lemos a segunda linha do arquivo, que será o comentário.
    linha = br.readLine();
    if (!linha.startsWith("#")) throw new IOException();
    comentário = linha.substring(1);
    // Lemos a terceira linha do arquivo, que deverá conter as dimensões.
    linha = br.readLine();
    StringTokenizer st = new StringTokenizer(linha);
    largura = Integer.parseInt(st.nextToken());
    altura = Integer.parseInt(st.nextToken());
    // Criamos a matriz de células.
    células = new boolean[largura][altura];
```

Jogo da Vida: classe Life

```
    // Vamos ler o arquivo e preencher a matriz.
    boolean lendo = true;
    int l=0,a=0;
    while(lendo)
      {
      linha = br.readLine(); if (linha == null) { lendo = false; break; }
      st = new StringTokenizer(linha);
      while(st.hasMoreTokens())
        {
        String token = st.nextToken();
        células[l][a] = token.equals("1") ? true : false;
        l++; if (l >= largura) { l = 0; a++; }
        }
      }
    br.close();
    }
  // Em caso de exceções, criamos uma simulação vazia de tamanho constante.
  catch(Exception ioe)
    {
    altura = 64;       largura = 64;
    comentário = "Simulação criada com constantes: erro no arquivo "+inicial;
    células = new boolean[largura][altura];
    for(int a=0;a<altura;a++) for(int l=0;l<largura;l++) células[l][a] = VAZIA;
    }
  éToroidal = false;
  backup();
  }
```

Jogo da Vida: classe Life

```
// Itera, calcula uma iteração do jogo da vida.
public void itera()
  {
  // Processamos tudo em uma cópia da matriz original.
  boolean[][] novasCélulas = new boolean[largura][altura];    númeroDeCélulas = 0;
  // Para cada célula verificamos a sua condição.
  for(int a=0;a<altura;a++)
    for(int l=0;l<largura;l++)
      {
      byte vizinhos = contaVizinhosVivos(a,l);
      // Aplicamos as regras do jogo.
      if ((células[l][a] == VAZIA) && (vizinhos == 3))
        {
        novasCélulas[l][a] = VIVA;     númeroDeCélulas++;
        }
      else if ((células[l][a] == VIVA) && ((vizinhos == 2) || (vizinhos == 3)))
        {
        novasCélulas[l][a] = VIVA;     númeroDeCélulas++;
        }
      else novasCélulas[l][a] = VAZIA;
      }
  // Copiamos a nova matriz para a original.
  iteração++;
  for(int a=0;a<altura;a++)
    for(int l=0;l<largura;l++)
       células[l][a] = novasCélulas[l][a];
  }
```

Jogo da Vida: classe Life

```
public byte contaVizinhosVivos(int a,int l)
  {
  byte contador=0;
  int realA,realL;
  if (éToroidal)
    {
    for(int pa=-1;pa<=1;pa++) for(int pl=-1;pl<=1;pl++)
        {
        if ((pa == 0) && (pl == 0)) continue;
        realA = a +pa; realL = l + pl;
        if (realA < 0) realA = altura-1;       if (realL < 0) realL = largura-1;
        if (realA > altura-1) realA = 0;       if (realL > largura-1) realL = 0;
        if (células[realL][realA] == VIVA) contador++;
        }
    }
  else
    {
    for(int pa=-1;pa<=1;pa++) for(int pl=-1;pl<=1;pl++)
        {
        if ((pa == 0) && (pl == 0)) continue;
        realA = a + pa; realL = l + pl;
        if ((realA < 0) || (realL < 0) || (realA > altura-1) || (realL > largura-1))
          continue;
        if (células[realL][realA] == VIVA) contador++;
        }
    }
  return contador;
  }
```

Jogo da Vida: classe Life

```
public void backup()
  {
  if (célulasOrig == null) célulasOrig = new boolean[largura][altura];
  for(int a=0;a<altura;a++)
    for(int l=0;l<largura;l++)
      célulasOrig[l][a] = células[l][a];
  }

public void restore()
  {
  for(int a=0;a<altura;a++)
    for(int l=0;l<largura;l++)
      células[l][a] = célulasOrig[l][a];
  iteração = 0;
  }

public boolean[][] getCélulas()  { return células; }

public int getAltura()  { return altura; }
public int getLargura() { return largura; }

public int getNúmeroDeCélulas() { return númeroDeCélulas; }

public int getIteração() { return iteração; }

public String getComentário() { return comentário; }
}
```

Jogo da Vida: classe LifeComponent

```
package life;

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

public class LifeComponent extends JComponent
  {
  private Life life;
  private int largura,altura;
  private int larguraComponente,alturaComponente;
  private static final int borda = 10;
  private static final int espessuraLinha = 1;
  private static final int tamanhoDaCélula = 4;
  private static final Color corFundo = new Color(245,245,255);
  private static final Color corLinha = new Color(210,210,255);
  private static final Color corCélula = new Color(0,0,255);

  public LifeComponent(Life life)
    {
    this.life = life;
    // Obtemos alguns valores para evitar chamadas repetidas aos métodos.
    largura = life.getLargura();     altura  = life.getAltura();
    // Calculamos previamente a largura e altura desejada para este componente.
    larguraComponente = borda*2+largura*tamanhoDaCélula+(largura+1)*espessuraLinha;
    alturaComponente =  borda*2+altura*tamanhoDaCélula+(altura+1)*espessuraLinha;
    }
```

Jogo da Vida: classe LifeComponent

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

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

public Dimension getPreferredSize()
  {
  return new Dimension(larguraComponente,alturaComponente);
  }
public void itera()
  {
  life.itera();
  }

public void reinicializa()
  {
  life.restore();
  }

public int getIteração()
  {
  return life.getIteração();
  }

public int getNúmeroDeCélulas()
  {
  return life.getNúmeroDeCélulas();
  }
```

