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*.

Processamento de Imagens: **JAI**

JAI

- http://jai.dev.java.net/
- JAI:
  - É um conjunto de classes para processamento de imagens *genéricas*.
  - Contém um modelo de programação que facilita o desenvolvimento de aplicações para processamento de *grandes* imagens.
  - Não tem interfaces gráficas (provê classes para visualização fora do pacote javax.jai)
  - Vai além do que AWT/Swing oferece!
  - Nos exemplos usaremos JAI + Swing/AWT + ImageIO.
- Originalmente subprojeto (JSR) da SUN, hoje aberto à comunidade.

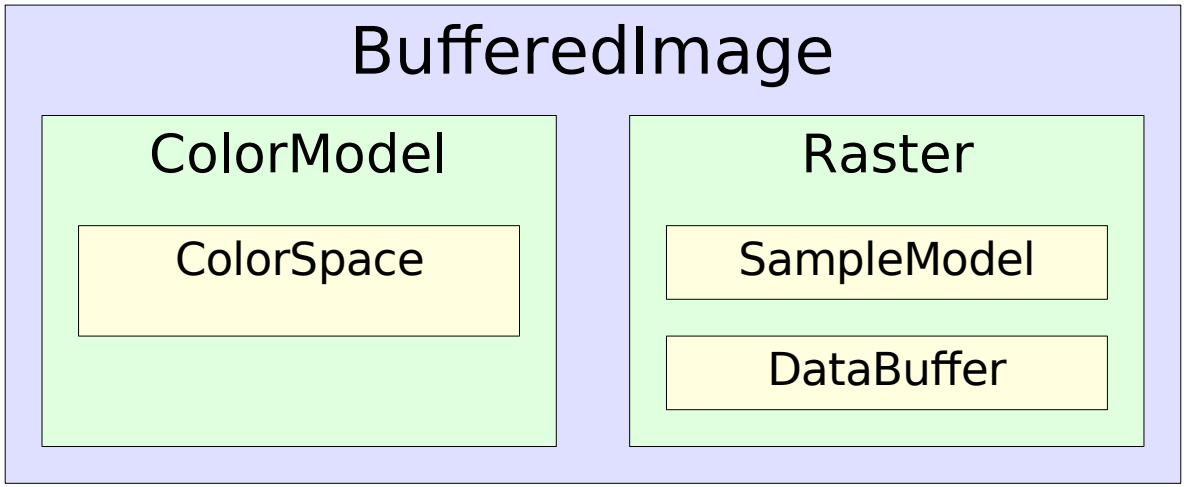
JAI

- Conceitos básicos de processamento de imagens:
  - Uma imagem é uma matriz de *pixels*.
  - Um *pixel* é uma medida (ou conjunto de medidas) sobre uma mesma área na imagem **ou** um índice em uma tabela de valores.
  - Uma imagem pode conter *metadados*.
  - Não somos limitados a valores no intervalo [0,255] ou mesmo a valores inteiros ou positivos!
  - Não somos limitados a um **ou** três valores por pixel: podemos ter imagens multi e hiperespectrais.

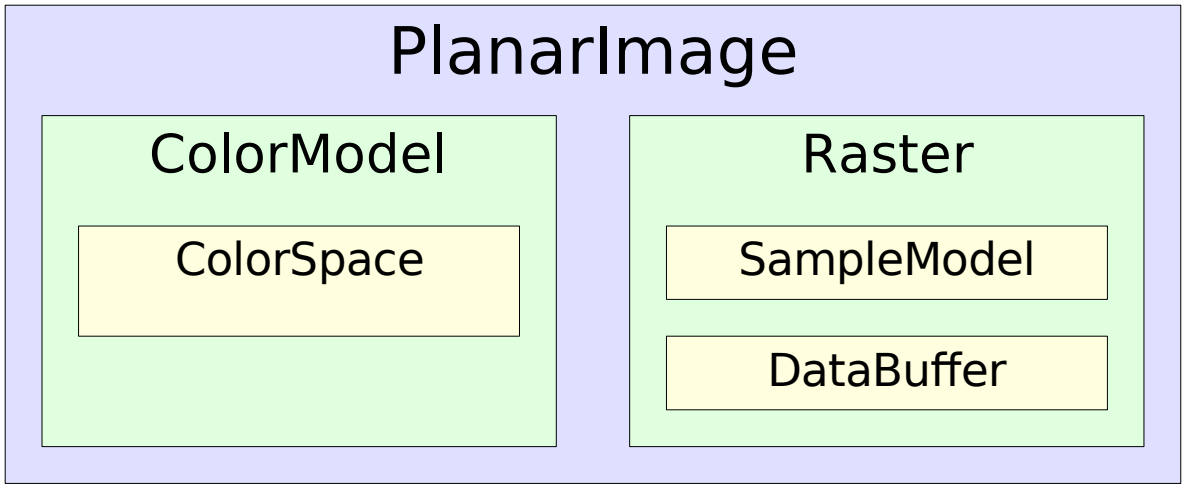
JAI

- Classes para representação de imagens:

AWT/Swing:



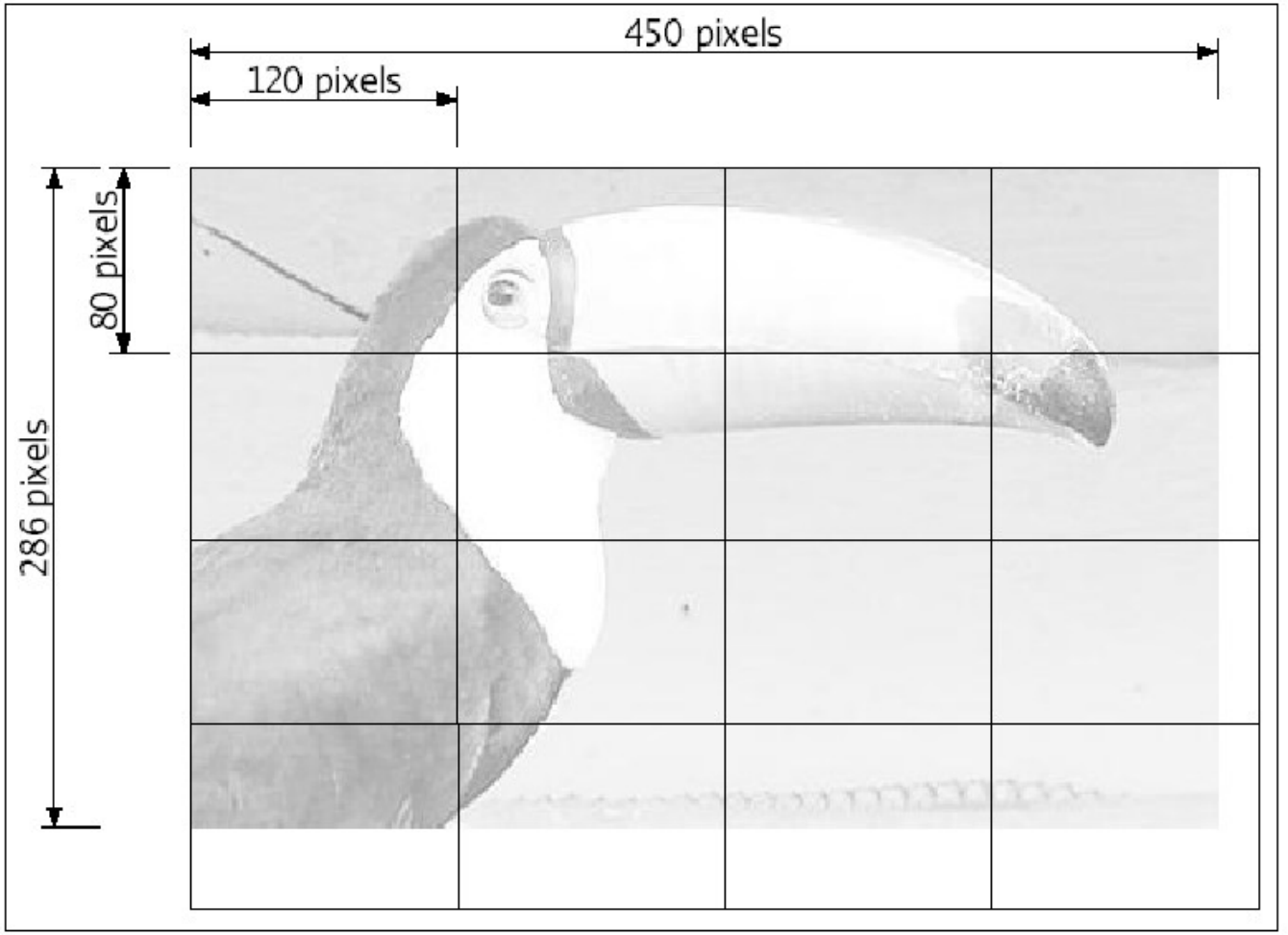
JAI:



Parece o mesmo, mas *PlanarImage*s podem ter *layouts* diferentes, e classes herdeiras tem características especiais.

JAI

- Imagens *Tiled*: permitem o processamento, quando possível, por ladrilhos, reduzindo necessidades de memória.



JAI: Estruturas de imagens

```
package jai;

import java.awt.Transparency;
import java.awt.image.*;
import java.net.MalformedURLException;
import java.net.URL;
import javax.media.jai.*;

public class ImageInfo
  {
  public static void main(String[] args) throws MalformedURLException
    {
    PlanarImage pi = JAI.create("url",new URL(args[0]));
    System.out.print("Dimensões: ");
    System.out.print(pi.getWidth()+"x"+pi.getHeight()+" pixels");
    System.out.println(" (de "+pi.getMinX()+","+pi.getMinY()+" para " +
                               (pi.getMaxX()-1)+","+(pi.getMaxY()-1)+")");
    if (pi.getNumXTiles()*pi.getNumYTiles() > 1)
      {
      System.out.print("Tiles: ");
      System.out.print(pi.getTileWidth()+"x"+pi.getTileHeight()+" pixels"+
                       " ("+pi.getNumXTiles()+"x"+pi.getNumYTiles()+" tiles)");
      System.out.print(" (de "+pi.getMinTileX()+","+pi.getMinTileY()+" para "+
                               pi.getMaxTileX()+","+pi.getMaxTileY()+")");
      System.out.println(" offset: "+pi.getTileGridXOffset()+","+
                                     pi.getTileGridXOffset());
      }
```

JAI: Estruturas de imagens

```
// Informaçõµes sobre o modelo da imagem
SampleModel sm = pi.getSampleModel();
System.out.println("Bandas: "+sm.getNumBands());
System.out.print("Tipo: ");
switch(sm.getDataType())
  {
  case DataBuffer.TYPE_BYTE:      System.out.println("byte");      break;
  case DataBuffer.TYPE_SHORT:     System.out.println("short");     break;
  case DataBuffer.TYPE_USHORT:    System.out.println("ushort");    break;
  case DataBuffer.TYPE_INT:       System.out.println("int");       break;
  case DataBuffer.TYPE_FLOAT:     System.out.println("float");     break;
  case DataBuffer.TYPE_DOUBLE:    System.out.println("double");    break;
  case DataBuffer.TYPE_UNDEFINED: System.out.println("undefined"); break;
  }
```

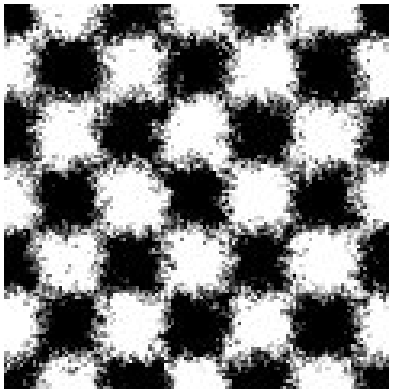
JAI: Estruturas de imagens

```
  // Modelo de cores
  ColorModel cm = pi.getColorModel();
  if (cm != null)
    {
    System.out.print("Colormap tipo: ");
    if (cm instanceof DirectColorModel) System.out.println("DirectColorModel");
    else if (cm instanceof PackedColorModel) System.out.println("PackedColorModel");
    else if (cm instanceof IndexColorModel)
      {
      IndexColorModel icm = (IndexColorModel)cm;
      System.out.println("IndexColorModel com "+icm.getMapSize()+" entradas");
      }
    else if (cm instanceof ComponentColorModel)
            System.out.println("ComponentColorModel");
    System.out.println("Número de componentes: "+cm.getNumComponents());
    System.out.println("Bits por pixel: "+cm.getPixelSize());
    System.out.print("Transparência: ");
    switch(cm.getTransparency())
      {
      case Transparency.OPAQUE:      System.out.println("opaca");        break;
      case Transparency.BITMASK:     System.out.println("bitmask");      break;
      case Transparency.TRANSLUCENT: System.out.println("translucente"); break;
      }
    }
  else System.out.println("Sem modelo de cores.");
  }
}
```

JAI: Estruturas de imagens

http://www.lac.inpe.br/~rafael.santos/Java/JAI/datasets/landsat.tif
Dimensões: 900x900 pixels (de 0,0 para 899,899)
Tiles: 900x8 pixels (1x113 tiles) (de 0,0 para 0,112) offset: 0,0
Bandas: 7
Tipo: byte
Sem modelo de cores.

http://www.lac.inpe.br/~rafael.santos/Java/JAI/datasets/heart.tif
Dimensões: 256x256 pixels (de 0,0 para 255,255)
Tiles: 256x8 pixels (1x32 tiles) (de 0,0 para 0,31) offset: 0,0
Bandas: 16
Tipo: ushort
Sem modelo de cores.



http://www.lac.inpe.br/~rafael.santos/Java/JAI/datasets/noisypattern.png
Dimensões: 300x300 pixels (de 0,0 para 299,299)
Bandas: 1
Tipo: byte
Colormap tipo: IndexColorModel com 2 entradas
Number of color components: 3
Bits per pixel: 1
Transparency: opaque

JAI: Lendo pixels de imagens

- Primeiro método: com *Iterator*

```
package jai;
import java.awt.image.SampleModel;
import java.net.MalformedURLException;
import java.net.URL;
import javax.media.jai.*;
import javax.media.jai.iterator.*;
public class PrintPixelsWithRandomIterator
  {
  public static void main(String[] args) throws MalformedURLException
    {
    PlanarImage pi = JAI.create("url",new URL(args[0]));
    int width = pi.getWidth();              int height = pi.getHeight();
    SampleModel sm = pi.getSampleModel();  int nbands = sm.getNumBands();
    int[] pixel = new int[nbands];    // Assumimos pixels inteiros!
    // Criamos um iterator retangular
    RandomIter iterator = RandomIterFactory.create(pi, null);
    // Varremos os pixels da imagem.
    for(int h=0;h<height;h++) for(int w=0;w<width;w++)
        {
        iterator.getPixel(w,h,pixel);
        System.out.print("Em ("+w+","+h+"): ");
        for(int band=0;band<nbands;band++) System.out.print(pixel[band]+" ");
        System.out.println();
        }
    }
  }
```

JAI: Lendo pixels de imagens

- Segundo método: com *Raster.getPixels()*

```
package jai;
import java.awt.image.*;
import java.net.*;
import javax.media.jai.*;
public class PrintPixelsWithBufferAccess
  {
  public static void main(String[] args) throws MalformedURLException
    {
    PlanarImage pi = JAI.create("url",new URL(args[0]));
    int width = pi.getWidth();                  int height = pi.getHeight();
    SampleModel sm = pi.getSampleModel();       int nbands = sm.getNumBands();
    // Raster da imagem e matriz de pixels
    Raster inputRaster = pi.getData();
    int[] pixels = new int[nbands*width*height];
    inputRaster.getPixels(0,0,width,height,pixels);
    // Varremos os pixels da imagem.
    for(int h=0;h<height;h++)
      for(int w=0;w<width;w++)
        {
        int offset = h*width*nbands+w*nbands;
        System.out.print("at ("+w+","+h+"): ");
        for(int band=0;band<nbands;band++)  System.out.print(pixels[offset+band]+" ");
        System.out.println();
        }
    }
  }
```

JAI: Criando imagens pixel a pixel

- Passos:
  1. Criar array w*h*d e colocar valores.
  2. Criar BufferedImage e obter seu WritableRaster.
  3. Colocar pixels no Raster.

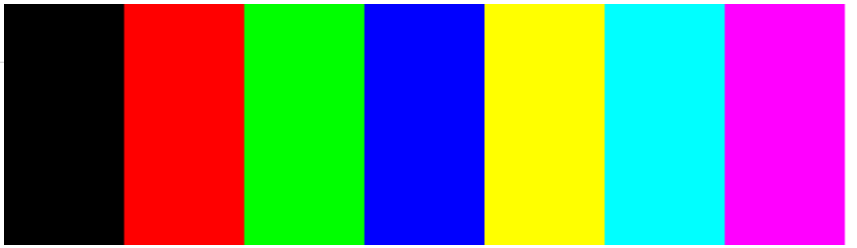
```
package jai;

import java.awt.image.*;
import java.io.*;
import javax.imageio.ImageIO;

public class CreateRGBImage
  {
  public static void main(String[] args) throws IOException
    {
    int width = 60*8;
    int height = 120;
    int[][][] imageData = new int[width][height][3];
```

JAI: Criando imagens pixel a pixel

```
for(int w=0;w<width;w++)
  for(int h=0;h<height;h++)
    {
    int whichBand = (int)(w/60);
    switch(whichBand) // KRGBYCMW
      {
      case 0:
        {
        imageData[w][h][0] =   0; imageData[w][h][1] =   0; imageData[w][h][2] =   0; break;
        }
      case 1:
        {
        imageData[w][h][0] = 255; imageData[w][h][1] =   0; imageData[w][h][2] =   0; break;
        }
      case 2:
        {
        imageData[w][h][0] =   0; imageData[w][h][1] = 255; imageData[w][h][2] =   0; break;
        }
      case 3:
        {
        imageData[w][h][0] =   0; imageData[w][h][1] =   0; imageData[w][h][2] = 255; break;
        }
      case 4:
        {
        imageData[w][h][0] = 255; imageData[w][h][1] = 255; imageData[w][h][2] =   0; break;
        }
```

JAI: Criando imagens pixel a pixel

```
      case 5:
        {
        imageData[w][h][0] =   0; imageData[w][h][1] = 255; imageData[w][h][2] = 255; break;
        }
      case 6:
        {
        imageData[w][h][0] = 255; imageData[w][h][1] =   0; imageData[w][h][2] = 255; break;
        }
      case 7:
        {
        imageData[w][h][0] = 255; imageData[w][h][1] = 255; imageData[w][h][2] = 255; break;
        }
      }
    }
    BufferedImage image = new BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
    WritableRaster raster = image.getRaster();
    for(int h=0;h<height;h++)
      for(int w=0;w<width;w++)
        raster.setPixel(w,h,imageData[w][h]);
    ImageIO.write(image,"PNG",new File("rgbpattern.png"));
    }
}
```

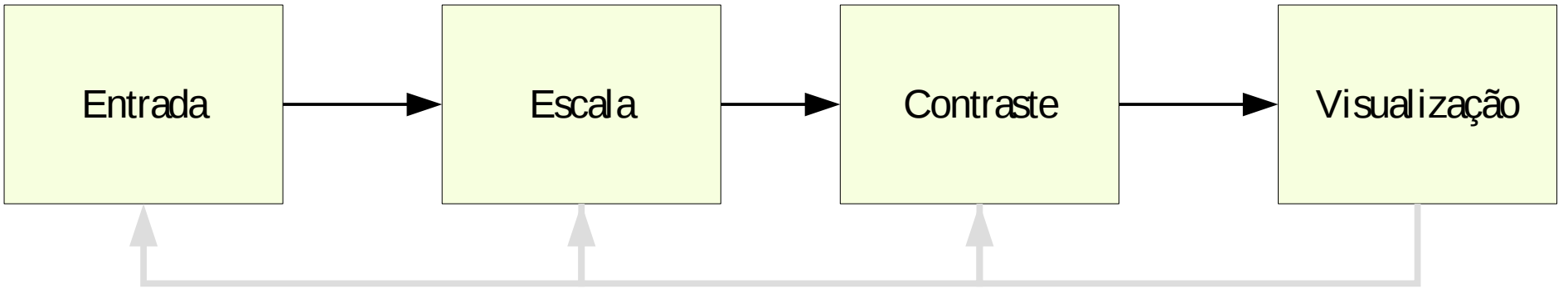


JAI: Criando imagens pixel a pixel

- O exemplo anterior não usa JAI, é mais simples mas serve só para imagens simples.
- Passos com JAI:
  1. Criar array (unidimensional) da imagem.
  2. Criar DataBuffer a partir do array.
  3. Criar SampleModel a partir do DataBuffer.
  4. Criar ColorModel a partir do DataBuffer.
  5. Criar Raster a partir do SampleModel e DataBuffer.
  6. Criar TiledImage a partir do SampleModel e ColorModel.
  7. Colocar Raster na TiledImage.

JAI: operadores e execução em demanda

- JAI provê dezenas de operadores de diversos tipos.
- Operadores podem ser encadeados como grafos, para realizar uma tarefa específica.
- Execução dos operadores não é imediata: se imagens são *tiled*, a execução é *em demanda*.
- Redução de uso de memória, possibilita tratamento de grandes imagens.