Jogo da Vida: classe LifeComponent

```
protected void paintComponent(Graphics g)
  {
  Graphics2D g2d = (Graphics2D)g;
  g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                       RenderingHints.VALUE_ANTIALIAS_ON);
  // Pintamos o fundo.
  g2d.setColor(corFundo);
  g2d.fillRect(0,0,getPreferredSize().width,getPreferredSize().height);
  // Desenhamos a borda das células (horizontais e verticais separadas).
  g2d.setColor(corLinha);
  for(int a=0;a<altura+1;a++)
    g2d.drawLine(borda,borda+a*(tamanhoDaCélula+espessuraLinha),
                 larguraComponente-borda-1,borda+a*(tamanhoDaCélula+espessuraLinha));
  for(int l=0;l<largura+1;l++)
    g2d.drawLine(borda+l*(tamanhoDaCélula+espessuraLinha),borda,
                 borda+l*(tamanhoDaCélula+espessuraLinha),alturaComponente-borda-1);
  // Desenhamos as células.
  g2d.setColor(corCélula);
  boolean[][] células = life.getCélulas();
  for(int a=0;a<altura;a++)
    for(int l=0;l<largura;l++)
      if (células[l][a]) g2d.fillOval(1+borda+l*(tamanhoDaCélula+espessuraLinha),
                                      1+borda+a*(tamanhoDaCélula+espessuraLinha),
                                      tamanhoDaCélula,tamanhoDaCélula);
  }
}
```

Jogo da Vida: classe LifeApp

```
package life;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.util.Hashtable;
import javax.swing.*;
import javax.swing.event.*;

public class LifeApp extends JFrame implements ActionListener,ChangeListener
  {
  private LifeComponent lifeComponent;
  private JButton inicializa,reinicializa;
  private JLabel númeroDeCélulas,iterações;
  private JSlider velocidade;
  private final int[] velocidades = {1,2,5,10,20,50,100,200,500};
  private Timer timer;
  public LifeApp(Life life)
    {
    super(life.getComentário());
    JPanel conteúdo = (JPanel)getContentPane();  conteúdo.setLayout(new BorderLayout());
    // Primeiro criamos o componente que vai conter o jogo da vida.
    JPanel cl = criaComponenteJogo(life);    conteúdo.add(cl,BorderLayout.CENTER);
    // Criamos o painel de controle com informações e botões.
    JPanel pc = criaPainelDeControle(life);  conteúdo.add(pc,BorderLayout.EAST);
    timer = new Timer(velocidades[3],this);  timer.setCoalesce(false);
    pack();  setVisible(true); setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

Jogo da Vida: classe LifeApp

```
protected JPanel criaComponenteJogo(Life life)
  {
  // Criamos uma instância do componente que vai conter o jogo da vida.
  lifeComponent = new LifeComponent(life);
  // De acordo com a sugestão da documentação (JComponent.setBorder), colocamos
  // o componente em um JPanel.
  JPanel lbc = new JPanel(new BorderLayout());
  lbc.add(lifeComponent,BorderLayout.CENTER);
  lbc.setBorder(BorderFactory.createLineBorder(Color.BLUE,3));
  return lbc;
  }

protected JPanel criaPainelDeControle(Life life)
  {
  JPanel painelDeControle = new JPanel(new GridBagLayout());
  GridBagConstraints gbc = new GridBagConstraints();
  gbc.gridx = 0;
  gbc.weightx = 1; gbc.weighty = 0;
  gbc.fill = GridBagConstraints.HORIZONTAL;
  gbc.anchor = GridBagConstraints.NORTH;
```

Jogo da Vida: classe LifeApp

```
    // Criamos o contador de iterações e o adicionamos ao painel de controle.
    criaContadorIterações();
    gbc.gridy = 0;
    painelDeControle.add(iterações, gbc);
    // Criamos o contador de células e o adicionamos ao painel de controle.
    criaContadorCélulas(life);
    gbc.gridy = 1;
    gbc.weighty = 0;
    painelDeControle.add(númeroDeCélulas,gbc);
    // Criamos os botões e outros controles. Vamos agrupá-los em um único JPanel.
    JPanel botões = new JPanel(new GridLayout(3,1));
    inicializa = new JButton("Inicializa");
    inicializa.addActionListener(this);
    botões.add(inicializa);
    reinicializa = new JButton("Reinicializa");
    reinicializa.addActionListener(this);
    botões.add(reinicializa);
    criaControleVelocidade();
    botões.add(velocidade);
    botões.setBorder(BorderFactory.createTitledBorder(
                     BorderFactory.createLineBorder(Color.BLUE),"Controles"));
    gbc.gridy = 2; gbc.weighty = 1;
    painelDeControle.add(botões,gbc);
    return painelDeControle;
    }
```

Jogo da Vida: classe LifeApp

```
protected void criaContadorIterações()
  {
  iterações = new JLabel("0");
  iterações.setHorizontalAlignment(SwingConstants.CENTER);
  iterações.setForeground(new Color(50, 50, 255));
  iterações.setFont(new Font("Arial", Font.BOLD, 32));
  iterações.setBorder(BorderFactory.createTitledBorder(
                      BorderFactory.createLineBorder(Color.BLUE), "Iterações"));