JAI: Exemplo de operadores

- JAI.create() será usado para ler, fazer convolução e inverter uma imagem.

```
package jai;
import java.net.*;
import javax.media.jai.*;
import javax.swing.JFrame;
public class EdgeInvert
  {
  public static void main(String[] args) throws MalformedURLException
    {
    PlanarImage input = JAI.create("url",new URL(args[0]));
    float[] kernelMatrix = { -1, -2, -1,
                              0,  0,  0,
                              1,  2,  1 };
    KernelJAI kernel = new KernelJAI(3,3,kernelMatrix);
    PlanarImage edge = JAI.create("convolve", input, kernel);
    PlanarImage output = JAI.create("invert", edge);
    JFrame frame = new JFrame();    frame.setTitle("Imagem "+args[0]+" invertida");
    frame.getContentPane().add(new DisplayTwoSynchronizedImages(input,output));
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();          frame.setVisible(true);
    }
  }
```

JAI: Exemplo de operadores

JAI: Operadores

- Operadores com JAI.create():
  - Manipulação de bandas: bandmerge, bandselect, bandcombine.
  - Geometria básica: crop, rotate, scale, shear, translate, transpose.
  - Modificação de valores globais: binarize, clamp, colorquantizer, lookup, colorconvert.
  - Aritmética e lógica: invert, add, subtract, multiply, divide, and, or, xor.
  - Regiões e MM: dilate, erode, boxfilter, convolve.
  - Estatística: histogram, extrema, mean.
  - Muitos outros!
- É possível escrever seus próprios operadores e registrar para uso com JAI.create().

JAI: Visualização

- Componente DisplayJAI, não é parte da API mas é distribuído com a mesma.
  - Uso: criamos uma instância de DisplayJAI com um Planar/TiledImage, adicionamos em um componente, opcionalmente com um JScrollPane.
- Alternativas: método PlanarImage.getAsBufferedImage(), TiledImage.getSubImage(), getTile, etc.
- Outros componentes *3rd-party*.

JAI: Usando DisplayJAI como base

```
package jai;

import java.awt.GridLayout;
import java.awt.event.*;
import java.awt.image.RenderedImage;
import javax.swing.*;
import com.sun.media.jai.widget.DisplayJAI;

public class DisplayTwoSynchronizedImages extends JPanel implements AdjustmentListener
  {
  protected DisplayJAI dj1;
  protected DisplayJAI dj2;
  protected JScrollPane jsp1;
  protected JScrollPane jsp2;

  public DisplayTwoSynchronizedImages(RenderedImage im1,RenderedImage im2)
    {
    super();
    setLayout(new GridLayout(1,2));
    dj1 = new DisplayJAI(im1);          dj2 = new DisplayJAI(im2);
    jsp1 = new JScrollPane(dj1);        jsp2 = new JScrollPane(dj2);
    add(jsp1);            add(jsp2);
    jsp1.getHorizontalScrollBar().addAdjustmentListener(this);
    jsp1.getVerticalScrollBar().addAdjustmentListener(this);
    jsp2.getHorizontalScrollBar().addAdjustmentListener(this);
    jsp2.getVerticalScrollBar().addAdjustmentListener(this);
    }
```

JAI: Usando DisplayJAI como base

```
public void setImage1(RenderedImage newimage)
  {
  dj1.set(newimage);
  repaint();
  }
public void setImage2(RenderedImage newimage)
  {
  dj2.set(newimage);
  repaint();
  }
public RenderedImage getImage1()
  {
  return dj1.getSource();
  }
public RenderedImage getImage2()
  {
  return dj2.getSource();
  }
public DisplayJAI getDisplayJAIComponent1()
  {
  return dj1;
  }
public DisplayJAI getDisplayJAIComponent2()
  {
  return dj2;
  }
```

JAI: Usando DisplayJAI como base

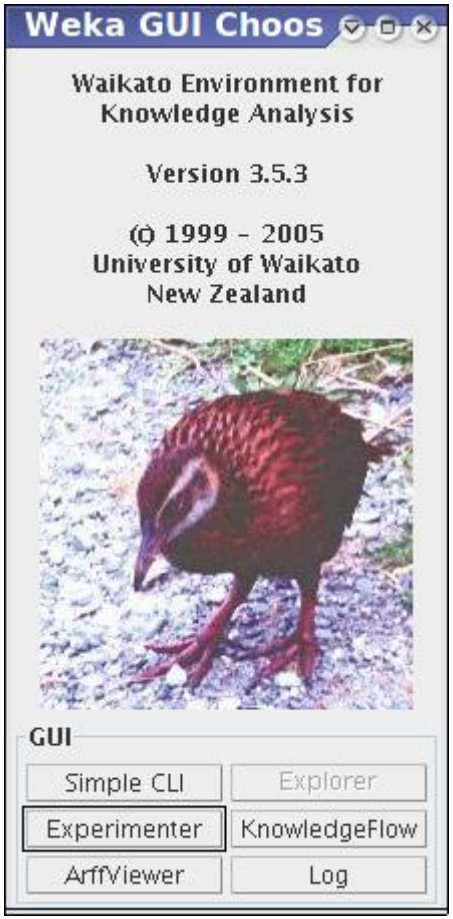
```
public void adjustmentValueChanged(AdjustmentEvent e)
  {
  if (e.getSource() == jsp1.getHorizontalScrollBar())
    {
    jsp2.getHorizontalScrollBar().setValue(e.getValue());
    }
  if (e.getSource() == jsp1.getVerticalScrollBar())
    {
    jsp2.getVerticalScrollBar().setValue(e.getValue());
    }
  if (e.getSource() == jsp2.getHorizontalScrollBar())
    {
    jsp1.getHorizontalScrollBar().setValue(e.getValue());
    }
  if (e.getSource() == jsp2.getVerticalScrollBar())
    {
    jsp1.getVerticalScrollBar().setValue(e.getValue());
    }
  }
}
```

- Fácil estender a idéia para N imagens.

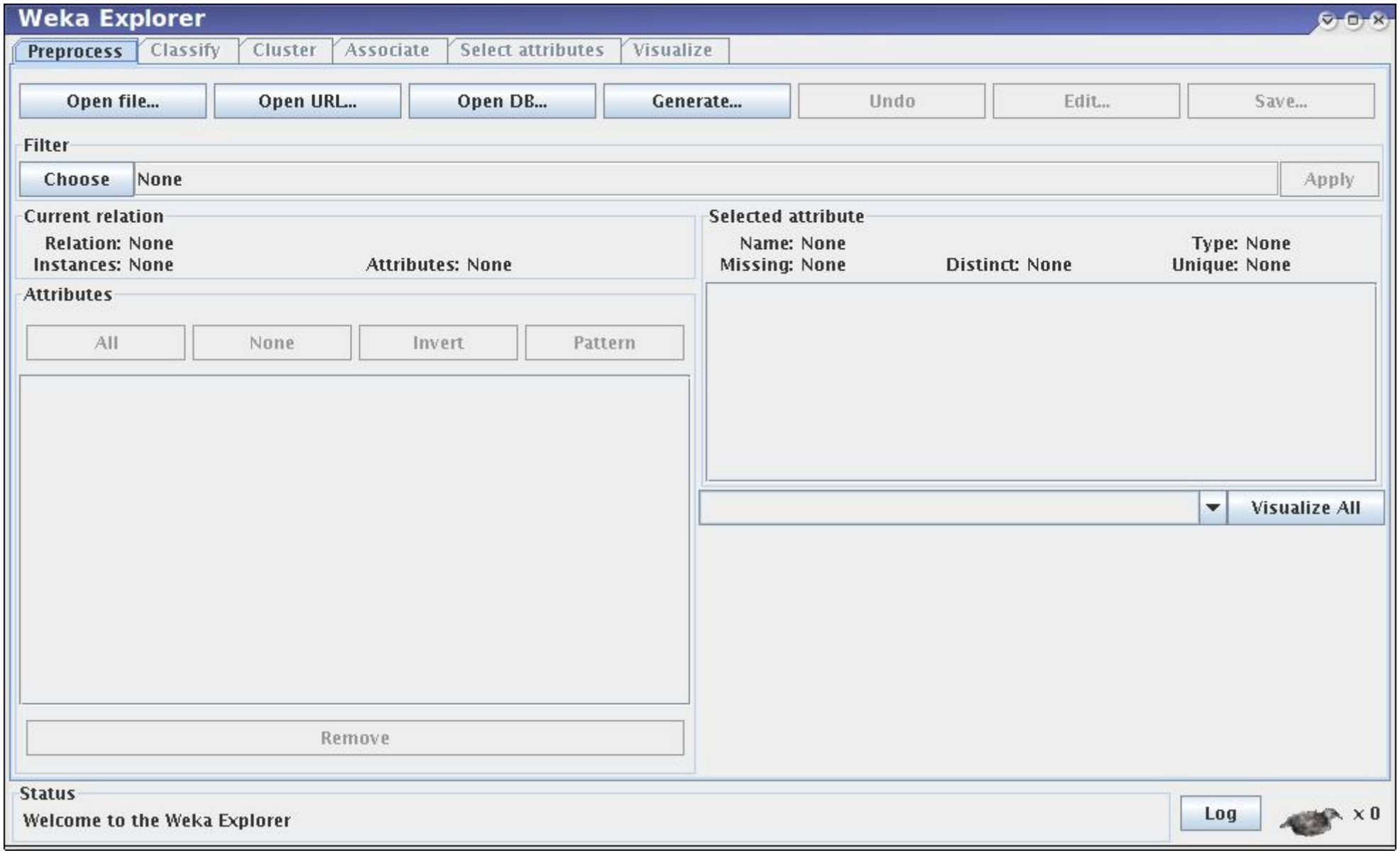
Data Mining: **Weka**

Weka

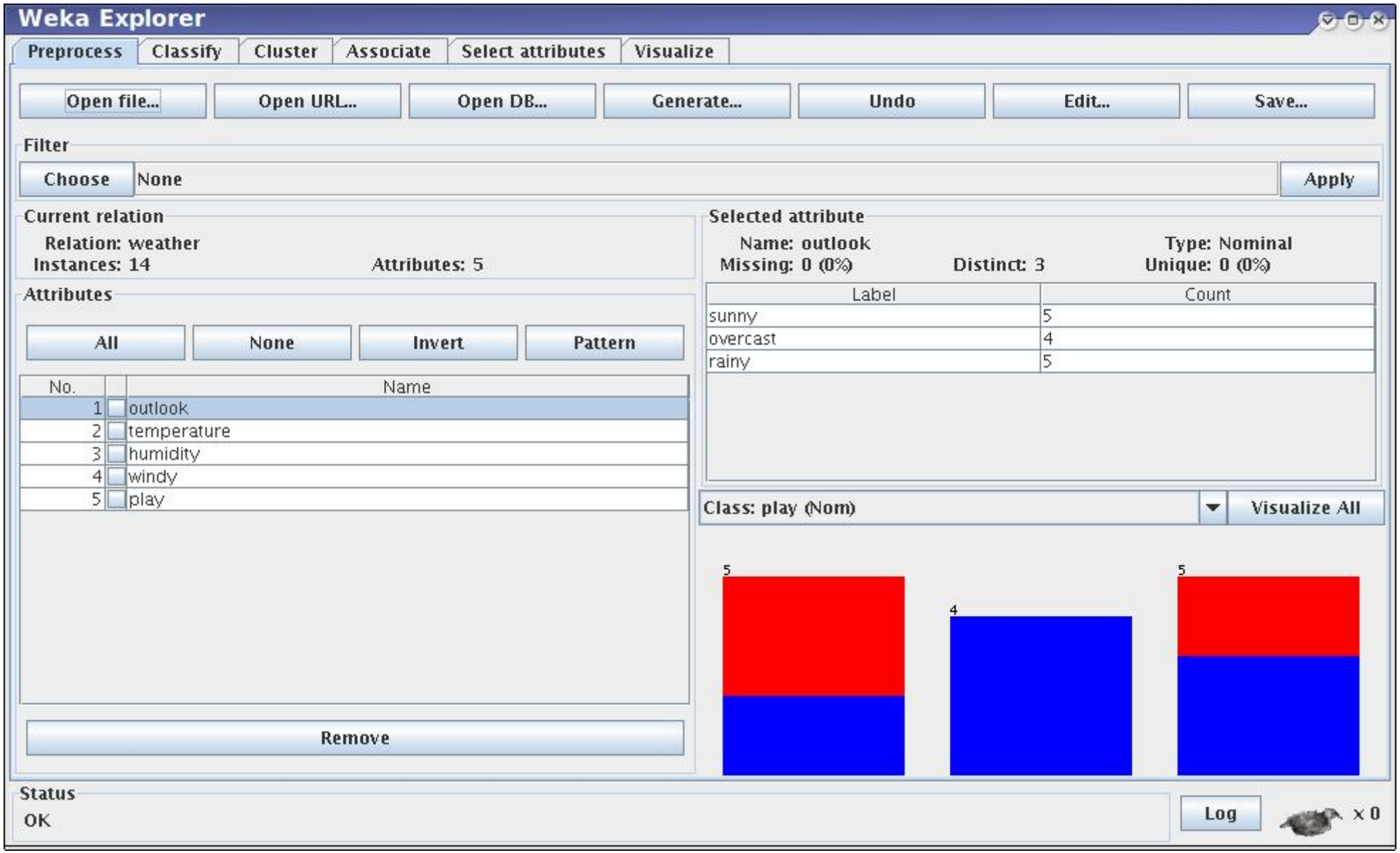
- *Waikato Environment for Knowledge Analysis*
- http://www.cs.waikato.ac.nz/ml/weka/index.html
- API e ambiente de testes com algoritmos de mineração de dados e aprendizado por computador.
  - Árvores de decisão, redes neurais, agrupamento, classificadores estatísticos, visualização básica...
  - *Ian H. Witten, Eibe Frank, Data Mining – Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann*, 2ª edição.



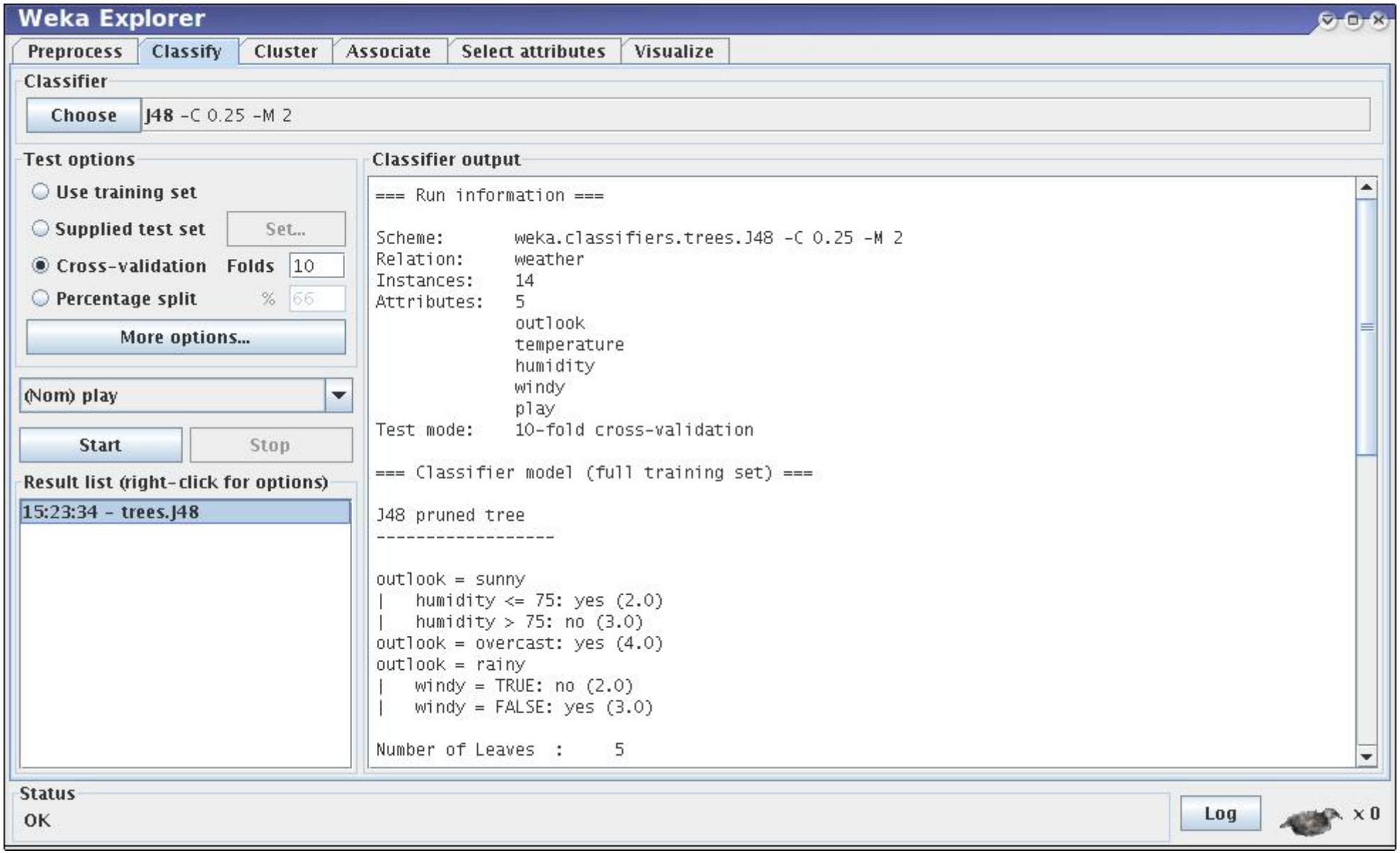
Weka – Explorer



Weka – Explorer

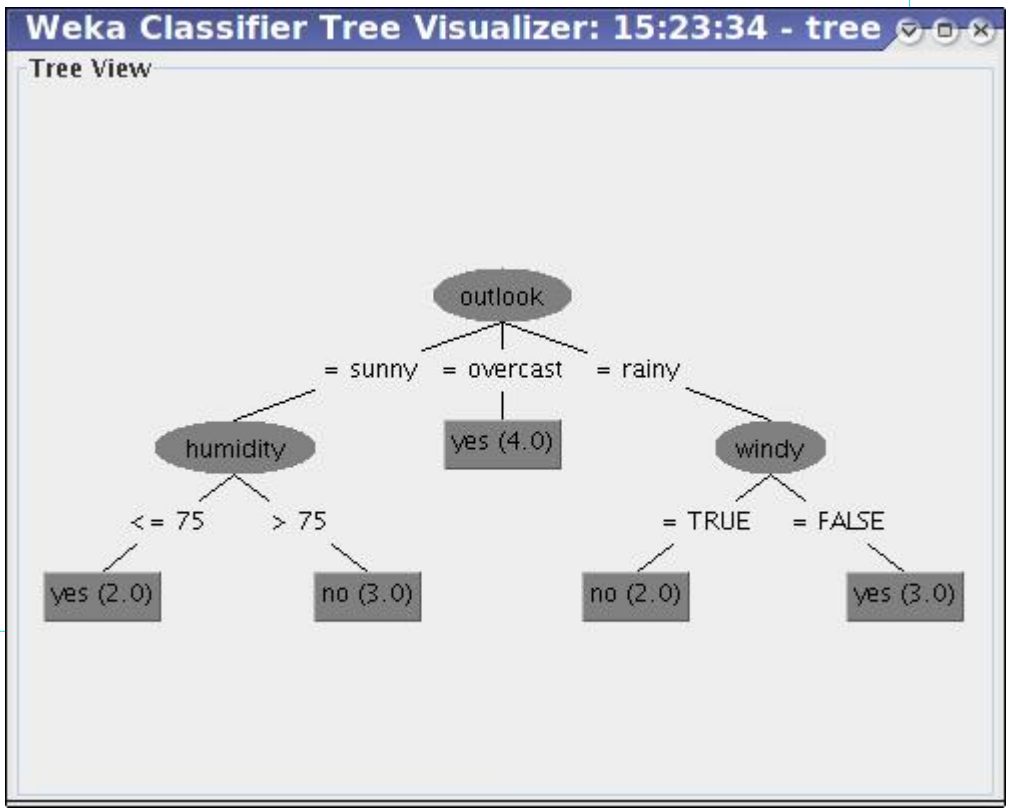


Weka – Explorer

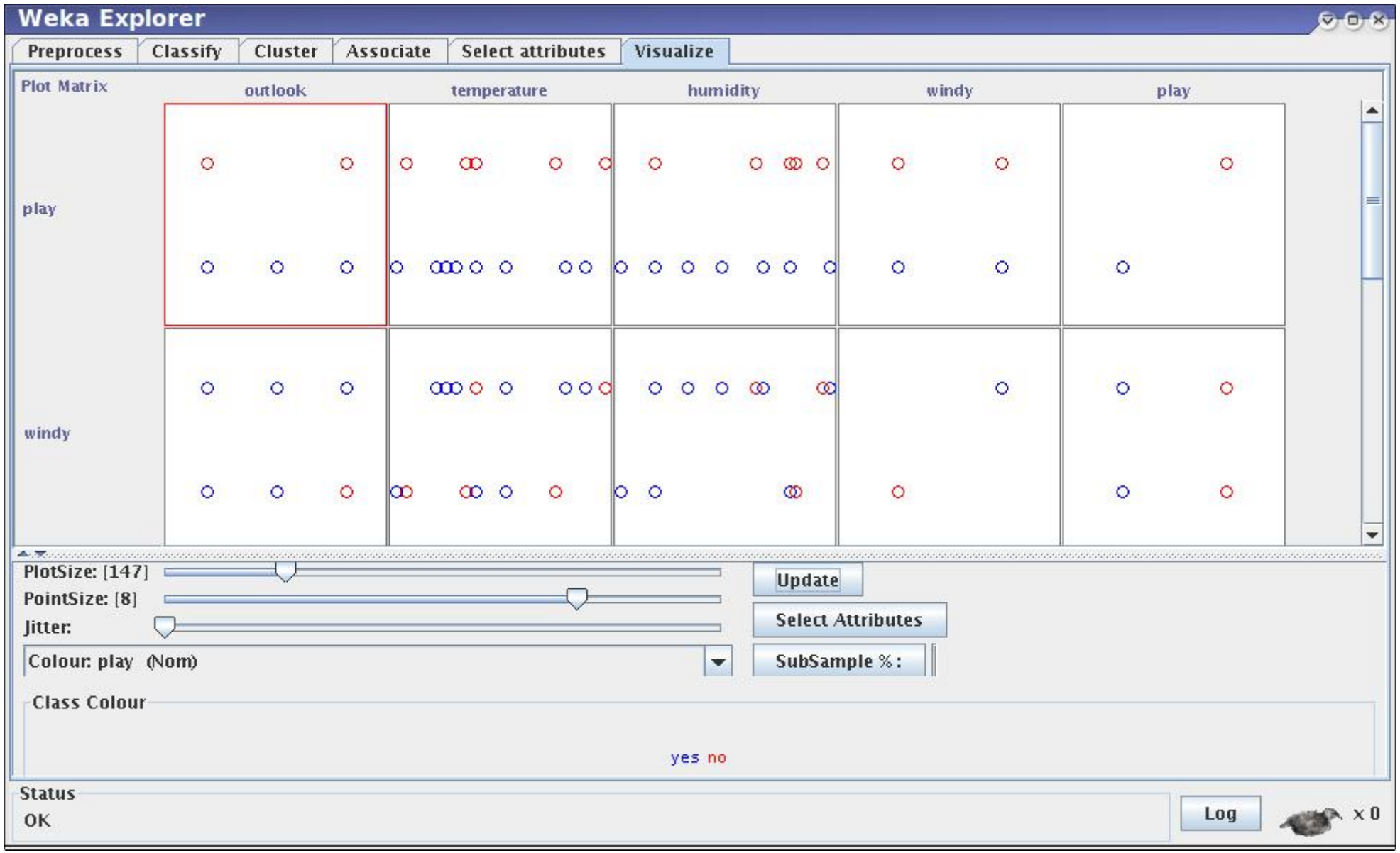


Weka – Explorer

```
=== Classifier model (full training set) ===

J48 pruned tree
------------------

outlook = sunny
|   humidity <= 75: yes (2.0)
|   humidity > 75: no (3.0)
outlook = overcast: yes (4.0)
outlook = rainy
|   windy = TRUE: no (2.0)
|   windy = FALSE: yes (3.0)

Number of Leaves  :     5
Size of the tree :  8


=== Confusion Matrix ===

 a b   <-- classified as
 7 2 | a = yes
 3 2 | b = no
```