  }

protected void criaContadorCélulas(Life life)
  {
  númeroDeCélulas = new JLabel("" + life.getNúmeroDeCélulas());
  númeroDeCélulas.setHorizontalAlignment(SwingConstants.CENTER);
  númeroDeCélulas.setForeground(Color.BLUE);
  númeroDeCélulas.setFont(new Font("Arial", Font.BOLD, 32));
  númeroDeCélulas.setBorder(BorderFactory.createTitledBorder(
                            BorderFactory.createLineBorder(Color.BLUE), "Células"));
  }
```

Jogo da Vida: classe LifeApp

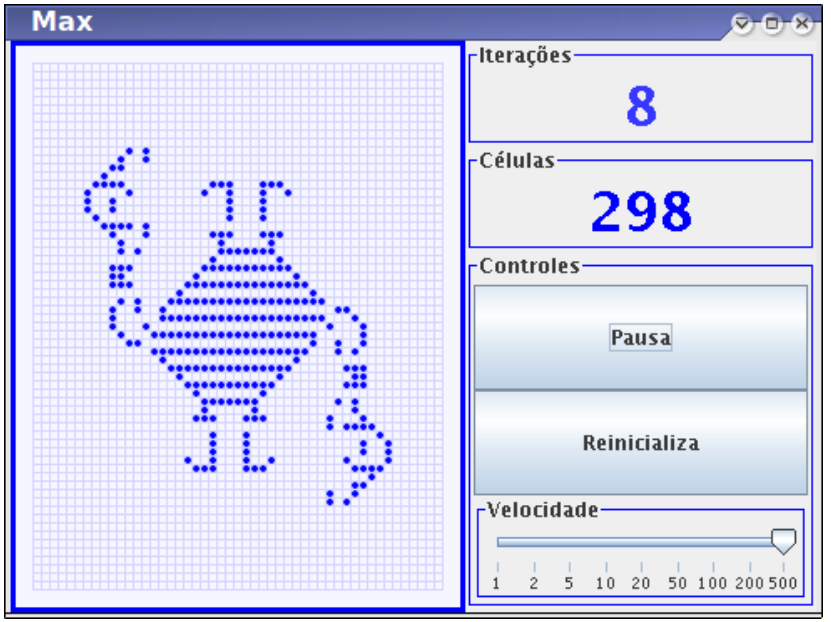
```
protected  void criaControleVelocidade()
  {
  velocidade = new JSlider(JSlider.HORIZONTAL,0,velocidades.length-1,3);
  velocidade.setMajorTickSpacing(1);
  velocidade.setMinorTickSpacing(1);
  velocidade.setPaintTicks(true);
  velocidade.setPaintLabels(true);
  velocidade.setSnapToTicks(true);
  // Colocamos os labels no controle de velocidade.
  Hashtable labelTable = new Hashtable();
  for(int v=0;v<velocidades.length;v++)
    {
    JLabel l = new JLabel(""+velocidades[v]);
    l.setFont(new Font("Arial",Font.PLAIN,9));
    labelTable.put(new Integer(v),l); // Sem generics :-(
    }
  velocidade.setLabelTable(labelTable);
  velocidade.setBorder(BorderFactory.createTitledBorder(
                       BorderFactory.createLineBorder(Color.BLUE),"Velocidade"));
  velocidade.addChangeListener(this);
  }
```

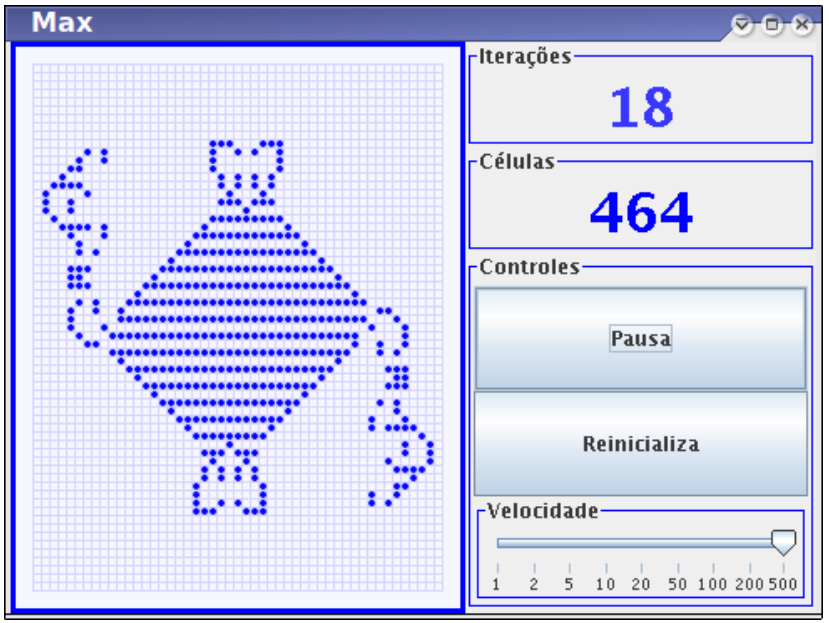
Jogo da Vida: classe LifeApp

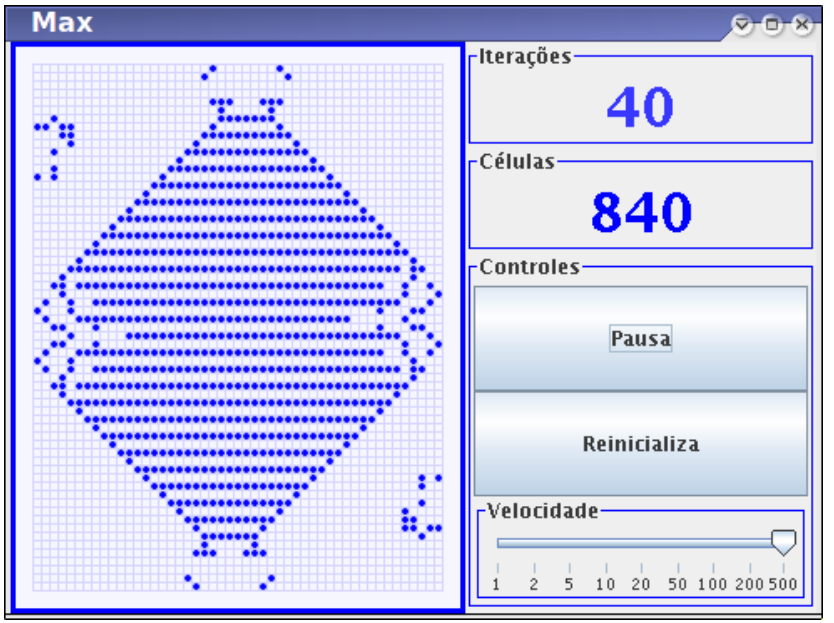
```
public void actionPerformed(ActionEvent e)
  {
  if (e.getSource() == timer) { lifeComponent.itera(); lifeComponent.repaint(); }
  else if (e.getSource() == inicializa)
    {
    if (!timer.isRunning()) { inicializa.setText("Pausa"); timer.start(); }
    else { inicializa.setText("Continua"); timer.stop(); }
    }
  else if (e.getSource() == reinicializa)
    {
    lifeComponent.reinicializa(); lifeComponent.repaint();
    }
  iterações.setText(""+lifeComponent.getIteração());
  númeroDeCélulas.setText(""+lifeComponent.getNúmeroDeCélulas());
  }