Weka – Explorer



Weka – Formato de Dados

- .ARFF (*Attribute-Relation File Format*): texto
  - Relação, atributos, dados, comentários.

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
% outras entradas deletadas...
rainy,75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```

Weka – API – Lendo .ARFF

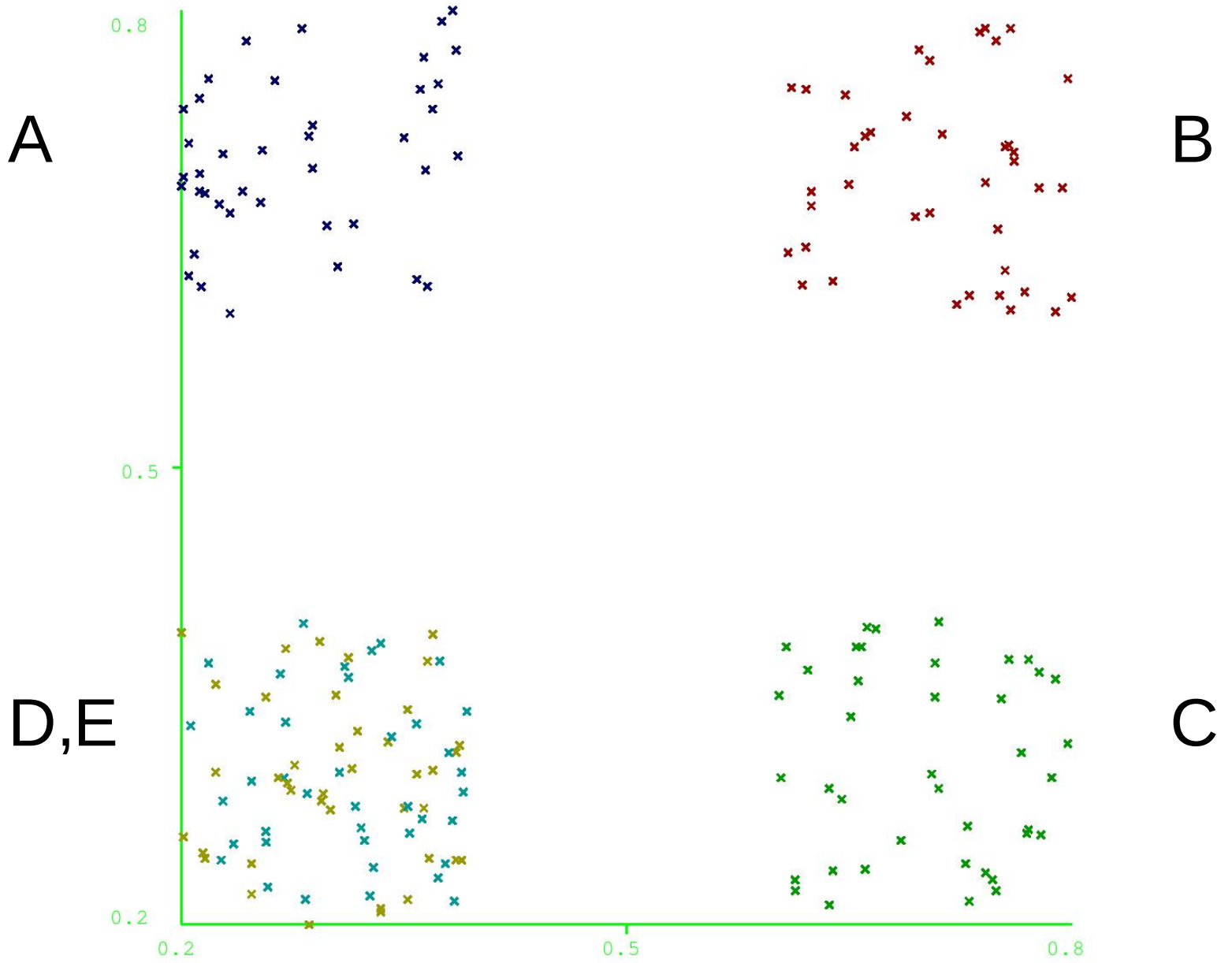
```
package weka;

import java.io.*;
import weka.core.*;

public class LeARFF
  {
  public static void main(String[] args) throws IOException
    {
    FileReader reader = new FileReader("../Dados/weather-m.arff");
    Instances instances = new Instances(reader);
    System.out.println(instances.relationName());
    System.out.println(instances.numAttributes()+" attributes");
    for(int i=0;i<instances.numAttributes();i++)
      {
      String name = instances.attribute(i).name();
      int type = instances.attribute(i).type();
      String typeName = "";
      switch(type)
        {
        case Attribute.NUMERIC: typeName = "Numeric"; break;
        case Attribute.NOMINAL: typeName = "Nominal"; break;
        case Attribute.STRING: typeName = "String"; break;
        }
      System.out.println(name+" type "+typeName);
      }
```

Weka – API – Lendo .ARFF

```
    for(int i=0;i<instances.numInstances();i++)
      {
      Instance instance = instances.instance(i);
      System.out.println((i+1)+": "+instance+" missing? "+instance.hasMissingValue());
      }
    }
  }
```

```
weather
5 attributes
outlook type Nominal
temperature type Numeric
humidity type Numeric
windy type Nominal
play type Nominal
1: sunny,85,85,FALSE,no missing? false
2: sunny,80,90,TRUE,no missing? false
3: overcast,83,?,FALSE,yes missing? true
4: rainy,70,96,FALSE,yes missing? false
5: rainy,68,80,FALSE,yes missing? false
6: rainy,65,70,TRUE,no missing? false
7: overcast,64,65,TRUE,yes missing? false
8: sunny,72,95,?,no missing? true
9: sunny,69,70,FALSE,yes missing? false
10: rainy,75,80,FALSE,yes missing? false
11: sunny,?,70,TRUE,yes missing? true
12: overcast,72,90,TRUE,yes missing? false
13: overcast,81,75,FALSE,yes missing? false
14: ?,71,91,TRUE,no missing? true
```

Weka – API – Criando .ARFF



Weka – API – Criando .ARFF

```
package weka;

import weka.core.*;

public class CriaARFF
  {
  public static void main(String[] args)
    {
    // Criamos os atributos, dois numéricos e um nominal.
    Attribute x = new Attribute("x");
    Attribute y = new Attribute("y");
    FastVector classesLabels = new FastVector(5);
    classesLabels.addElement("A");
    classesLabels.addElement("B");
    classesLabels.addElement("C");
    classesLabels.addElement("D");
    classesLabels.addElement("E");
    Attribute classes = new Attribute("classes",classesLabels);
    // Vetor com informações dos atributos.
    FastVector attributes = new FastVector(3);
    attributes.addElement(x);
    attributes.addElement(y);
    attributes.addElement(classes);
    // Conjunto de instâncias.
    Instances instances = new Instances("random",attributes,0);
```

Weka – API – Criando .ARFF

```
    for(int i=0;i<40;i++) // 200 instâncias, 40 de cada uma das 5 classes
      {
      Instance inst = new Instance(3);
      // Classe A.
      inst.setValue(x,0.2+0.2*Math.random());  inst.setValue(y,0.6+0.2*Math.random());
      inst.setValue(classes,"A");
      instances.add(inst);
      // Classe B.
      inst.setValue(x,0.6+0.2*Math.random());  inst.setValue(y,0.6+0.2*Math.random());
      inst.setValue(classes,"B");
      instances.add(inst);
      // Classe C.
      inst.setValue(x,0.6+0.2*Math.random());  inst.setValue(y,0.2+0.2*Math.random());
      inst.setValue(classes,"C");
      instances.add(inst);
      // Classes D e E.
      inst.setValue(x,0.2+0.2*Math.random());  inst.setValue(y,0.2+0.2*Math.random());
      inst.setValue(classes,"D");
      instances.add(inst);
      inst.setValue(x,0.2+0.2*Math.random());  inst.setValue(y,0.2+0.2*Math.random());
      inst.setValue(classes,"E");
      instances.add(inst);
      }
    // Podemos gravar com BufferedWriter.
    System.out.println(instances);
    }
}
```

Weka – API – Classificador em Java

```
package weka;
import java.io.*;
import weka.core.*;
import weka.classifiers.Evaluation;
import weka.classifiers.trees.J48;

public class ExecutaJ48
  {
  public static void main(String[] args)
    {
    try
      {
      FileReader reader = new FileReader("/usr/local/weka-3-5-3/data/weather.arff");
      Instances instances = new Instances(reader);
      instances.setClassIndex(4);
      J48 thisClassifier = new J48();
      thisClassifier.buildClassifier(instances);
      System.out.println(thisClassifier.toString());
      // Avalia resultados.
      Evaluation evaluation = new Evaluation(instances);
      evaluation.evaluateModel(thisClassifier,instances);
      System.out.print(evaluation.correct()+"/");
      System.out.println(instances.numInstances());
      }
    catch (Exception e)  {  System.err.println(e.getMessage());  }
    }
  }
```

Weka – API – Agrupamento

```
package weka;
import java.io.FileReader;
import weka.clusterers.SimpleKMeans;
import weka.core.Instances;
public class ManualKMeans
  {
  public static void main(String[] args)
    {
    try
      {
      FileReader reader = new FileReader("../Dados/random.arff");
      Instances instances = new Instances(reader);
      // Ignoramos os atributos para criar os clusters
      instances.deleteAttributeAt(2);
      // Criamos o KMeans.
      SimpleKMeans thisClusterer = new SimpleKMeans();
      // Executamos com 2..10 clusters.
      for(int c=2;c<=10;c++)
        {
        thisClusterer.setNumClusters(c);
        thisClusterer.buildClusterer(instances);
        System.out.println(c+":"+thisClusterer.getSquaredError());
        }
      }
    catch (Exception e) { System.out.println(e.getMessage()); }
    }
  }
```

```
2:24.119017133063014
3:15.408290575637777
4:3.8676656715642466
5:3.6471327646445766
6:3.0839521558223
7:2.626022027427651
8:2.267438101027189
9:2.1741394171352475
10:1.918261459173919
```

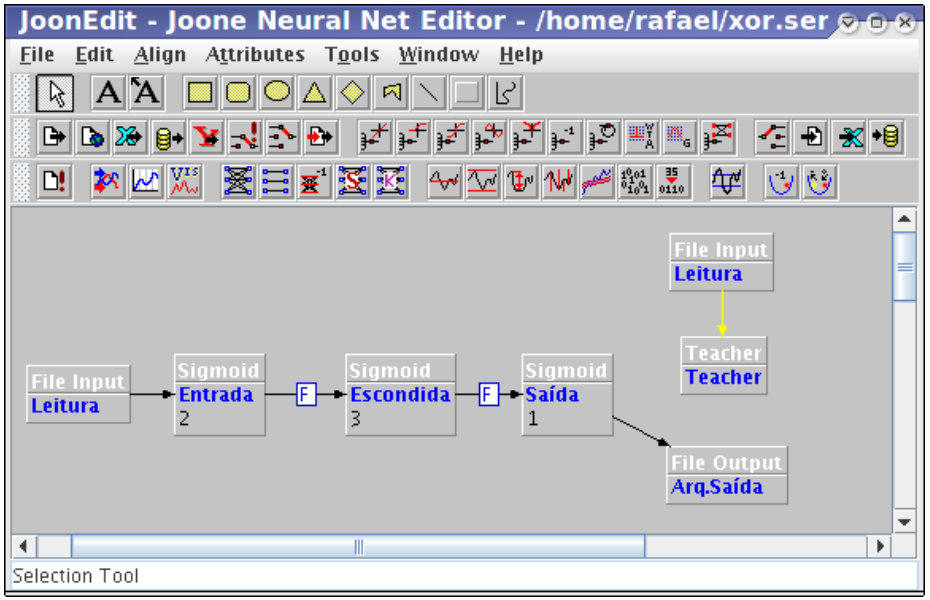
Redes Neurais: **JOONE**

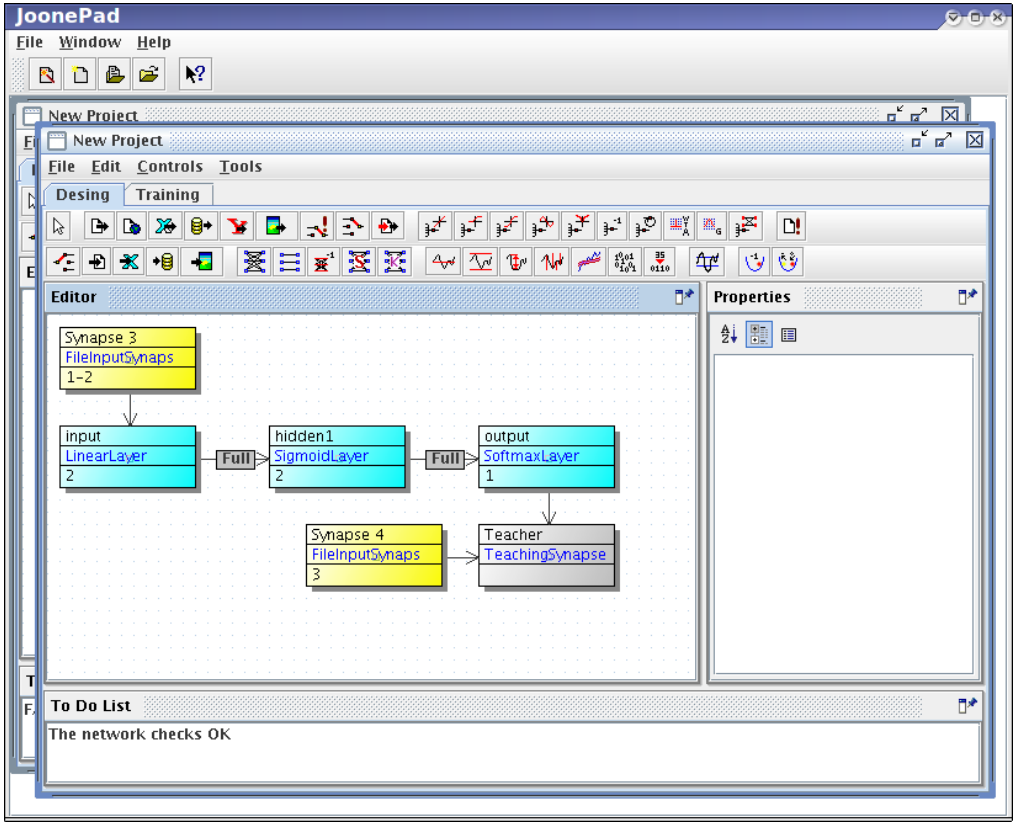
JOONE

- http://www.jooneworld.com/
- Ambiente gratuito para criação, testes e exploração de redes neurais, simples o suficiente para uso básico e poderoso o suficiente para uso profissional.
- Originalmente projeto de uma pessoa, tem atraído vários colaboradores.

JOONE

- Interface gráfica original: JoonEdit
- Nova interface: JoonePad (*download* separado, agora com *wizards*!)





JOONE: Exemplo XOR

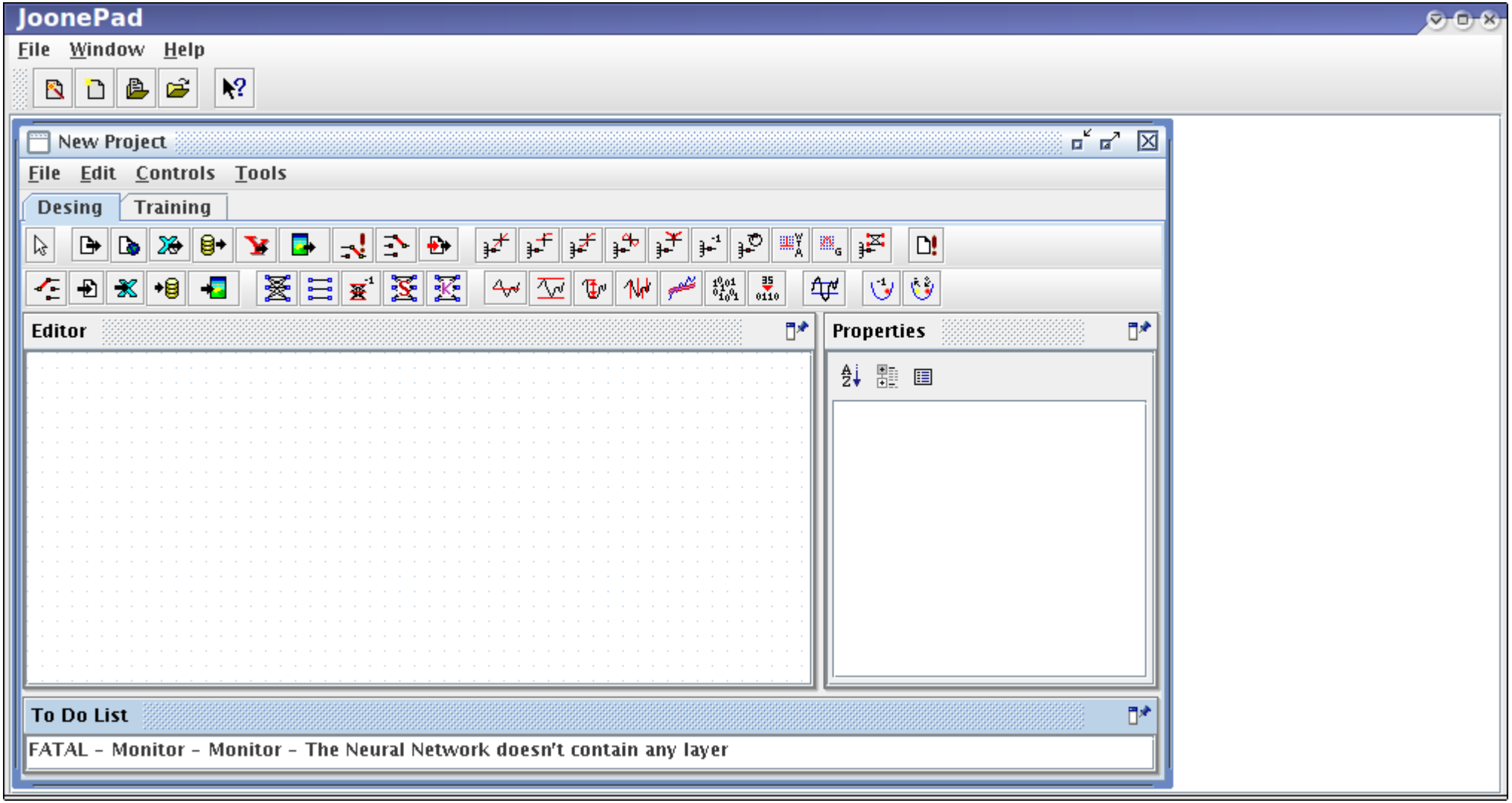
- Exemplo clássico de redes neurais: aprender a função XOR.
- Entradas e saídas:

```
0.0;0.0;0.0
0.0;1.0;1.0
1.0;0.0;1.0
1.0;1.0;0.0
```