public void stateChanged(ChangeEvent e)
  {
  timer.setDelay(velocidades[velocidade.getValue()]);
  }

public static void main(String[] args)
  {
  Life life = new Life(new File("../Dados/max.pbm"));
  new LifeApp(life);
  }
}
```

Jogo da Vida


Max
Iterações
0
Células
0
Controles
Inicializa
Reinicializa
Velocidade
1 2 5 10 20 50 100 200 500


Max
Iterações
8
Células
298
Controles
Pausa
Reinicializa
Velocidade
1 2 5 10 20 50 100 200 500


Max
Iterações
18
Células
464
Controles
Pausa
Reinicializa
Velocidade
1 2 5 10 20 50 100 200 500


Max
Iterações
40
Células
840
Controles
Pausa
Reinicializa
Velocidade
1 2 5 10 20 50 100 200 500

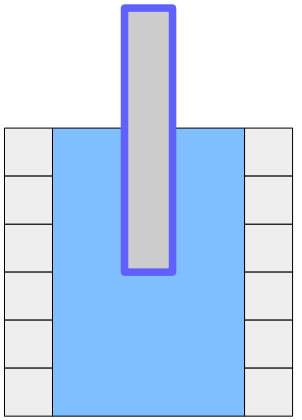
Exemplo: Tanque

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**

```
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;
     }
```

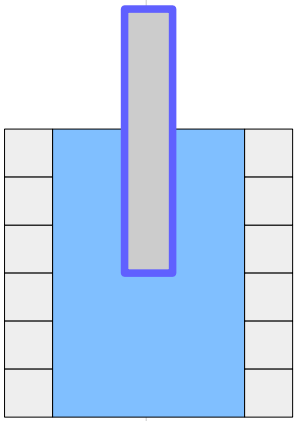


Tanques: classe Tanque

```
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;
   }
```

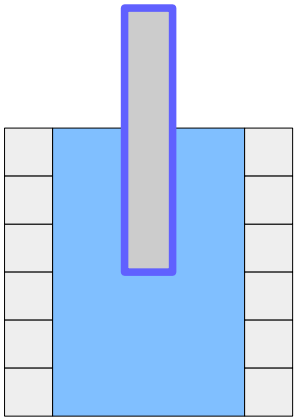
Tanques: classe Tanque

```
public void draw(Graphics2D g2d)
  {
  // Armazenamos o sistema de coordenadas original.
  AffineTransform antes = g2d.getTransform();
  // Criamos um sistema de coordenadas para o tanque.
  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

```
    // 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);
```

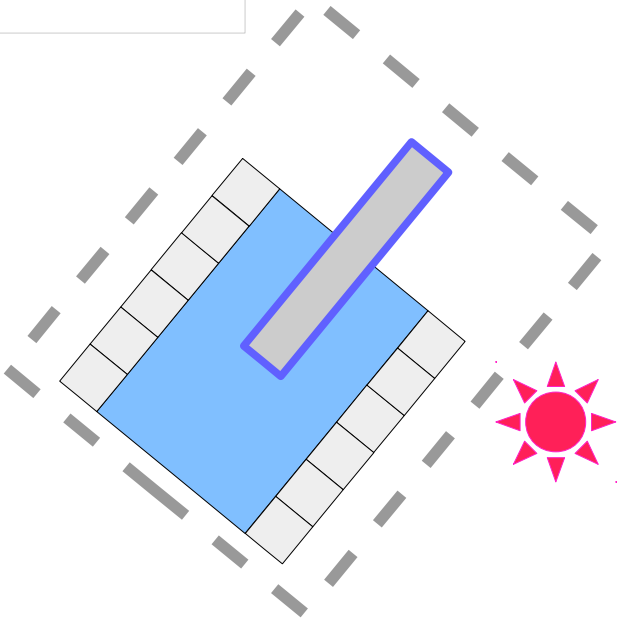


Tanques: classe Tanque

```
    // 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

```
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**

```
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

```
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**

```
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**

```
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();
  }
```

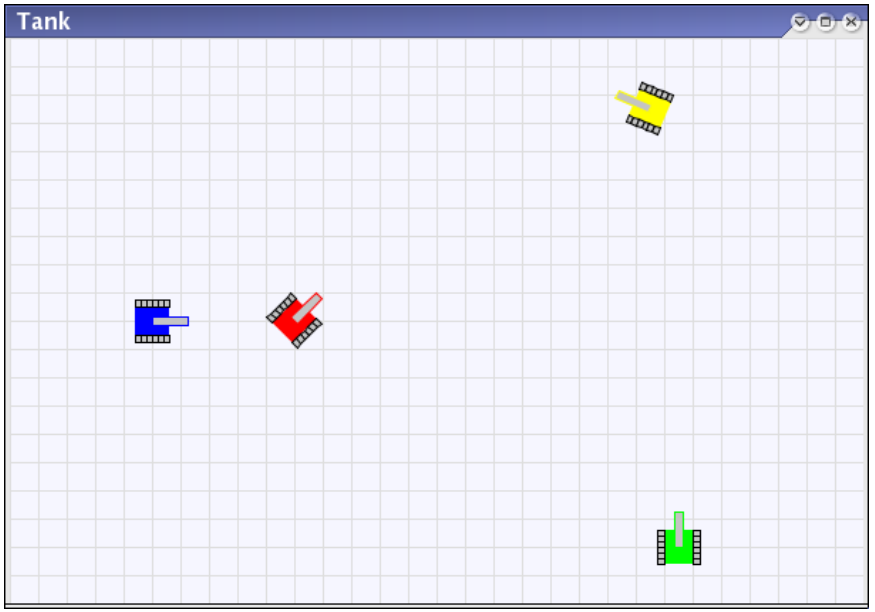
Tanques: classe **Arena**

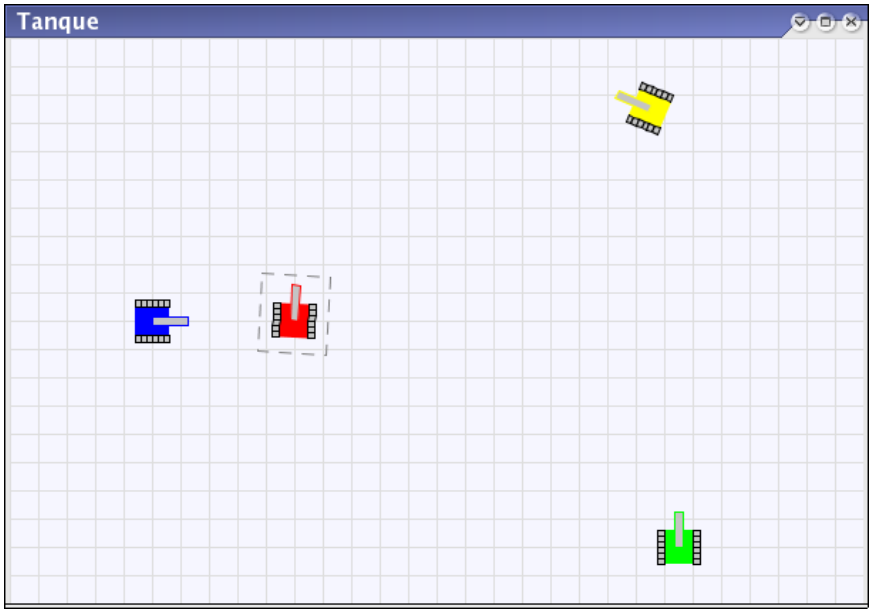
```
public void mouseEntered(MouseEvent e) { }

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();
   }

}
```

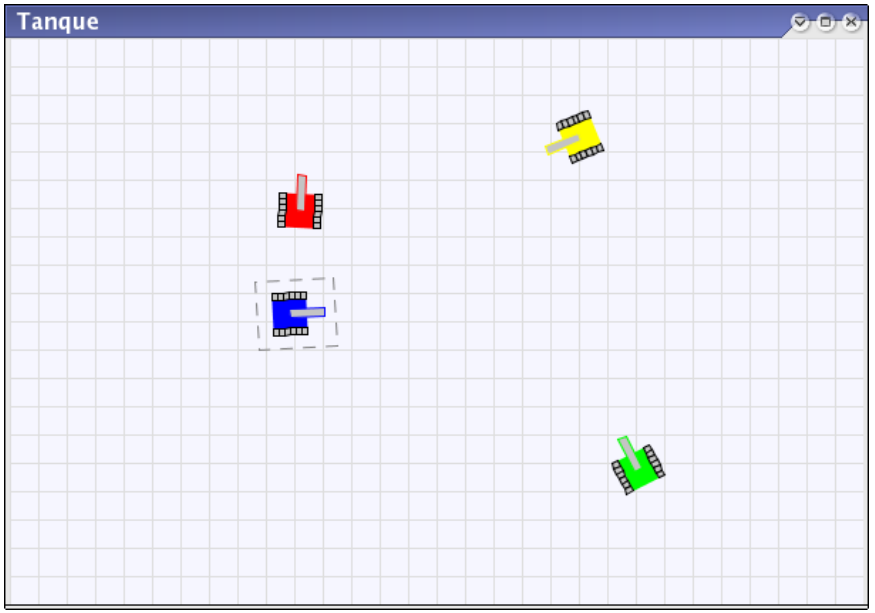
Tanques: classe App

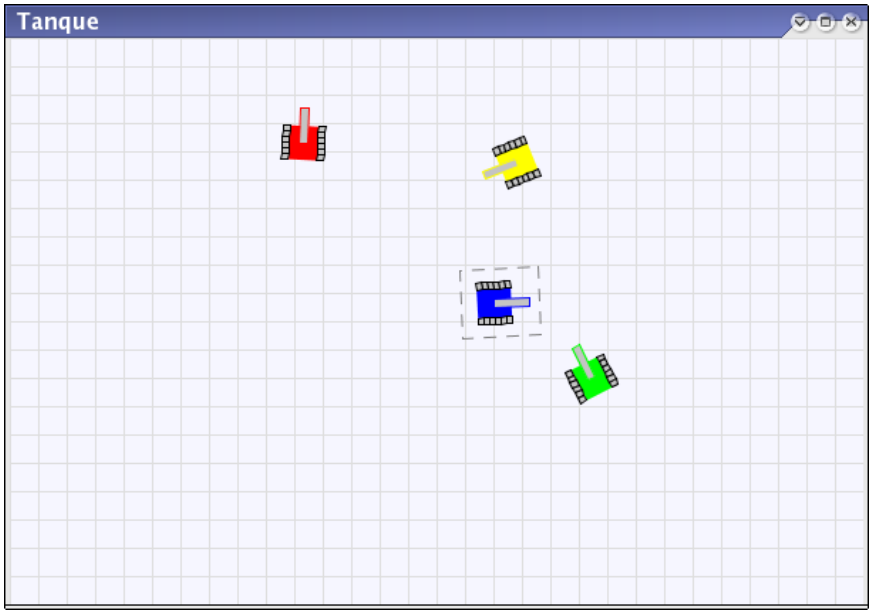
```
package tanques;