- Usaremos o Joo[illegible]s *wizards*!
- Precisamos do *Java Web Start*
- http://www.jooneworld.com/joonepad/ws/JoonePad.jnlp

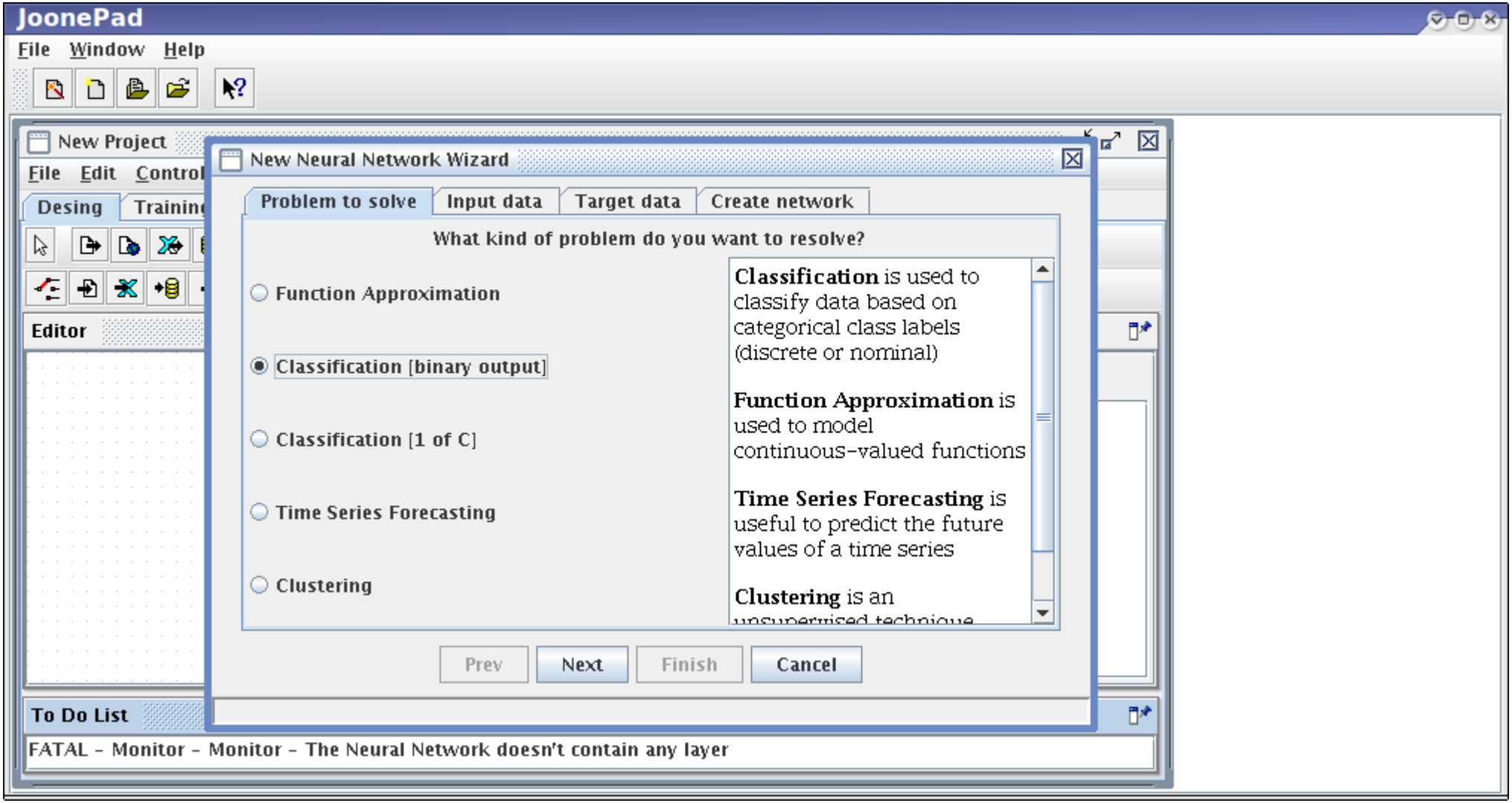
JOONE: Exemplo XOR

- JoonePad: Novo Projeto (vazio). Usaremos o *wizard*...



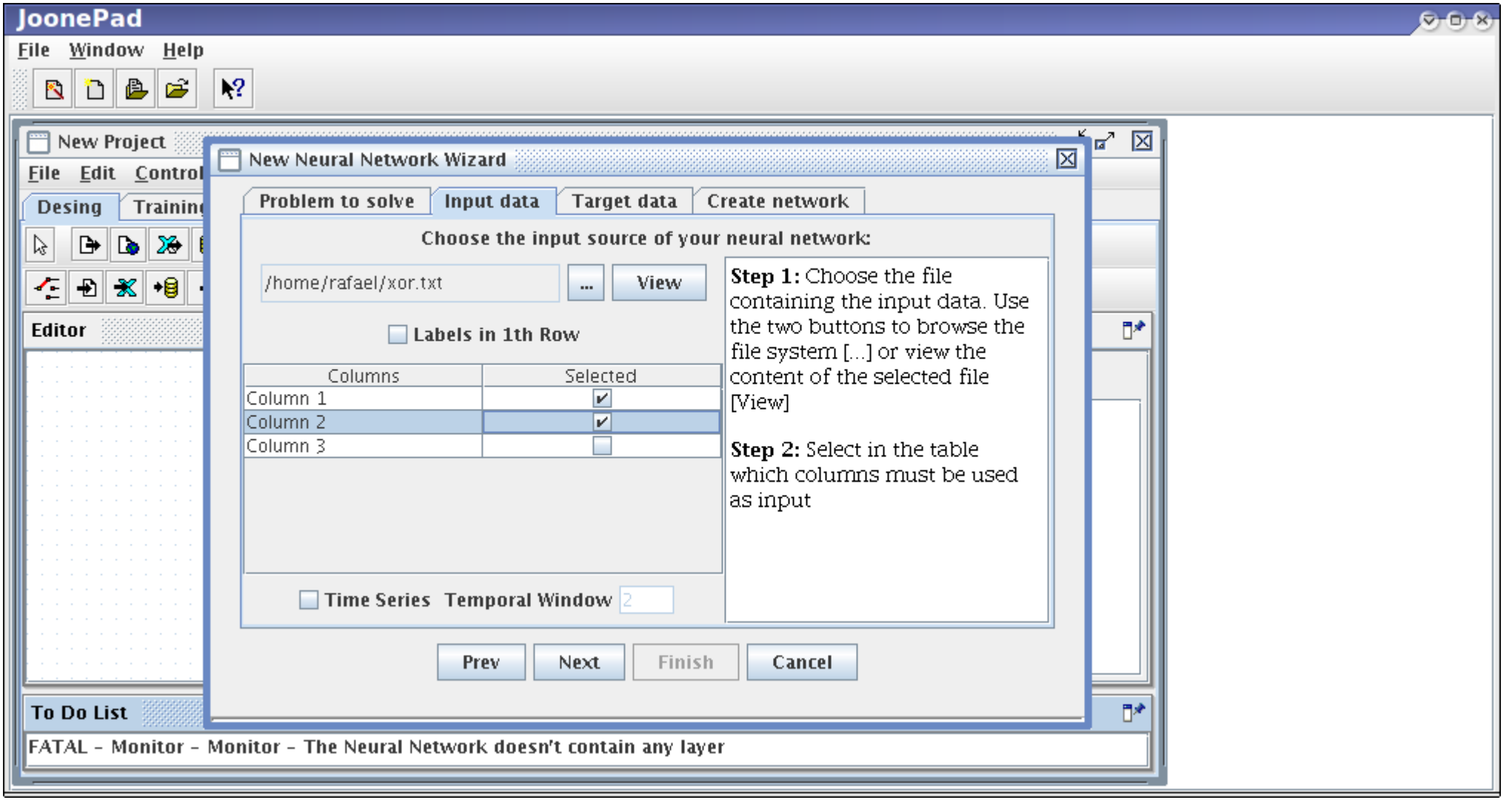
JOONE: Exemplo XOR

- JoonePAD: *Wizard* (Que tarefa será executada?)



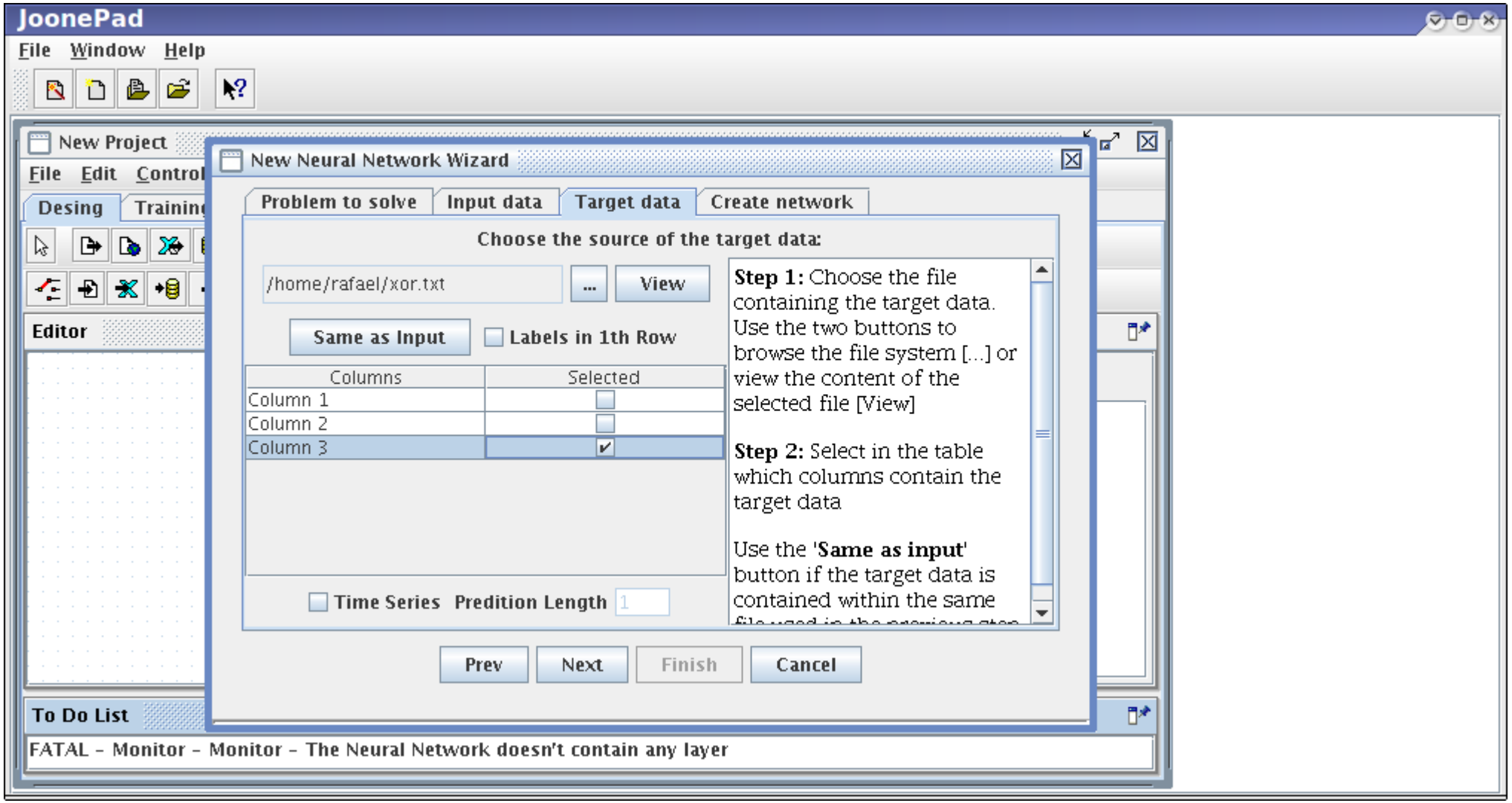
JOONE: Exemplo XOR

- JoonePAD: *Wizard* (Que dados de treinamento? Que estrutura?)



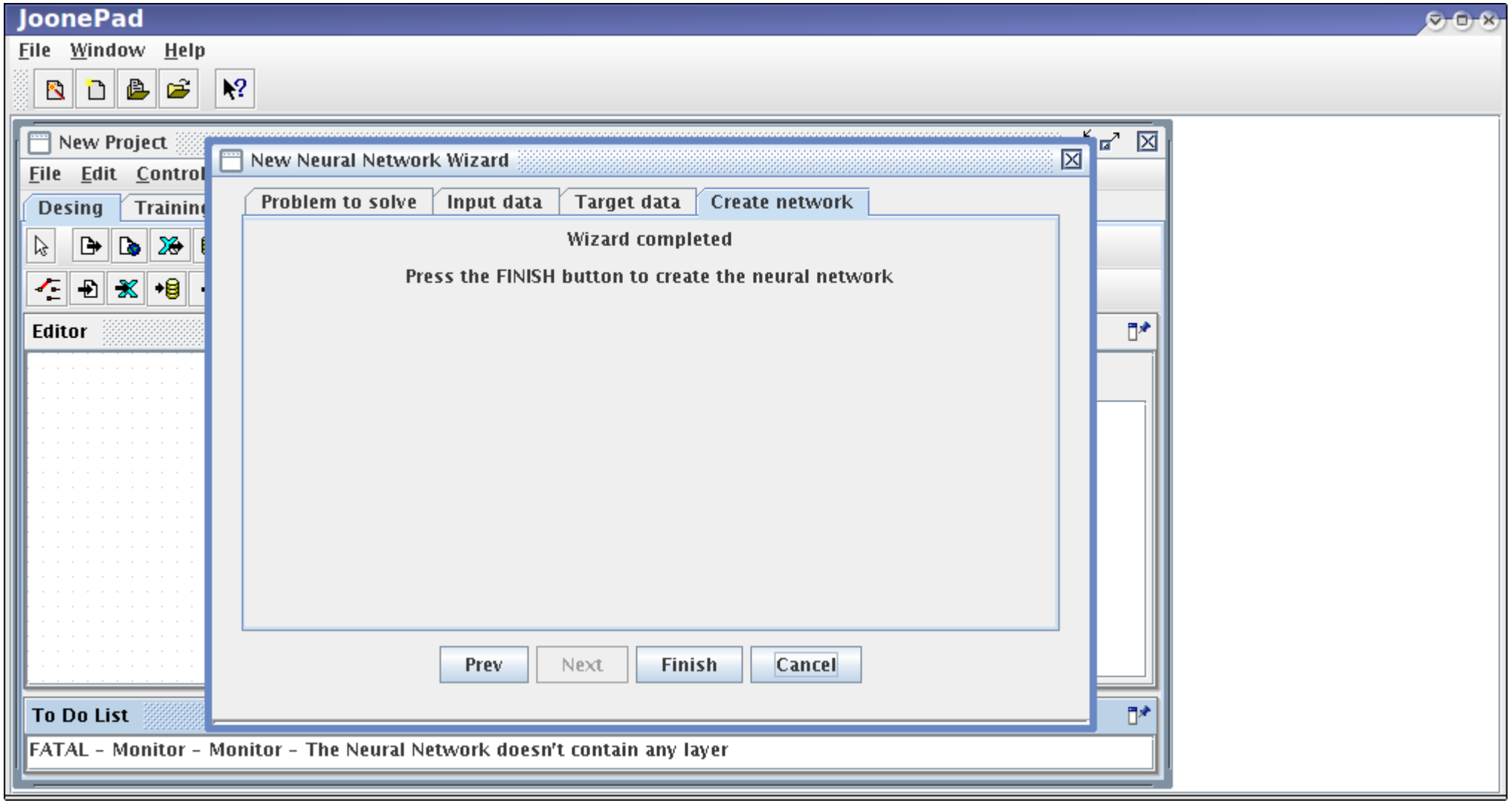
JOONE: Exemplo XOR

- JoonePAD: *Wizard* (Que classes? Que estrutura?)



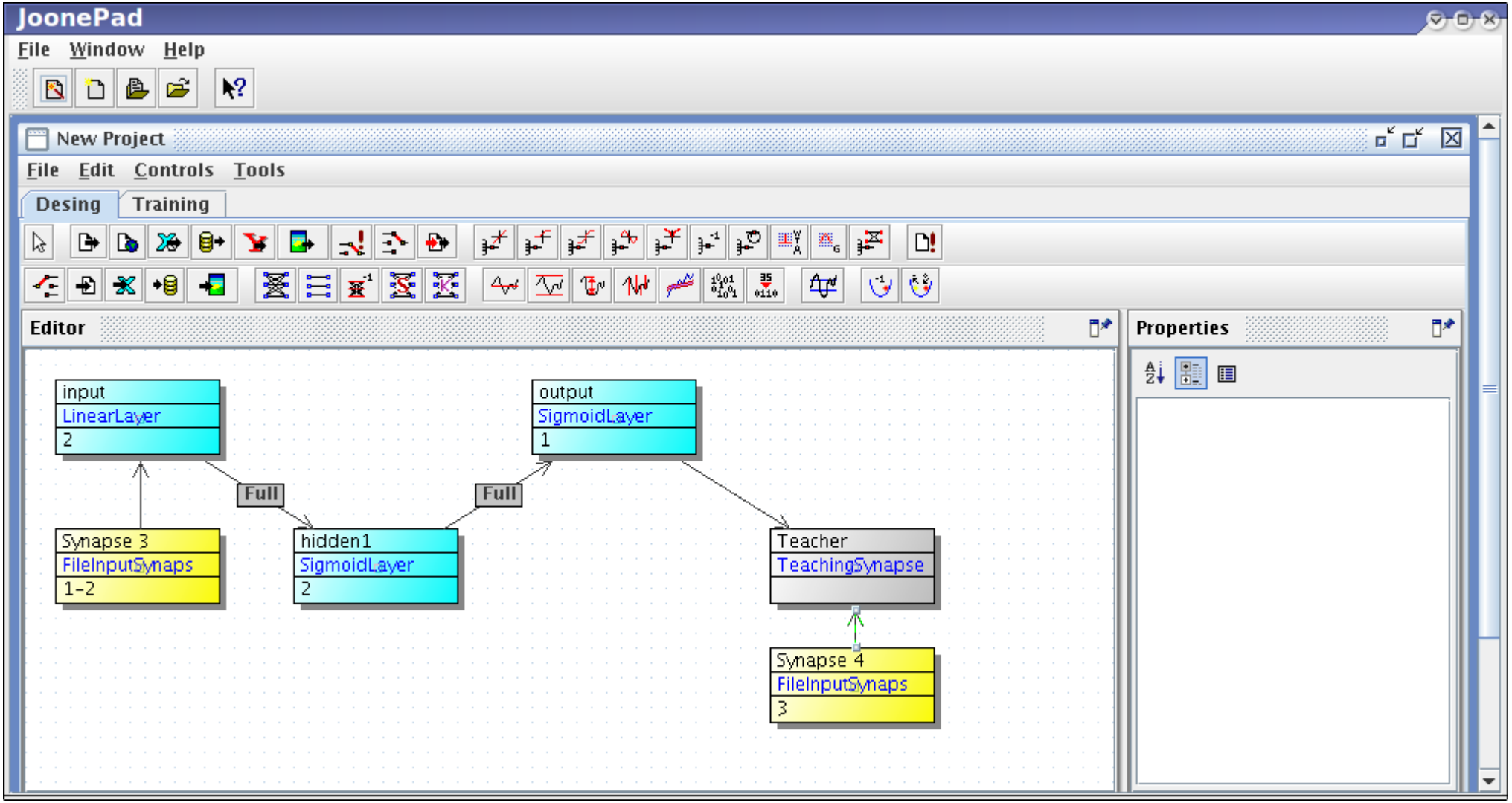
JOONE: Exemplo XOR

- JoonePAD: *Wizard* (Criando a rede)