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







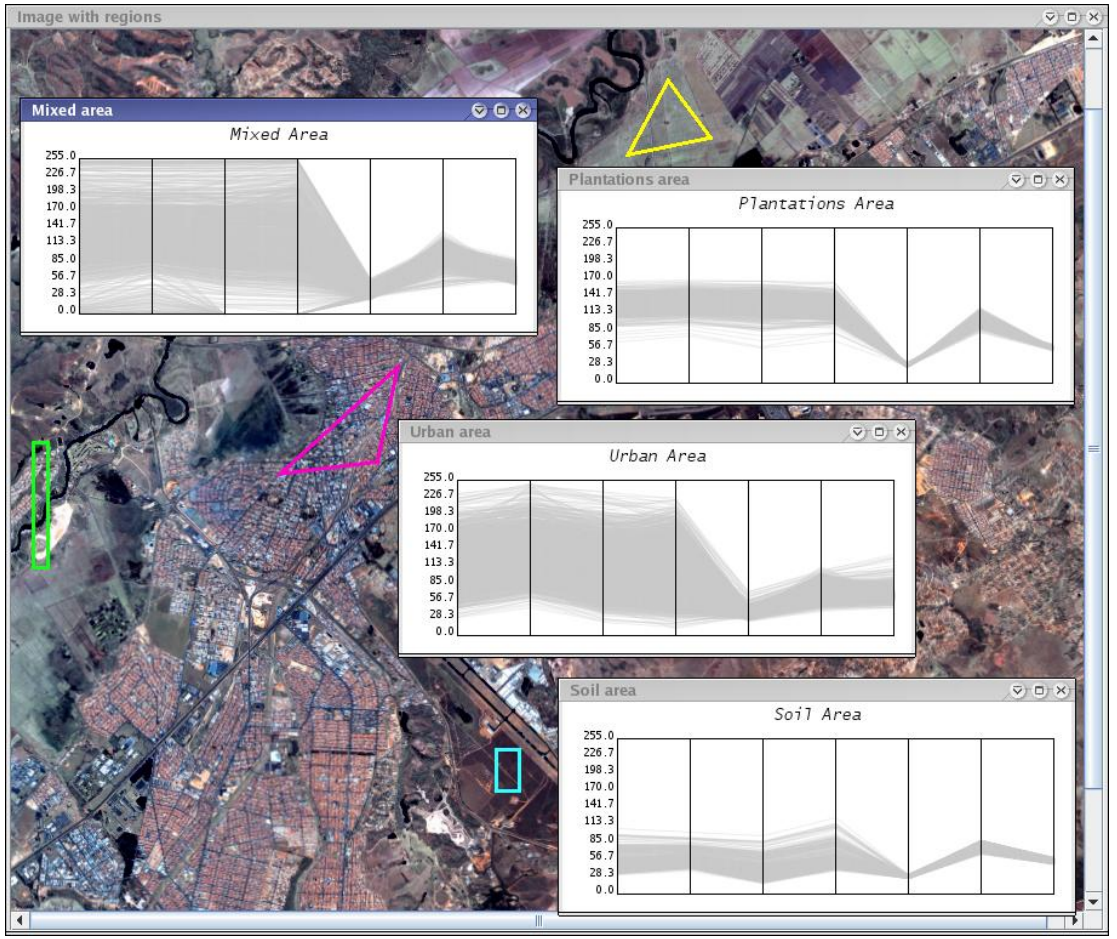


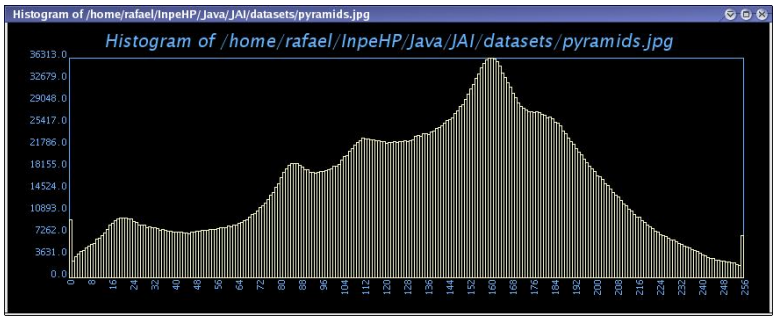
Tanques

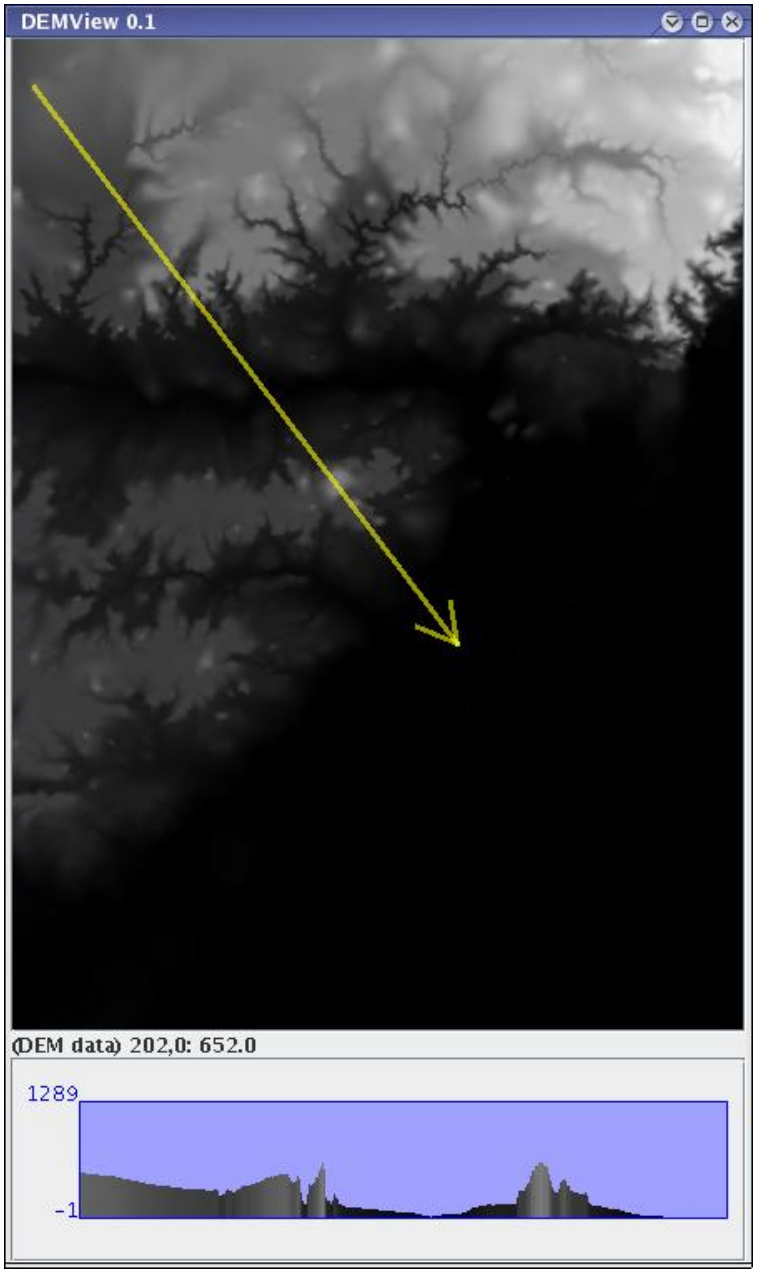
- Como está, não é interessante, mas...
- É simples criar um ambiente onde podemos interagir com vários objetos.