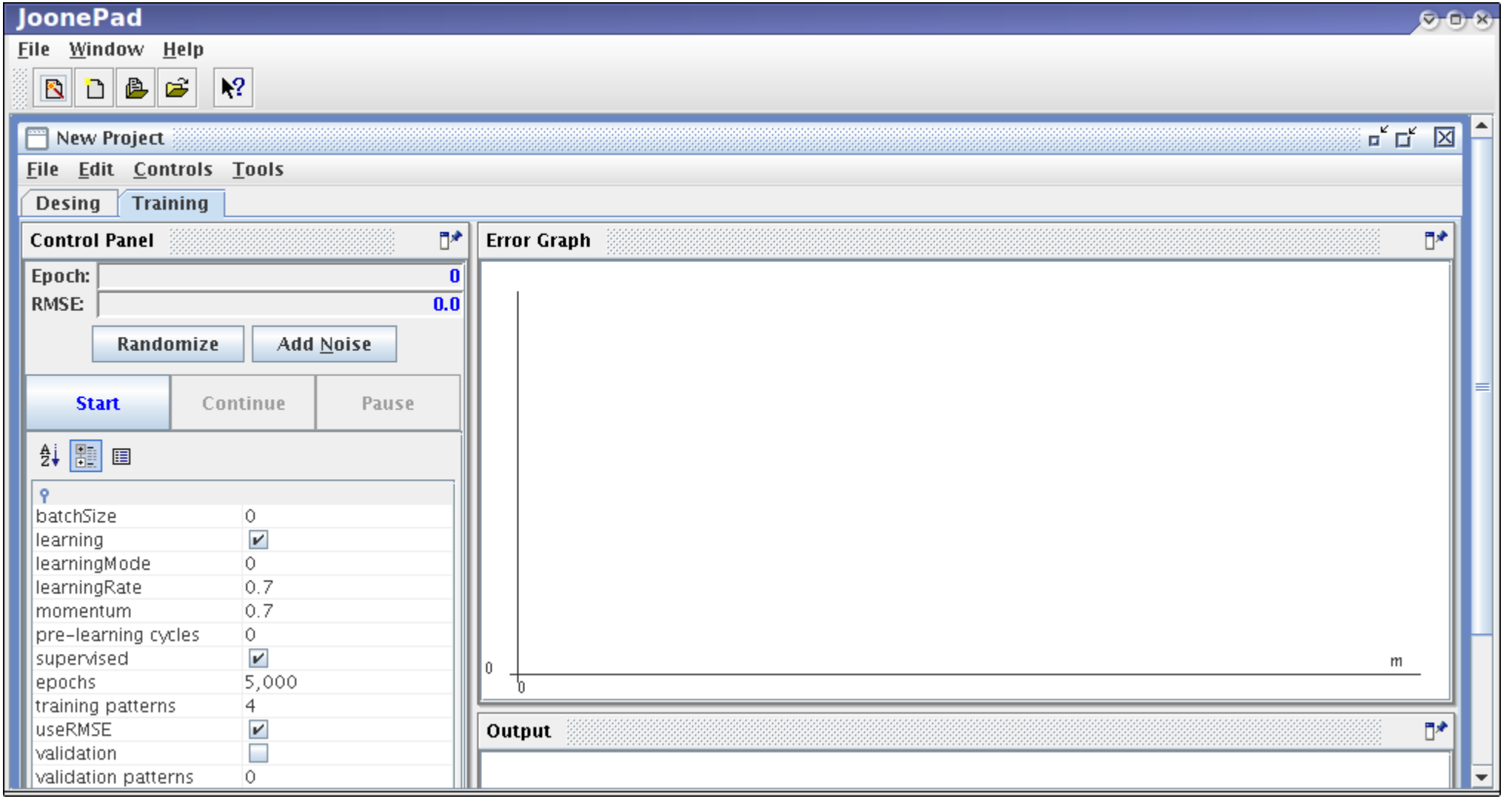
JOONE: Exemplo XOR

- JoonePAD: Usando a rede



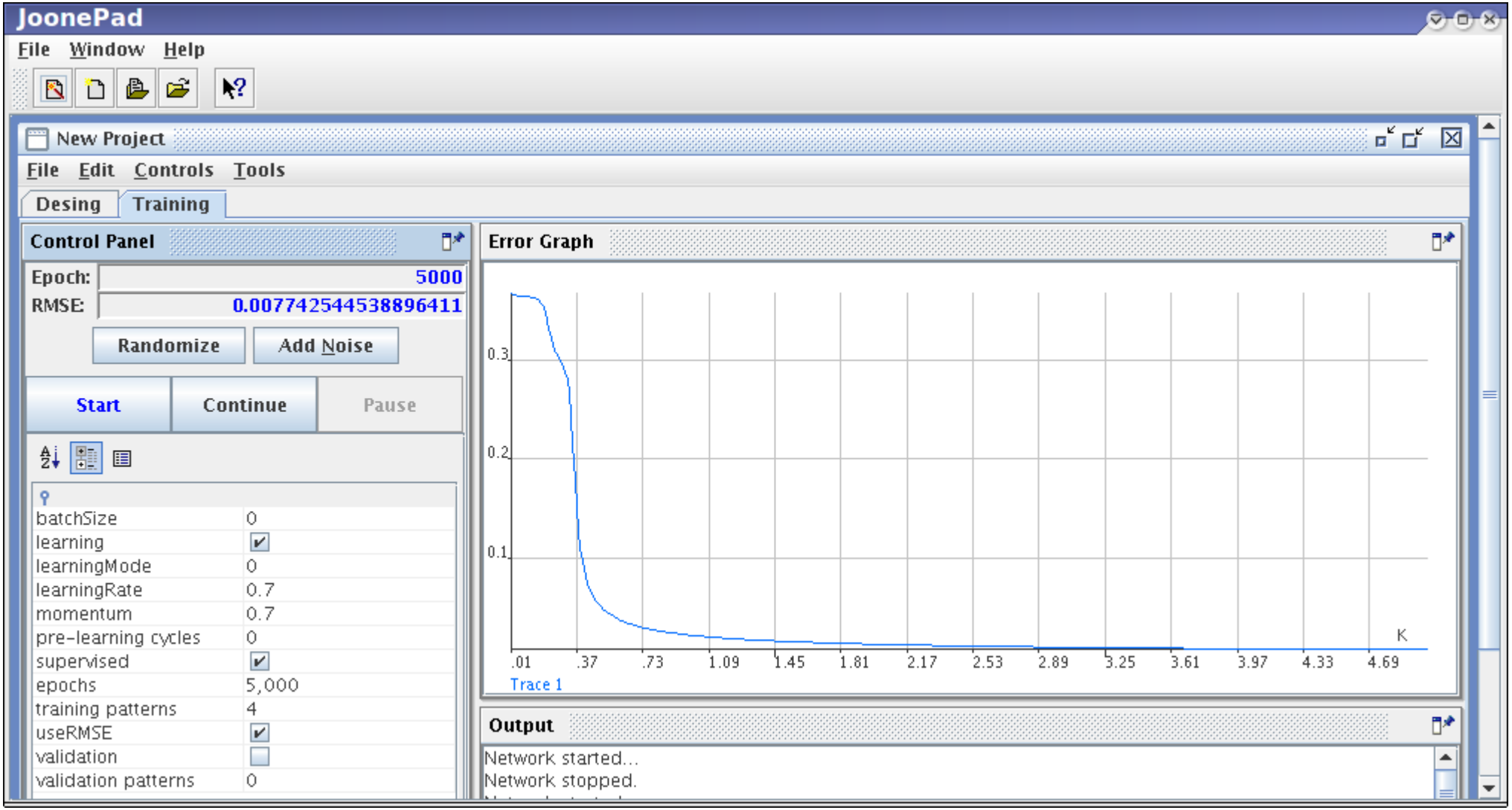
JOONE: Exemplo XOR

- JoonePAD: Usando a rede (treinamento)



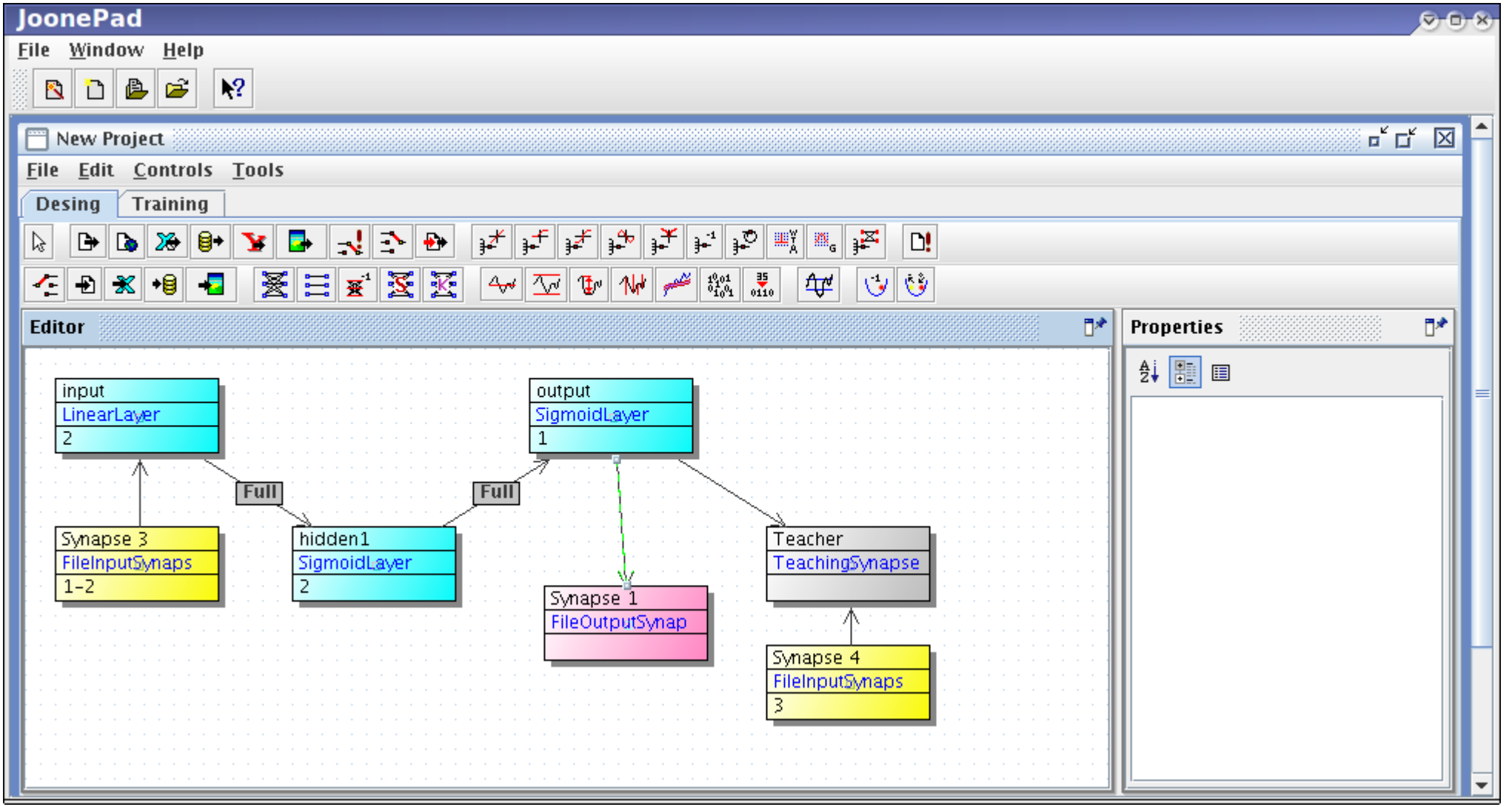
JOONE: Exemplo XOR

- JoonePAD: Usando a rede (treinamento)



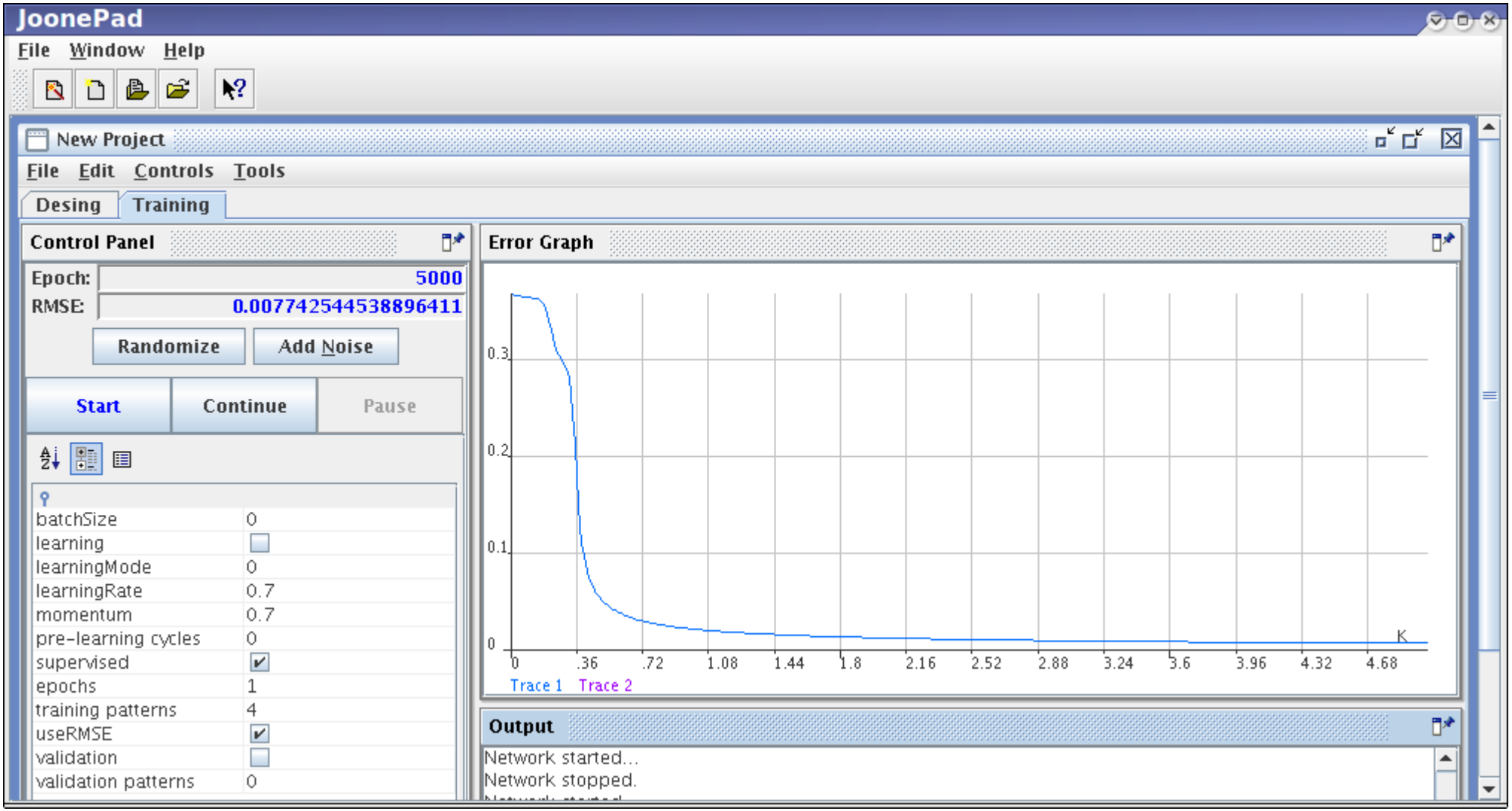
JOONE: Exemplo XOR

- JoonePAD: Usando a rede (classificação)



JOONE: Exemplo XOR

- JoonePAD: Usando a rede (classificação)



JOONE: Exemplo XOR

- Resultados:

I1	I2	Classe	Resultado
0	0	0	0.0099557
0	1	1	0.9895956
1	0	1	0.9895957
1	1	0	0.0127975

JOONE: Exemplos com programação

- Usamos o exemplo do sítio do Joone como base.
- Ainda o problema do XOR.

```
package joone;

import org.joone.engine.*;
import org.joone.engine.learning.TeachingSynapse;
import org.joone.io.*;

public class RedeXOR implements NeuralNetListener
  {
  public RedeXOR()
    {
    // Criamos as três camadas para a rede.
    // Primeiro a de entrada:
    LinearLayer entrada = new LinearLayer();
    entrada.setRows(2);
    // Agora a escondida:
    SigmoidLayer escondida = new SigmoidLayer();
    escondida.setRows(5);
    // Agora a de saída:
    SigmoidLayer saída = new SigmoidLayer();
    saída.setRows(1);
```

JOONE: Exemplos com programação

```
// Criamos conexões (sinapses) entre as camadas.
FullSynapse ent_esc = new FullSynapse();
FullSynapse esc_sai = new FullSynapse();
// Conectamos as camadas com as sinapses.
entrada.addOutputSynapse(ent_esc);
escondida.addInputSynapse(ent_esc);
escondida.addOutputSynapse(esc_sai);
saída.addInputSynapse(esc_sai);
// Criamos um Monitor para fazer o treinamento/uso da rede.
Monitor monitor = new Monitor();
// As camadas devem ter uma referência do monitor.
entrada.setMonitor(monitor);
escondida.setMonitor(monitor);
saída.setMonitor(monitor);
// A própria aplicação "escutará" eventos do Monitor.
monitor.addNeuralNetListener(this);

// Criamos uma associação/filtro de arquivo para a entrada.
FileInputSynapse dadosEntrada = new FileInputSynapse();
// Usaremos as duas primeiras colunas
dadosEntrada.setAdvancedColumnSelector("1,2");
// Usaremos este arquivo de entrada
dadosEntrada.setFileName("/tmp/xor.txt");
// Estes dados serão usados na camada de entrada
entrada.addInputSynapse(dadosEntrada);
```

JOONE: Exemplos com programação

```
// Criamos uma estrutura para conter dados de treinamento
TeachingSynapse sinapseTreinamento = new TeachingSynapse();
// Criamos uma associação/filtro de arquivo para os dados esperados
FileInputSynapse dadosTreinamento = new FileInputSynapse();
// Usaremos a terceira coluna
dadosTreinamento.setAdvancedColumnSelector("3");
// Usaremos este arquivo de entrada
dadosTreinamento.setFileName("/tmp/xor.txt");
// Associamos dados à estrutura para treinamento
sinapseTreinamento.setDesired(dadosTreinamento);
// Esta estrutura também deve ter uma referência ao monitor
sinapseTreinamento.setMonitor(monitor);
// A camada de saída deve ser associada à estrutura de treinamento
saída.addOutputSynapse(sinapseTreinamento);
```

JOONE: Exemplos com programação

```
// Iniciamos o funcionamento das camadas
entrada.start();
escondida.start();
saída.start();
// Configuramos o monitor que executará a rede
monitor.setLearningRate(0.7);
monitor.setMomentum(0.5);
monitor.setTrainingPatterns(4); // número de padrões
monitor.setTotCicles(2000); // quantas vezes usaremos os padrões para
                            // treinar a rede
monitor.setLearning(true); // rede deve ser treinada
monitor.Go(); // iniciamos a execuçã£o da rede (treinamento)
while(monitor.getCurrentCicle() > 0) { } // treinando, aguardamos...
```

JOONE: Exemplos com programação

```
// Assumimos que a rede está treinada.
// Criamos uma sinapse de saída e a associamos à um arquivo e à
// camada de saída
FileOutputSynapse saídaArquivo = new FileOutputSynapse();
saídaArquivo.setFileName("/tmp/xorout.txt");
saída.addOutputSynapse(saídaArquivo);
// Associamos a sinapse de saída ao monitor
saídaArquivo.setMonitor(monitor);
// Reiniciamos o funcionamento das camadas
entrada.start();
escondida.start();
saída.start();
// Executamos a rede somente uma vez, sem treiná-la novamente
monitor.setTotCicles(1);
monitor.setLearning(false);
monitor.Go(); // Iniciamos a execução da rede (classificação)
}
```

JOONE: Exemplos com programação

```
public void cicleTerminated(NeuralNetEvent nne) { }

public void errorChanged(NeuralNetEvent nne) { }

public void netStarted(NeuralNetEvent nne) { }

public void netStopped(NeuralNetEvent nne) { }

public void netStoppedError(NeuralNetEvent nne, String msg) { }

// Executa a aplicação (criando uma nova instância da classe)
public static void main(String[] args)
  {
  new RedeXOR();
  }

}
```

JOONE: Exemplos com programação

- Arquivo de entrada:

```
0.0;0.0;0.0
0.0;1.0;1.0
1.0;0.0;1.0
1.0;1.0;0.0
```

- Arquivo de saída:

```
0.023841741363915
0.9751083280392937
0.9748847696705181
0.03188682992907227 6
```

JOONE: Exemplos com programação

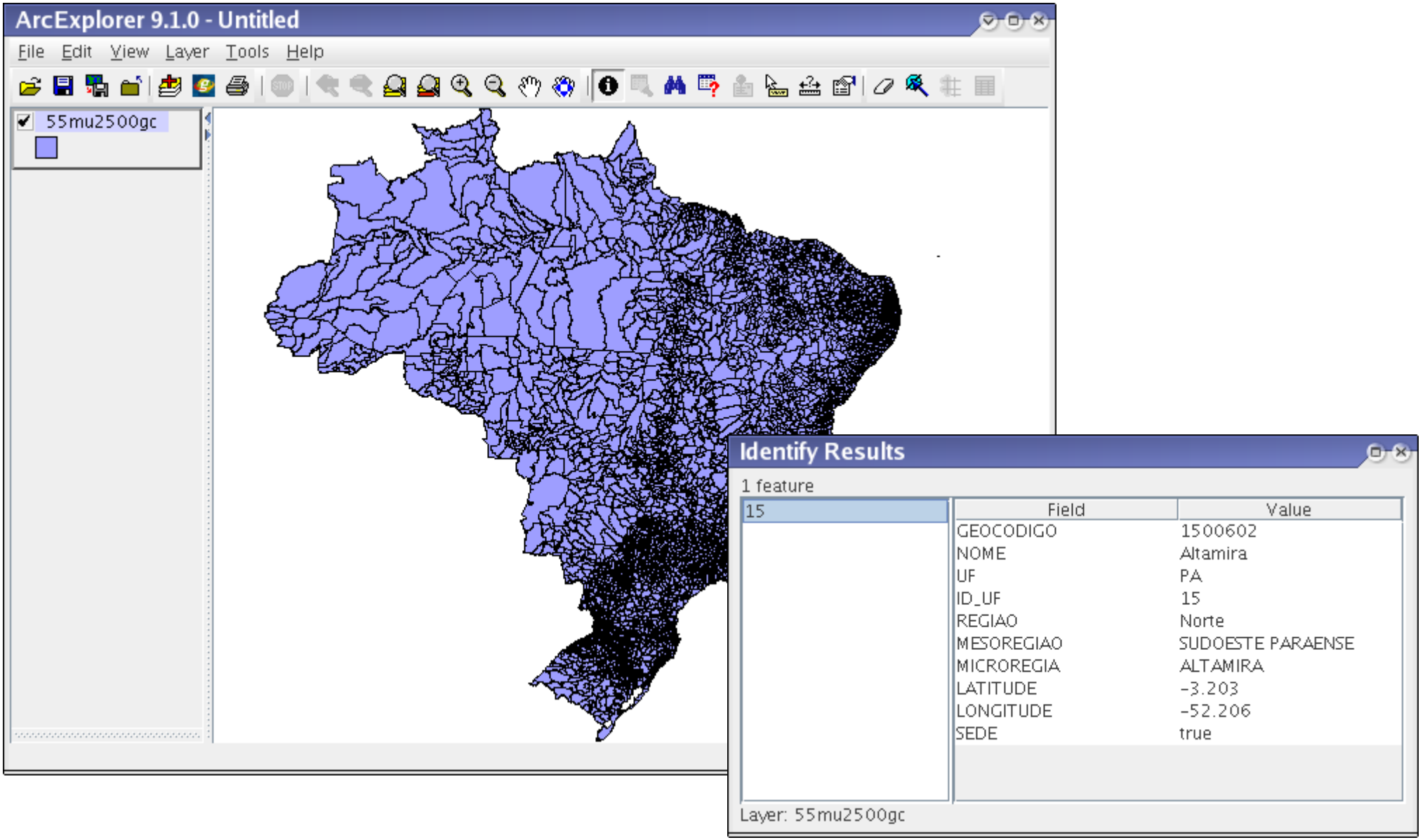
- Usando o exemplo do sítio do Joone como base foi relativamente simples reproduzir o exemplo do XOR.
- Tentei reproduzir o exemplo de classificação das íris, **absolutamente sem sucesso**.
  - Usando JoonePAD: OK!
  - Código gerado pelo JoonePAD: Inconclusivo, incompatível?
  - Inúmeras tentativas; resultados inaceitáveis.
  - Documentação insuficiente e inconclusiva.

Aplicações Georeferenciadas: **GeoTools**

Geotools

- **É uma API, não é uma solução GIS**.
- Permite o desenvolvimento de aplicações georeferenciadas.
- 100% Java: sem necessidade de instalação de .so ou .dll.
- Código aberto, implementa especificações do *Open Geospatial Consortium (OGC)*.
- Base para outros projetos.
- Algumas de suas classes usam a API JAI.
- http://docs.codehaus.org/display/GEOTOOLS/Home
- Veremos alguns exemplos usando *shapefiles*.

Geotools – Dados e Exemplo



http://www.ibge.gov.br/home/geociencias/default_prod.shtm

Geotools: Informações em um *shapefile*

- Vamos ler informações de um *shapefile*.

```
package geotools;

import java.io.*;
import java.net.URL;
import java.util.Iterator;
import org.geotools.data.FeatureSource;
import org.geotools.data.shapefile.ShapefileDataStore;
import org.geotools.feature.*;
import org.geotools.geometry.Geometry;

public class ShapefileInfo
  {
  public static void main(String[] args) throws IOException
    {
    // Onde estão nossos shapefiles?
    String baseDir = "/home/rafael/Docs/ELAC/Dados/E2500/";
    // Criamos uma URL com o shapefile que nos interessa.
    URL s = new File(baseDir+"Brasil/55mu2500gc.shp").toURL();
    // Criamos um DataStore a partir do Shapefile
    ShapefileDataStore data = new ShapefileDataStore(s);
```

Geotools: Informações em um *shapefile*

```
// O nome do DataStore é o próprio nome do Shapefile (só tem um)
String nome = data.getTypeNames()[0];
// Recuperamos o schema para estes dados.
FeatureType ft = data.getSchema();
// Imprimimos pares atributo/tipo (classe)
for (int i=0; i<ft.getAttributeCount(); i++)
  {
  System.out.print(String.format("%3d: ",new Object[]{i}));
  AttributeType at = ft.getAttributeType(i);
  if (!Geometry.class.isAssignableFrom(at.getType()))
    {
    System.out.print("Feature ");
    System.out.print(at.getName()+"\t");
    System.out.print(at.getType().getName());
    }
  else
    {
    System.out.print("Geometry ");
    }
  System.out.println();
  }
```

Geotools: Informações em um *shapefile*

```
    // Recuperamos a referência para tabelas e shapefiles.
    FeatureSource source = data.getFeatureSource(nome);
    // Recuperamos todas as features.
    FeatureCollection features = source.getFeatures();
    System.out.println("Temos "+features.size()+" features.");
    Iterator iterator = features.iterator(); // Sem generics :-(
    while(iterator.hasNext())
      {
      Feature feature = (Feature) iterator.next();
      // O ID daquela feature.
      System.out.print(feature.getID() + "\t");
      // Imprimimos atributos não-geométricos.
      for (int attr=0;attr<feature.getNumberOfAttributes();attr++)
        {
        Object attribute = feature.getAttribute(attr);
        if (!(attribute instanceof com.vividsolutions.jts.geom.Geometry))
          {
          if (attribute instanceof String) System.out.print("'"+attribute+"' ");
          else System.out.print(attribute+" ");
          }
        }
      System.out.println();
      }
    }
  }
```

Geotools: Informações em um *shapefile*

- Saída:

```
Jan 14, 2007 12:21:24 AM org.geotools.data.shapefile.ShapefileDataStore openPrjReader
WARNING: projection (.prj) for shapefile:
file:/home/rafael/Java/Geo/kernelestimator/WebContent/Shapefiles/Brasil/55mu2500gc.shp is not available
Jan 14, 2007 12:21:24 AM org.geotools.data.shapefile.ShapefileDataStore openPrjReader
WARNING: projection (.prj) for shapefile:
file:/home/rafael/Java/Geo/kernelestimator/WebContent/Shapefiles/Brasil/55mu2500gc.shp is not available
  0: Feature the_geom    com.vividsolutions.jts.geom.MultiPolygon
  1: Feature GEOCODIGO   java.lang.Long
  2: Feature NOME   java.lang.String
  3: Feature UF     java.lang.String
  4: Feature ID_UF  java.lang.Long
  5: Feature REGIAO java.lang.String
  6: Feature MESOREGIAO  java.lang.String
  7: Feature MICROREGIA  java.lang.String
  8: Feature LATITUDE    java.lang.String
  9: Feature LONGITUDE   java.lang.String
 10: Feature SEDE   java.lang.Boolean
Temos 5807 features.
55mu2500gc.1   1200336 'Mâncio Lima' 'AC' 12 'Norte' 'VALE DO JURUA' 'CRUZEIRO DO SUL' '-7.614' '-
72.896' true
55mu2500gc.2   1300201 'Atalaia do Norte' 'AM' 13 'Norte' 'SUDOESTE AMAZONENSE' 'ALTO SOLIMOES' '-4.372'
'-70.192' true
....
55mu2500gc.58053205309 'Vitória' 'ES' 32 'Sudeste' 'CENTRAL ESPIRITO-SANTENSE' 'VITORIA' '-20.319' '-
40.338' false
55mu2500gc.58065003207 'Corumbá' 'MS' 50 'Centro-Oeste' 'PANTANAL SUL MATO-GROSSENSE' 'BAIXO PANTANAL'
'-18.507' '-54.760' true
55mu2500gc.58075003306 'Coxim' 'MS' 50 'Centro-Oeste' 'SUDOESTE DE MATO GROSSO DO SUL' 'ALTO TAQUARI' '-
19.009' '-57.653' true
```

Geotools: Visualizando *shapefiles*

- Primeiro criamos um componente para visualização.

```
package geotools;

import java.awt.*;
import java.awt.Dimension;
import java.awt.geom.AffineTransform;
import java.io.IOException;
import javax.swing.JComponent;
import org.geotools.data.shapefile.ShapefileDataStore;
import org.geotools.map.*;
import org.geotools.referencing.crs.DefaultGeographicCRS;
import org.geotools.renderer.shape.ShapefileRenderer;
import org.geotools.styling.*;
import com.vividsolutions.jts.geom.*;

public class ShapefileComponent extends JComponent
  {
  private ShapefileDataStore sds;  // Datastore para o shapefile.
  private ShapefileRenderer renderer;   // Renderer para o shapefile.
  private Envelope envelope;   // Envelope em coordenadas do shapefile.
  private final Coordinate center;  // Coordenada do centro.
  private double zoom;  // Zoom para a escala.
  // Dimensões para o componente.
  private final int width = 640;
  private final int height = 640;
```

Geotools: Visualizando *shapefiles*

```
// Construtor, cria instâncias internas.
public ShapefileComponent(ShapefileDataStore sds) throws IOException
  {
  this.sds = sds;
  // Criamos um Style
  StyleBuilder sb = new StyleBuilder();
  // Criamos um tipo de linha
  LineSymbolizer lineSymb =
    (LineSymbolizer)sb.createLineSymbolizer(Color.BLUE, 1);
  // e um estilo a partir desta linha.
  Style style = sb.createStyle(lineSymb);
  // Criamos um MapContext um sistema de coordenadas e adicionamos um layer.
  MapContext mc = new DefaultMapContext(DefaultGeographicCRS.WGS84);
  mc.addLayer(sds.getFeatureSource(),style);
  // Agora com o contexto criamos um ShapefileRenderer.
  renderer = new ShapefileRenderer(mc);
  // Calculamos o envelope e centro para os dados.
  envelope = mc.getLayerBounds();
  center = new Coordinate((envelope.getMinX()+envelope.getMaxX())/2,
                          (envelope.getMinY()+envelope.getMaxY())/2);
  zoom = 1;
  }
// Tamanhos máximo e mínimo para o componente.
public Dimension getMaximumSize() { return getPreferredSize(); }
public Dimension getMinimumSize() { return getPreferredSize(); }
// Tamanho preferido para o componente.
public Dimension getPreferredSize() { return new Dimension(width,height); }
```

Geotools: Visualizando *shapefiles*

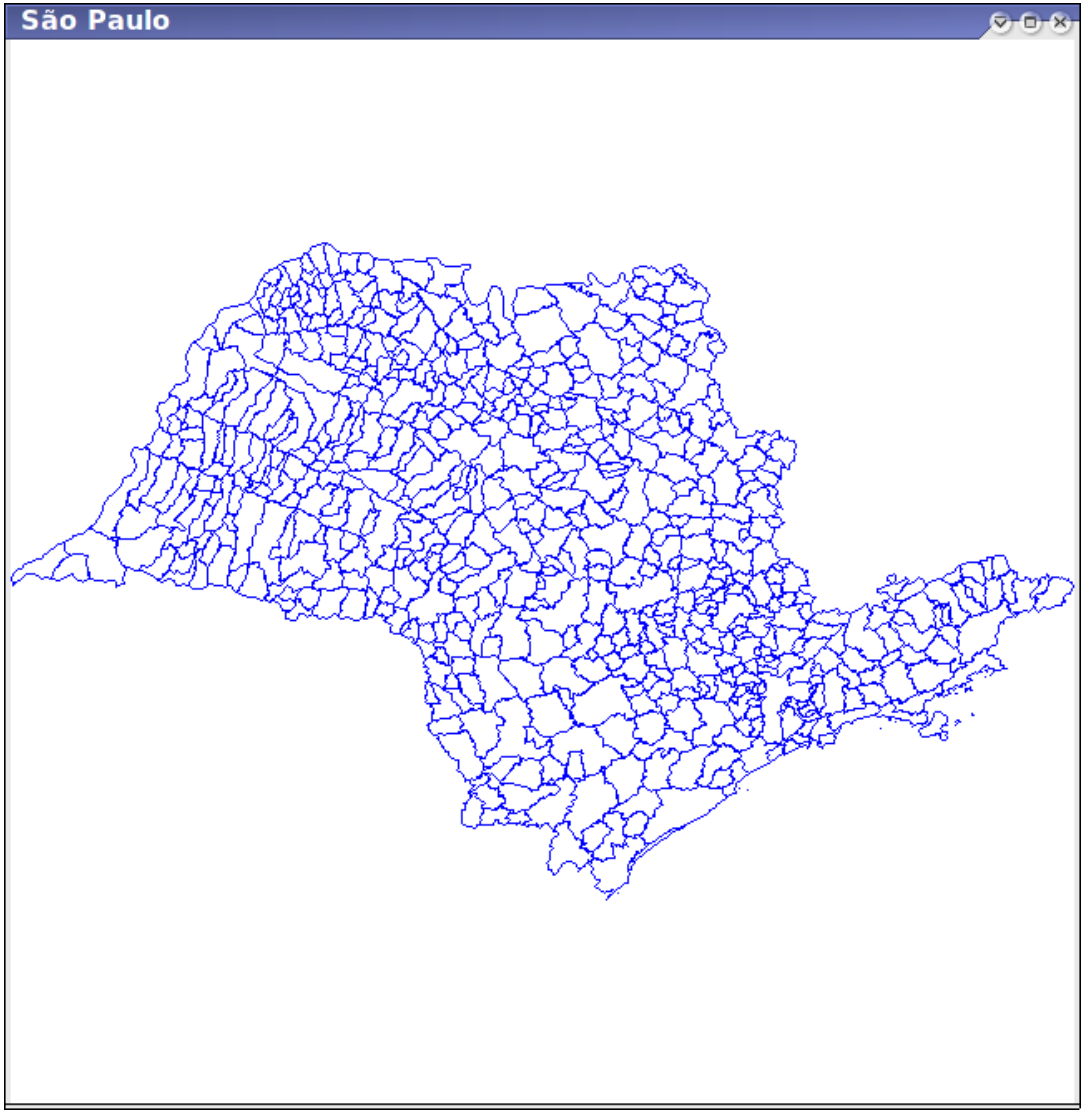
```
// Como o componente deve ser pintado?
protected void paintComponent(Graphics g)
  {
  Graphics2D g2d = (Graphics2D)g;
  // g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
  //                      RenderingHints.VALUE_ANTIALIAS_ON);
  g2d.setColor(Color.WHITE);
  g2d.fillRect(0,0,width,height);
  // Uma AffineTransform que mapeie dados no shapefile com a área para plotagem
  AffineTransform at = new AffineTransform();
  // Calculamos a escala
  double escala = (Math.min(getWidth()/envelope.getWidth(),
                   getHeight()/envelope.getHeight())*zoom);
  // Translação ao centro do componente.
  at.translate (getWidth()/2,getHeight()/2);
  // Escala com y negativo para corrigir orientação.
  at.scale(escala,-escala);
  // Translação ao centro dos dados.
  at.translate(-center.x,-center.y);
  // Pintamos com a transformada.
  renderer.paint(g2d,getBounds(),at);
  }

}
```

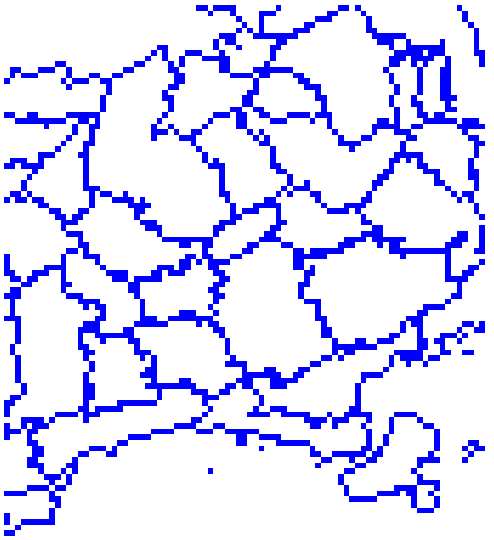
Geotools: Visualizando *shapefiles*

- Usando o componente para visualização.

```
package geotools;

import java.io.*;
import java.net.URL;
import javax.swing.JFrame;
import org.geotools.data.shapefile.ShapefileDataStore;

public class DisplayShapefile extends JFrame
  {
  // Construtor, monta a GUI.
  public DisplayShapefile(URL s) throws IOException
    {
    super("São Paulo");
    ShapefileDataStore data = new ShapefileDataStore(s);
    ShapefileComponent sc = new ShapefileComponent(data);
    getContentPane().add(sc);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);   pack();   setVisible(true);
    }
  // Main, cria a aplicação
  public static void main(String[] args) throws IOException
    {
    URL s = new File("../Dados/E2500/Uf/SP/35mu2500gc.shp").toURL();
    new DisplayShapefile(s);
    }
  }
```

Geotools: Visualizando *shapefiles*





Geotools: Criando novos *shapefiles*

- “Receita de bolo” consideravelmente mais complexa...
- Primeiro, uma classe para encapsular Features.

```
package geotools;

import org.geotools.feature.*;
import com.vividsolutions.jts.geom.Point;

// Esta classe representa um evento que ocorre em uma lat/long e timestamp.
public class EventoLatLong extends DefaultFeature
  {
  public EventoLatLong(Object[] parâmetros,String id)
                throws IllegalAttributeException
    {
    super(criaSchema(), parâmetros, id);
    }
```

Geotools: Criando novos *shapefiles*

```
// Método que cria o schema do "banco de dados"
public static DefaultFeatureType criaSchema()
  {
  GeometryAttributeType latlongAtt =
    (GeometryAttributeType)AttributeTypeFactory.newAttributeType("geometry",
                                                    Point.class);
  AttributeType dataAttr = AttributeTypeFactory.newAttributeType("DATA",String.class);
  AttributeType horaAttr = AttributeTypeFactory.newAttributeType("HORA",String.class);
  FeatureTypeBuilder factory = FeatureTypeBuilder.newInstance("EventoLatLong");
  factory.addType(latlongAtt);
  factory.addType(dataAttr);
  factory.addType(horaAttr);
  DefaultFeatureType ft = null;
  try
    {
    ft = (DefaultFeatureType)factory.getFeatureType();
    }
  catch(SchemaException e)
    {
    e.printStackTrace();
    }
  return ft;
  }
}
```

Geotools: Criando novos *shapefiles*

- Classe que cria e armazena instâncias de EventoLatLong.