- Como fazer objetos interagir com outros?
  - Colisão: detectável com métodos que verificam intersecção entre retângulos envolventes.
  - Limites da arena: verificável de forma simples.
-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

Outros exemplos

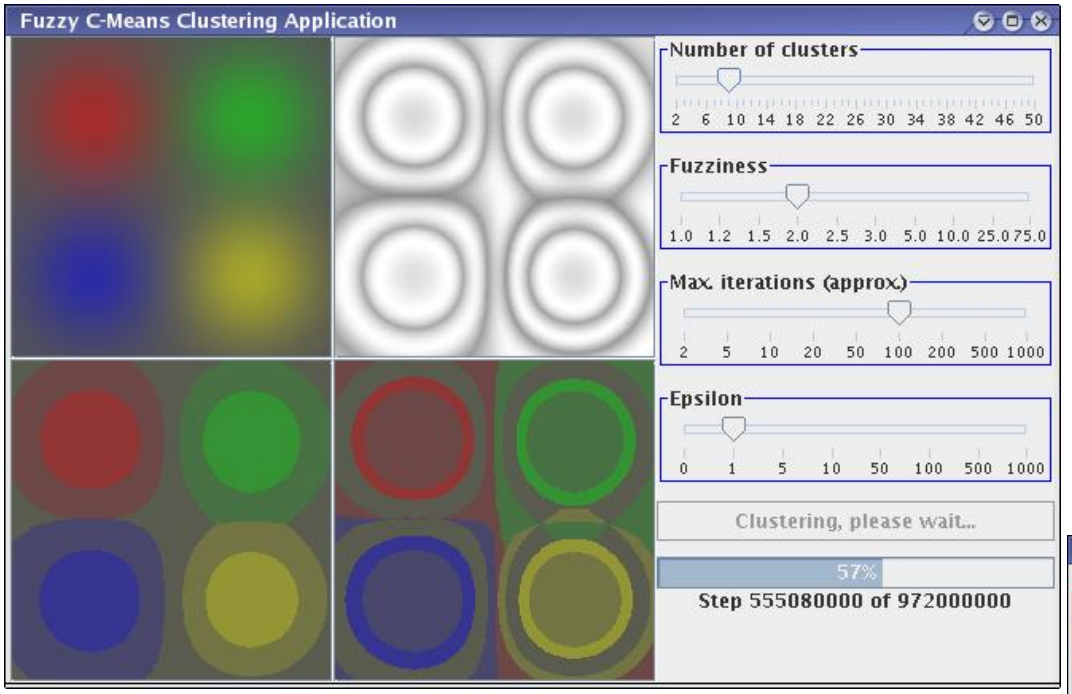
Exemplos de componentes (Proc. Imagens)