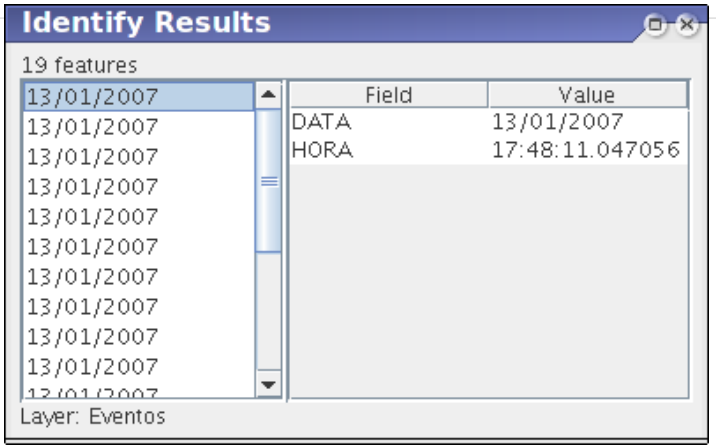
```
package geotools;

import java.io.*;
import java.net.URL;
import java.util.StringTokenizer;
import org.geotools.data.FeatureStore;
import org.geotools.data.shapefile.ShapefileDataStore;
import org.geotools.feature.*;
import com.vividsolutions.jts.geom.*;

public class CreateShapefileFromLatLongPoints
  {
  public static void main(String[] args) throws IOException
    {
    // Coleção de features
    FeatureCollection features = FeatureCollections.newCollection();
    // Abrimos um arquivo em um formato conhecido para leitura.
    BufferedReader br = new BufferedReader(new FileReader("../Dados/20070113.storm"));
    br.readLine(); // Pulamos a primeira linha (cabeçalho)
    int lineNumber = 0;
```

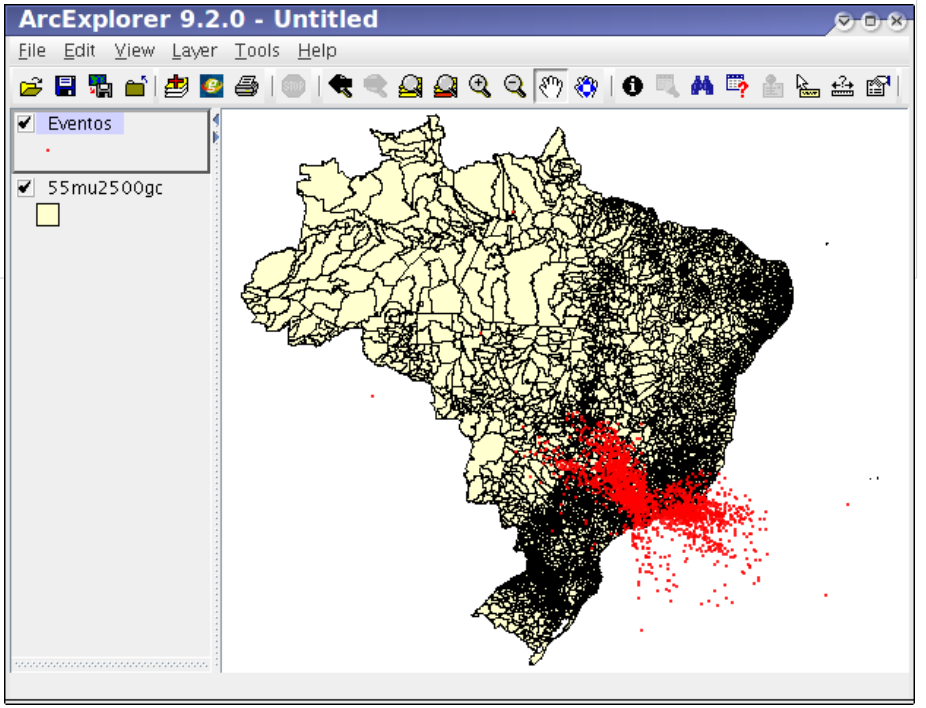
Geotools: Criando novos *shapefiles*

```
    // Enquanto houver linhas...
    while(true)
      {
      String line = br.readLine(); if (line == null) break;
      StringTokenizer st = new StringTokenizer(line);        lineNumber++;
      // Recuperamos os campos.
      String UTCSeconds = st.nextToken();
      double longitude = Double.parseDouble(st.nextToken());
      double latitude = Double.parseDouble(st.nextToken());
      String data = st.nextToken();       String hora = st.nextToken();
      // Criamos uma feature com estes campos.
      Feature f = createFeature(""+lineNumber,latitude,longitude,data,hora);
      features.add(f); // Warning!
      if (lineNumber % 10 == 0) System.out.print(lineNumber+" ");
      if (lineNumber % 100 == 0) System.out.println();
      }
    System.out.println("\n"+features.size());
    br.close();
    // Criamos um ShapefileDataStore
    URL s = new File("/tmp/Eventos.shp").toURL();
    ShapefileDataStore data = new ShapefileDataStore(s);
    // Criamos um schema para este ShapefileDataStore.
    data.createSchema(EventoLatLong.criaSchema());
    // Criamos um FeatureStore para usar addFeatures().
    FeatureStore store = (FeatureStore)data.getFeatureSource();
    store.addFeatures(features);
    System.out.println("OK");
    }
```

Geotools: Criando novos *shapefiles*

```
// Organizamos os dados em um Feature (no caso, EventoLatLong)
private static Feature createFeature(String id,double latitude,double longitude,
                                     String data,String hora)
  {
  GeometryFactory gf = new GeometryFactory();
  Point latlong = gf.createPoint(new Coordinate(latitude,longitude));
  Object[] attributes = {latlong,data,hora};
  Feature f = null;
  try
    {
    f = new EventoLatLong(attributes,id);
    }
  catch (IllegalAttributeException e)
    {
    e.printStackTrace();
    }
  return f;
  }
}
```



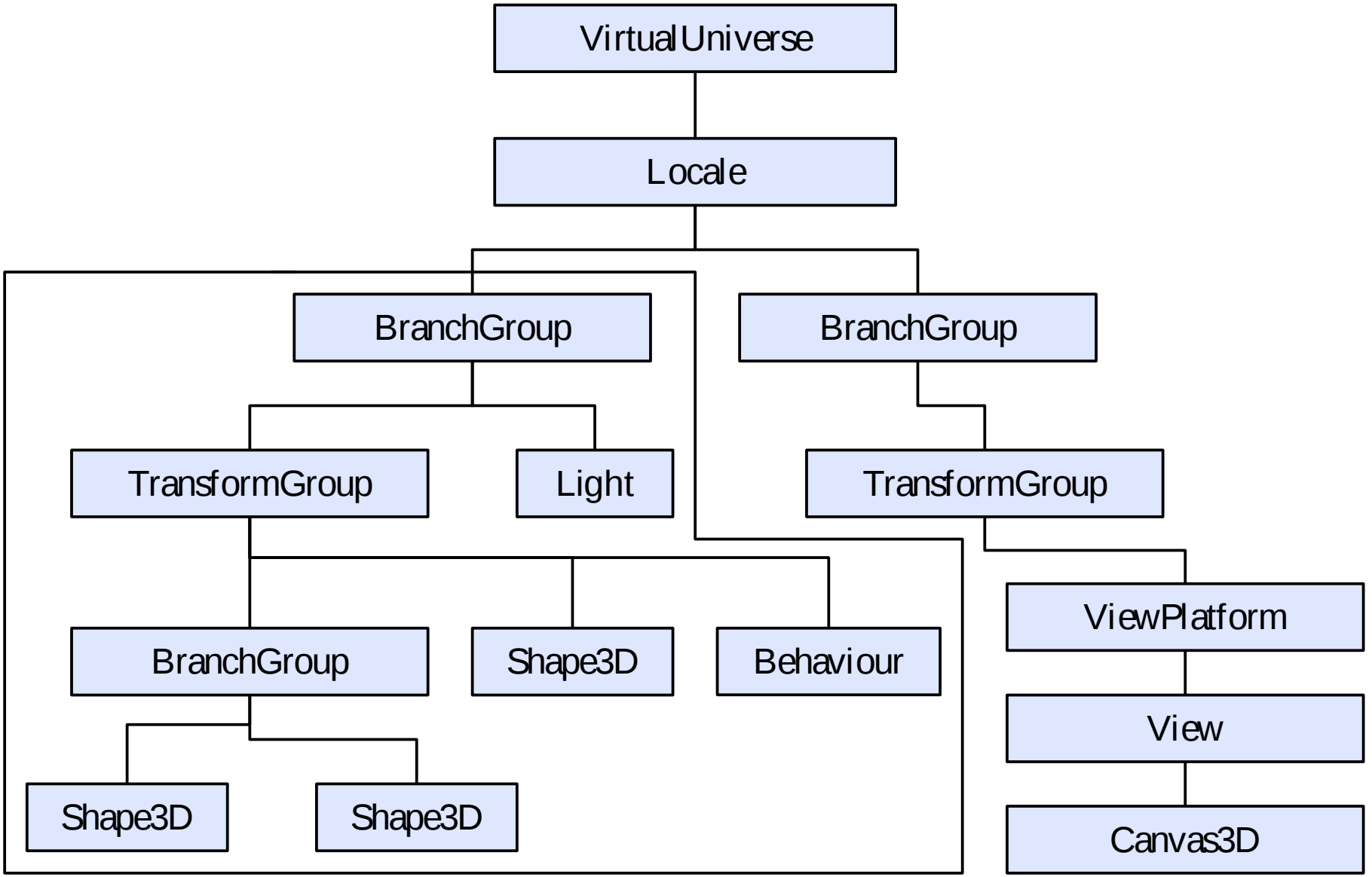


Visualização/3D: **Java3D**.

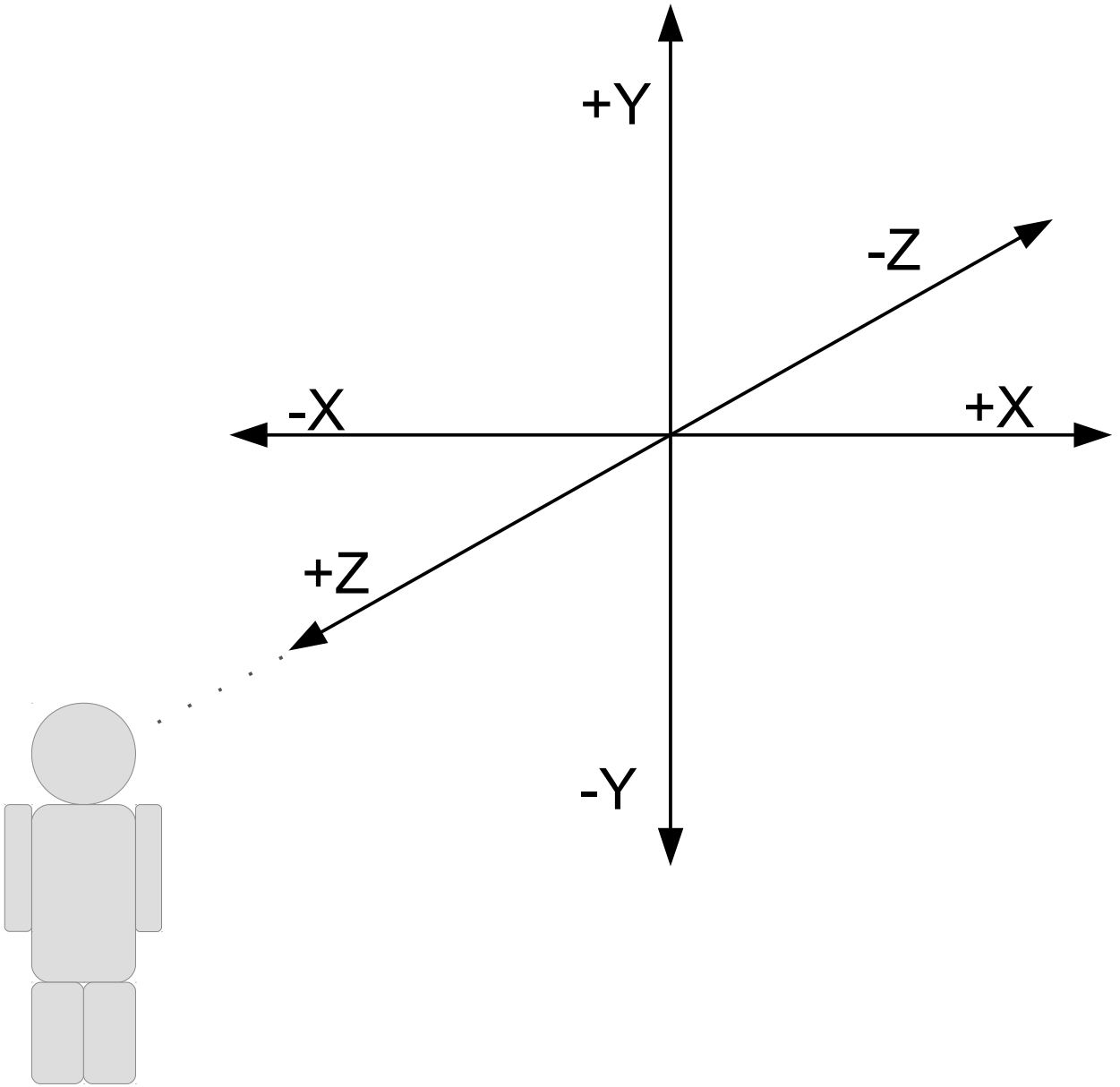
Java3D

- http://java.sun.com/products/java-media/3D/
- Coleção de classes que representam elementos de alto nível para criação, renderização e manipulação de cenas em 3D.
  - Não é ferramenta de criação de conteúdo (ao menos não visualmente!)
  - Não é VRML: descrição de objetos e características é feita com instâncias de classes.
  - Não é OpenGL: abstração de OO é usada.
- Cenas são descritas como grafos acíclicos diretos (*DAGs*).
- Objetos podem ser compostos a partir de outros.

Java3D: Grafos de Cenas



Java3D: Eixos em 3D



Java3D: Aplicações simples

- Como escrever uma aplicação simples que usa Java3D:
  1. Criamos um **Canvas3D** com um **GraphicsConfiguration**.
  2. Criamos um grafo de cena (**BranchGroup** e folhas) e compilamos esta cena.
  3. Criamos um **SimpleUniverse** com o **Canvas3D** e adicionamos o grafo de cena ao **SimpleUniverse**.
  4. Adicionamos o **Canvas3D** à um componente gráfico em uma aplicação ou *applet*.

Java3D: *Hello World*

```
package java3d;

import java.awt.*;
import javax.media.j3d.*;
import javax.swing.JFrame;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;

// Classe que cria e mostra uma cena simples
public class HelloWorld3D extends JFrame
  {
  // Constantes para dimensionamento.
  private static Dimension tamCanvas = new Dimension(500,500);
  private static BoundingSphere abrangência =
                          new BoundingSphere(new Point3d(0.0,0.0,0.0),100.0);

  // Construtor, cria a cena e a GUI.
  public HelloWorld3D()
    {
    super("Hello World 3D!");
    // Criamos um Canvas3D e o adicionamos à GUI
    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
    Canvas3D canvas3D = new Canvas3D(config);
    getContentPane().add(canvas3D);
```

Java3D: *Hello World*

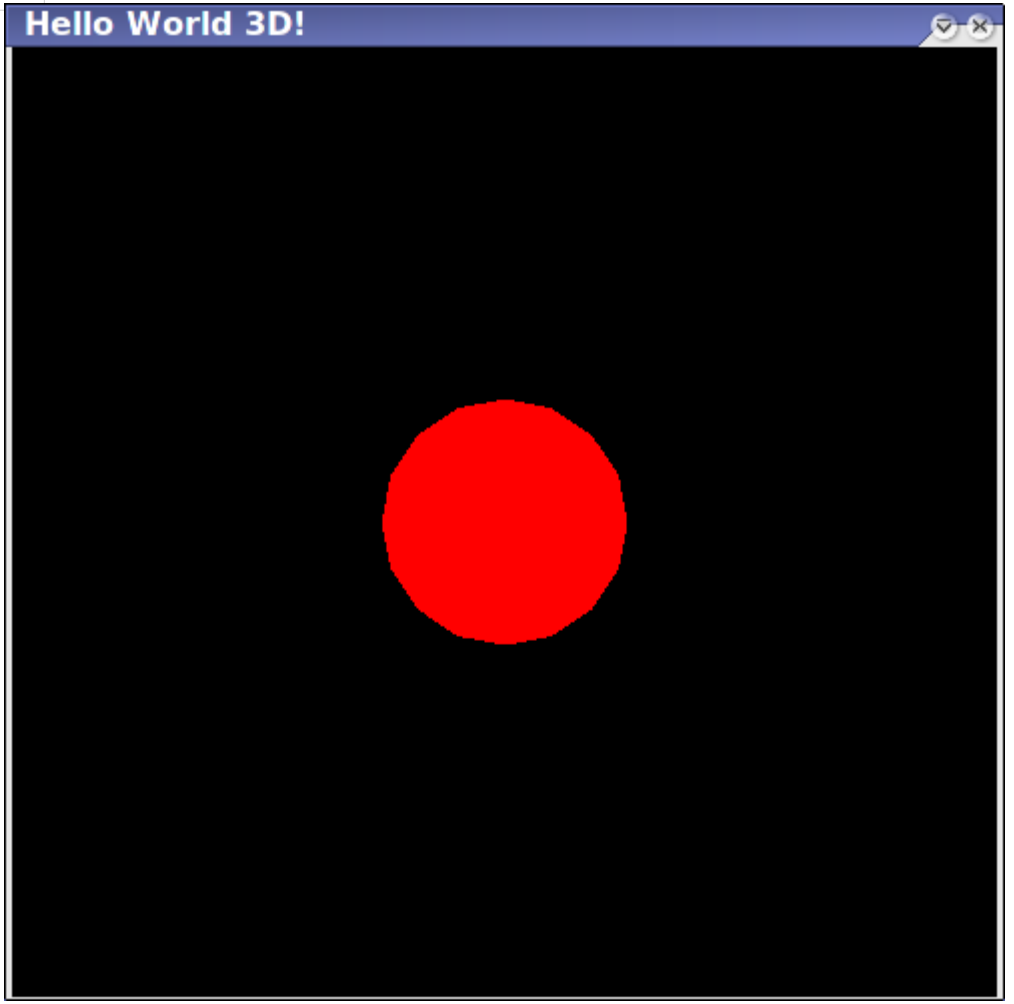
```
    // Criamos a cena
    BranchGroup scene = createSceneGraph();
    scene.compile();
    // Criamos um SimpleUniverse
    SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
    simpleU.getViewingPlatform().setNominalViewingTransform();
    simpleU.addBranchGraph(scene);
    // Opções do JFrame
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(tamCanvas);
    setResizable(false);
    setVisible(true);
    }

  private static BranchGroup createSceneGraph()
    {
    // Criamos o BranchGroup
    BranchGroup bg = new BranchGroup();
    // Adicionamos uma esfera vermelha
    bg.addChild(esferaVermelha());
    // Adicionamos uma luz ambiente
    bg.addChild(luzAmbiente());
    // Retornamos o BranchGroup
    return bg;
    }
```

Java3D: *Hello World*

```
private static Sphere esferaVermelha()
  {
  // Criamos uma aparência
  Appearance aparência = new Appearance();
  Color3f vermelho = new Color3f(Color.RED);
  Color3f preto = new Color3f(Color.BLACK);
  // Cores ambiente, emissiva, difusa e especular.
  aparência.setMaterial(new Material(vermelho,preto,vermelho,preto,80.0f));
  // Criamos e retornamos uma esfera com esta aparência
  Sphere sphere = new Sphere(0.25f,Primitive.GENERATE_NORMALS,aparência);
  return sphere;
  }

private static AmbientLight luzAmbiente()
  {
  // Criamos uma luz ambiente branca
  Color3f white = new Color3f(Color.WHITE);
  AmbientLight luz = new AmbientLight(white);
  // A luz terá uma área de influência
  luz.setInfluencingBounds(abrangência);
  return luz;
  }
```

Java3D: *Hello World*

```
  public static void main(String[] args)
    {
    new HelloWorld3D();
    }
  }
```



Java3D: Cubos Coloridos

- Criaremos vários objetos, precisamos aplicar transformações pois objetos pré-fabricados são centrados em (0,0,0).

```
package java3d;

import java.awt.*;
import javax.media.j3d.*;
import javax.swing.JFrame;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;

// Classe que cria e mostra uma cena mais complexa
public class Cubos3D extends JFrame
  {
  // Constantes para dimensionamento.
  private static Dimension tamCanvas = new Dimension(500,500);
  private static BoundingSphere abrangência =
                              new BoundingSphere(new Point3d(0.0,0.0,0.0),100.0);
```

Java3D: Cubos Coloridos

```
// Construtor, cria a cena e a GUI.
public Cubos3D()
  {
  super("Cubos Coloridos 3D!");
  // Criamos um Canvas3D e o adicionamos à GUI
  GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
  Canvas3D canvas3D = new Canvas3D(config);
  getContentPane().add(canvas3D);
  // Criamos a cena
  BranchGroup scene = createSceneGraph();
  scene.compile();
  // Criamos um SimpleUniverse
  SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
  simpleU.getViewingPlatform().setNominalViewingTransform();
  simpleU.addBranchGraph(scene);
  // Opções do JFrame
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(tamCanvas);
  setResizable(false);
  setVisible(true);
  }
```

Java3D: Cubos Coloridos

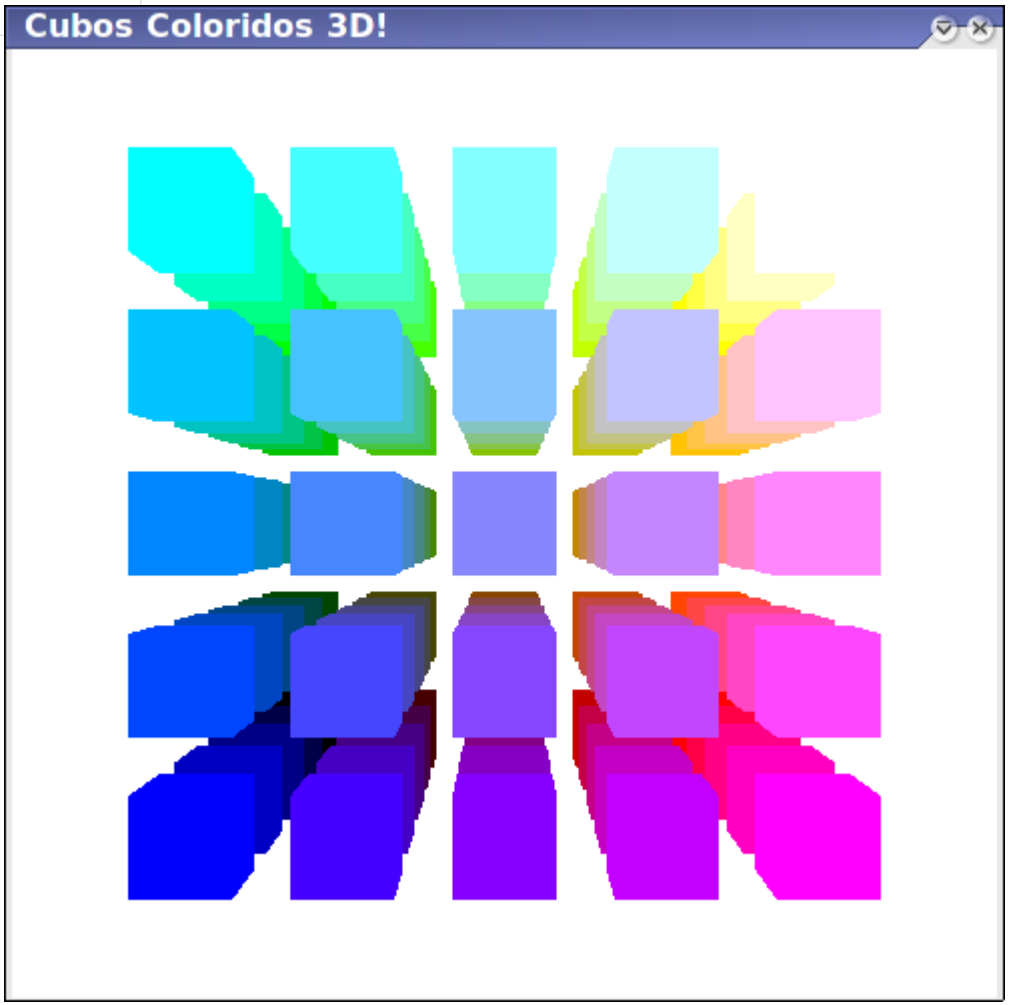
```
private static BranchGroup createSceneGraph()
  {
  // Criamos o BranchGroup
  BranchGroup bg = new BranchGroup();
  // Que tal um background?
  Background fundo = new Background(new Color3f(1.0f,1.0f,1.0f));
  fundo.setApplicationBounds(abrangência);
  bg.addChild(fundo);
  // Adicionamos 125 cubos com cores e coordenadas diferentes
  for(float x=-0.5f;x<0.75f;x+=0.25f)
    for(float y=-0.5f;y<0.75f;y+=0.25f)
      for(float z=-0.5f;z<0.75f;z+=0.25f)
        bg.addChild(criaCubo(x,y,z,new Color3f(0.5f+x,0.5f+y,0.5f+z)));
  // Adicionamos uma luz ambiente
  bg.addChild(luzAmbiente());
  // Retornamos o BranchGroup
  return bg;
  }
```