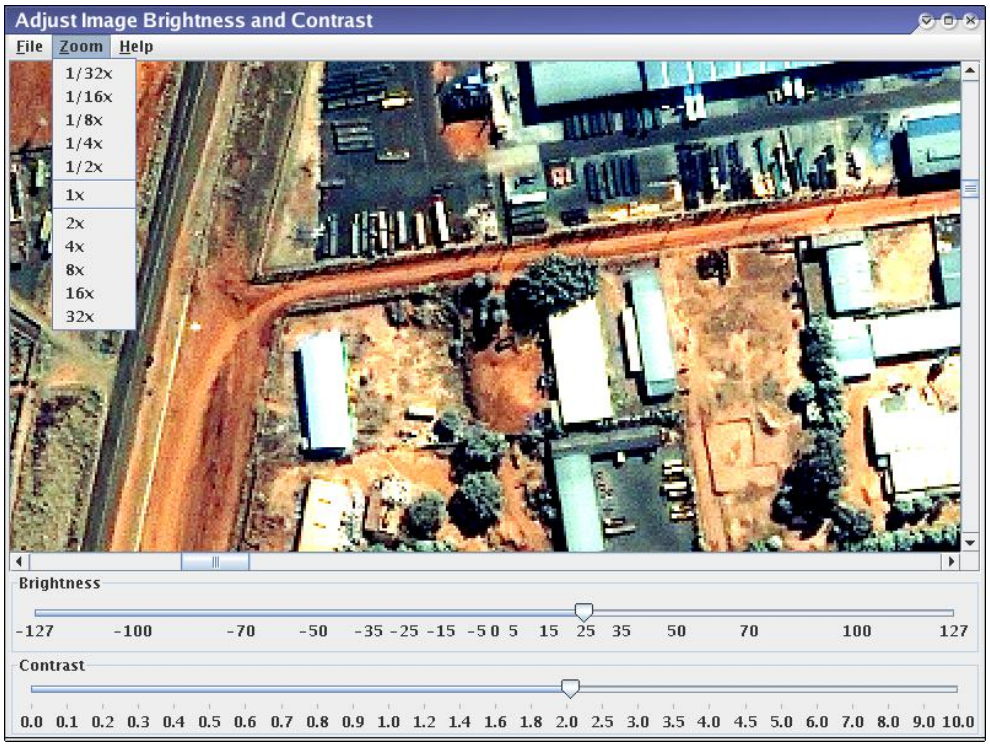






Exemplos de aplicações (Proc. Imagens)



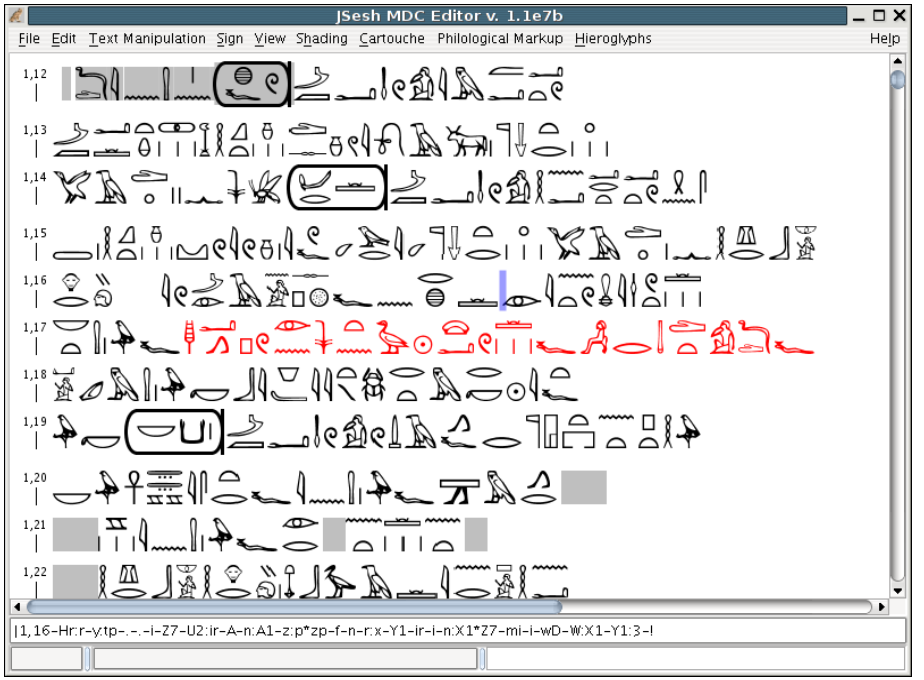


Concluindo...

- Vimos...
  - Algumas noções de programação com interfaces gráficas em Java.
  - Como usar alguns componentes comuns.
  - Como usar eventos em suas aplicações.
  - Como desenvolver componentes simples.
- Não vimos...
  - **Muita** coisa!
  - Formalização do modelo **MVC**.
  - Componentes mais complexos, layouts mais complexos, etc.

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
- *Swing Sightings* (http://java.sun.com/products/jfc/tsc/sightings/)



Perguntas?