Java3D: Cubos Coloridos

```
private static BranchGroup criaCubo(float x,float y,float z,Color3f cor)
  {
  // Criamos uma Aparência
  Appearance aparência = new Appearance();
  Color3f preto = new Color3f(Color.BLACK);
  // Cores ambiente, emissiva, difusa e especular.
  aparência.setMaterial(new Material(cor,preto,cor,preto,80.0f));
  // Criamos e retornamos um cubo com esta aparência e transformada
  BranchGroup node = new BranchGroup();
  Transform3D t3d = new Transform3D(); t3d.set(new Vector3f(x,y,z));
  TransformGroup tg = new TransformGroup(t3d);
  tg.addChild(new Box(0.08f,0.08f,0.08f,aparência));
  node.addChild(tg);
  return node;
  }

private static AmbientLight luzAmbiente()
  {
  // Criamos uma luz ambiente branca
  Color3f white = new Color3f(Color.WHITE);
  AmbientLight luz = new AmbientLight(white);
  // A luz terá uma área de influência
  luz.setInfluencingBounds(abrangência);
  return luz;
  }
```

Java3D: Cubos Coloridos

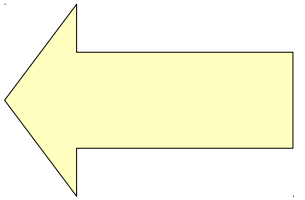
```
  public static void main(String[] args)
    {
    new Cubos3D();
    }
  }
```



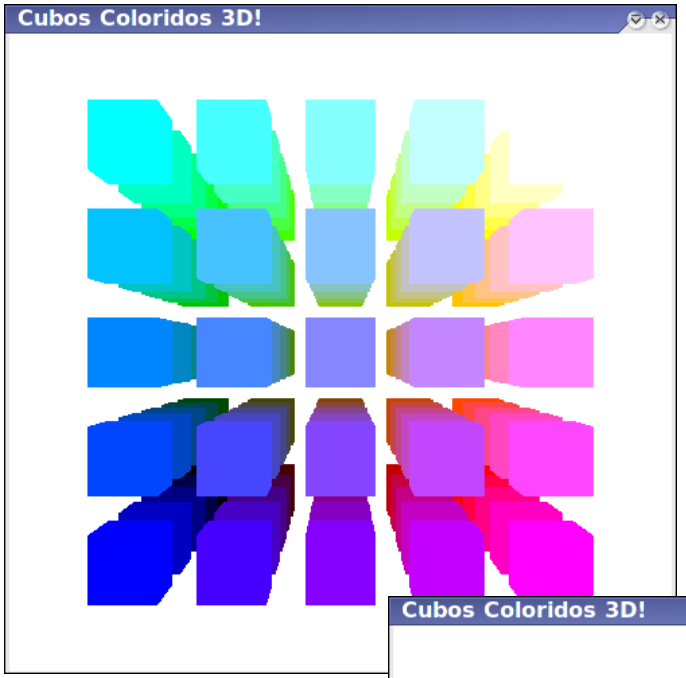
Java3D: Cubos Coloridos Interativos

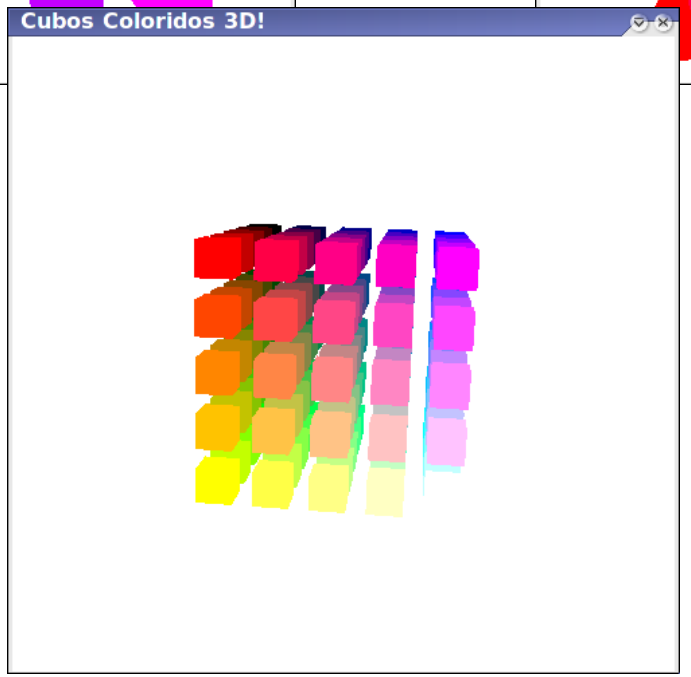
- Como adicionar interação via mouse?
  - Basta criar um *Behavior*.

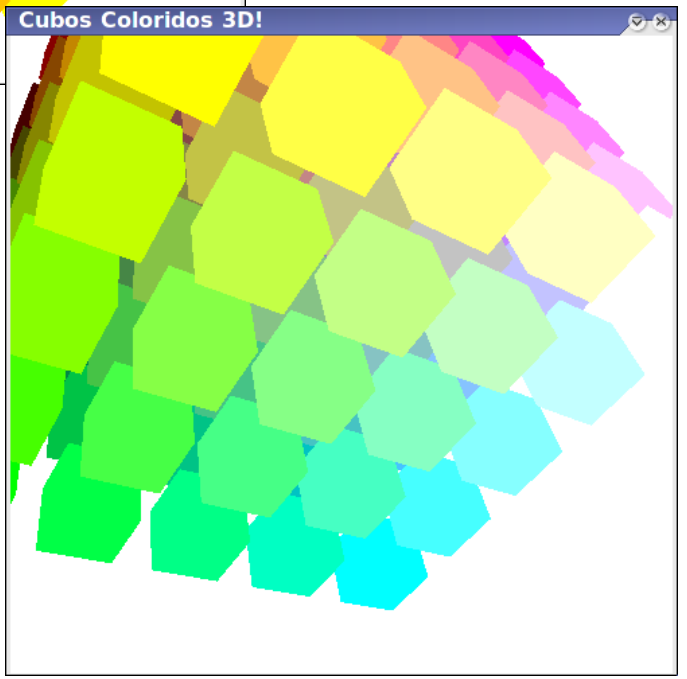
```
// Construtor, cria a cena e a GUI.
public CubosInterativos3D()
  {
  super("Cubos Coloridos 3D!");
  // Criamos um Canvas3D e o adicionamos Ã  GUI
  GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
  Canvas3D canvas3D = new Canvas3D(config);     getContentPane().add(canvas3D);
  // Criamos um SimpleUniverse
  SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
  simpleU.getViewingPlatform().setNominalViewingTransform();
  // Criamos a cena
  BranchGroup scene = createSceneGraph(); scene.compile();
  simpleU.addBranchGraph(scene);
  // Criamos um comportamento interativo.
  OrbitBehavior comportamento = new OrbitBehavior(canvas3D);
  comportamento.setSchedulingBounds(abrangência);
  ViewingPlatform plataforma = simpleU.getViewingPlatform();
  plataforma.setViewPlatformBehavior(comportamento);
  // Opções do JFrame
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(tamCanvas);  setResizable(false);  setVisible(true);
  }
```



Java3D: Cubos Coloridos Interativos







Java3D: Exemplo de Aplicação

- Como os pixels de uma imagem RGB se comportam no espaço de atributos?
  - Abrir uma imagem, obter os valores de seus pixels e criar pequenos pontos (esferas) nas coordenadas RGB.
  - Usaremos uma classe com métodos estáticos para criar objetos (eixos).

```
package java3d;

import java.awt.Color;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;

// Classe com métodos estáticos para criar eixos.
public class Eixo3D
  {
```

Java3D: Exemplo de Aplicação

```
// Cria um eixo Y, composto de um cilindro e um cone.
public static BranchGroup createYAxis(float min,float max,float thickness,Color c)
  {
  BranchGroup bg = new BranchGroup();
  // Criamos uma aparência para este cilindro.
  Appearance aparência = new Appearance();
  Color3f cor = new Color3f(c);       Color3f preto = new Color3f(Color.BLACK);
  aparência.setMaterial(new Material(cor,preto,cor,preto,80.0f));
  // Criamos um cilindro com o tamanho adequado.
  Cylinder cil = new Cylinder(thickness,(max-min),aparência);
  // Transformamos as coordenadas do cilindro.
  Transform3D t3d = new Transform3D();
  t3d.setTranslation(new Vector3f(0,(max-min)/2f,0));
  TransformGroup tgCil = new TransformGroup(t3d);                tgCil.addChild(cil);
  // Criamos também um cone na ponta com dimensões em função do cilindro.
  Color3f corPonta = new Color3f(c.darker());
  Appearance aparênciaPonta = new Appearance();
  aparênciaPonta.setMaterial(new Material(corPonta,preto,corPonta,preto,80.0f));
  Cone cone = new Cone(thickness*2,thickness*8,aparênciaPonta);
  // Reposicionamos o cone.
  t3d = new Transform3D();
  t3d.setTranslation(new Vector3f(0,(max-min)+thickness*4,0));
  TransformGroup tgCone = new TransformGroup(t3d);               tgCone.addChild(cone);

  // Agora adicionamos o conjunto.
  bg.addChild(tgCil);      bg.addChild(tgCone);
  return bg;
  }
```

Java3D: Exemplo de Aplicação

```
// Cria um eixo X, que é um eixo Y girado sobre o eixo Z.
public static BranchGroup createXAxis(float min,float max,float thickness,Color c)
  {
  BranchGroup bg = new BranchGroup();
  // Criamos um eixo Y, que é mais simples (sem rotações).
  BranchGroup eixo = createYAxis(min, max, thickness, c);
  // Rodamos o eixo como um todo.
  Transform3D t3d = new Transform3D();                t3d.rotZ(Math.toRadians(-90));
  TransformGroup tgEixo = new TransformGroup(t3d);
  tgEixo.addChild(eixo);
  bg.addChild(tgEixo);
  return bg;
  }
// Cria um eixo Z, que é um eixo Y girado sobre o eixo X.
public static BranchGroup createZAxis(float min,float max,float thickness,Color c)
  {
  BranchGroup bg = new BranchGroup();
  // Criamos um eixo Y, que é mais simples (sem rotações).
  BranchGroup eixo = createYAxis(min, max, thickness, c);
  // Rodamos o eixo como um todo.
  Transform3D t3d = new Transform3D();                t3d.rotX(Math.toRadians(90));
  TransformGroup tgEixo = new TransformGroup(t3d);
  tgEixo.addChild(eixo);
  bg.addChild(tgEixo);
  return bg;
  }
}
```

Java3D: Exemplo de Aplicação

```
package java3d;

import java.awt.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import javax.media.j3d.*;
import javax.swing.*;
import javax.vecmath.*;
import com.sun.j3d.utils.behaviors.vp.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;

// Classe que cria e mostra pixels no espaço de atributos RGB.
// Veja que todas as coordenadas são normalizadas para [0..1] por causa
// do frustum.
public class Plot3DPixels extends JFrame
  {
  // Constantes para dimensionamento.
  private static Dimension tamCanvas = new Dimension(1000,1000);
  private static BoundingBox abrangência = new BoundingBox(new Point3d(-1,-1,-1),
                                                           new Point3d( 1, 1, 1));
```

Java3D: Exemplo de Aplicação

```
// Construtor, cria a cena e a GUI.
public Plot3DPixels(BufferedImage image)
  {
  super("Pixels em 3D!");
  // Criamos um Canvas3D e o adicionamos à GUI
  GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
  Canvas3D canvas3D = new Canvas3D(config);        getContentPane().add(canvas3D);
  // Criamos um SimpleUniverse
  SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
  simpleU.getViewingPlatform().setNominalViewingTransform();
  // Criamos a cena
  BranchGroup scene = createSceneGraph(image);     scene.compile();
  simpleU.addBranchGraph(scene);
  // Criamos um comportamento interativo.
  OrbitBehavior comportamento = new OrbitBehavior(canvas3D);
  comportamento.setSchedulingBounds(abrangência);
  ViewingPlatform plataforma = simpleU.getViewingPlatform();
  plataforma.setViewPlatformBehavior(comportamento);
  // Posicionamos o observador na ponta do eixo "cinza".
  TransformGroup po = plataforma.getViewPlatformTransform();
  Transform3D t3d = new Transform3D();          po.getTransform(t3d);
  t3d.lookAt(new Point3d(1.4f,2.4f,1.4f),new Point3d(0,0,0),new Vector3d(0,1,0));
  t3d.invert();
  po.setTransform(t3d);
  // Opções do JFrame
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(tamCanvas);           setResizable(false);          setVisible(true);
  }
```

Java3D: Exemplo de Aplicação

```
private static BranchGroup createSceneGraph(BufferedImage image)
  {
  // Criamos o BranchGroup
  BranchGroup bg = new BranchGroup();
  // Que tal um background?
  Background fundo = new Background(new Color3f(1.0f,1.0f,1.0f));
  fundo.setApplicationBounds(abrangência);
  bg.addChild(fundo);
  // Adicionamos eixos para valores de pixels.
  bg.addChild(Eixo3D.createXAxis(0f,1f,0.01f,Color.RED));
  bg.addChild(Eixo3D.createYAxis(0f,1f,0.01f,Color.GREEN));
  bg.addChild(Eixo3D.createZAxis(0f,1f,0.01f,Color.BLUE));
  // Adicionamos uma luz ambiente
  bg.addChild(luzAmbiente());
  // Finalmente adicionamos uma pequena esfera para cada pixel na imagem.
  Color3f preto = new Color3f(Color.BLACK);
  int largura = image.getWidth();
  int altura = image.getHeight();
  int numPixels = 0;
```

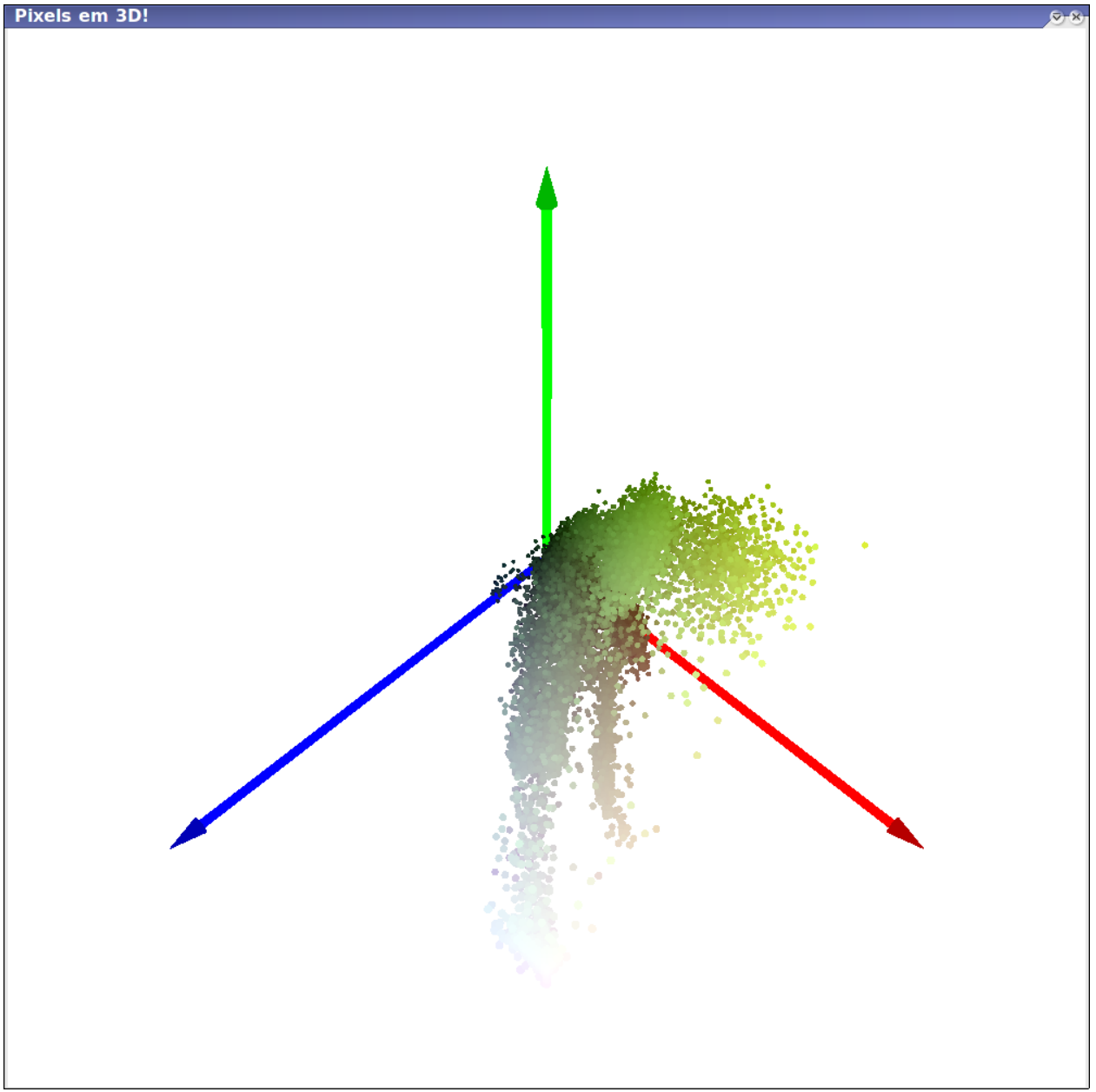
Java3D: Exemplo de Aplicação

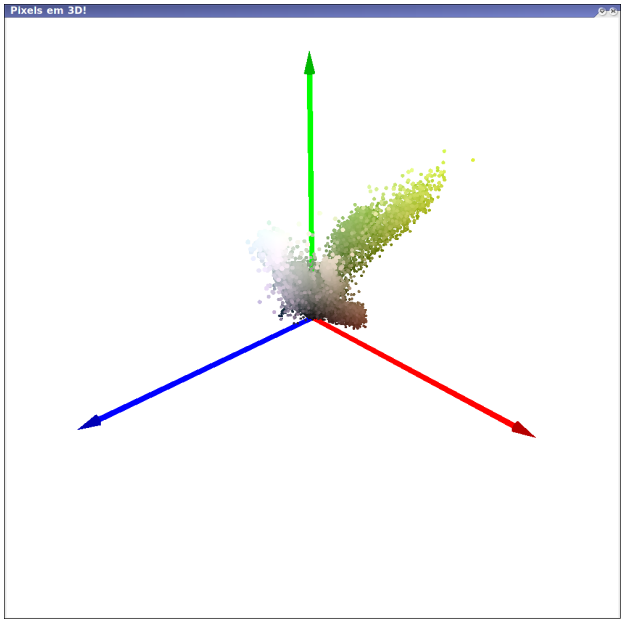
```
    for(int a=0;a<altura;a+=5)
      for(int l=0;l<largura;l+=5)
        {
        int pixel = image.getRGB(l,a);
        int R = (pixel&0x00FF0000)>>16;
        int G = (pixel&0x0000FF00)>>8;
        int B = (pixel&0x000000FF);
        // Com os valores RGB criaremos pequenas esferas.
        Color3f corEsfera = new Color3f(new Color(R,G,B));
        Appearance aparência = new Appearance();
        // Cores ambiente, emissiva, difusa e especular.
        aparência.setMaterial(new Material(corEsfera,preto,corEsfera,preto,0.0f));
        Sphere sp = new Sphere(0.005f,Primitive.GENERATE_NORMALS,aparência);
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(R/255f,G/255f,B/255f));
        TransformGroup tg = new TransformGroup(t3d);
        tg.addChild(sp);
        bg.addChild(tg);
        }
    // Retornamos o BranchGroup
    return bg;
    }
```

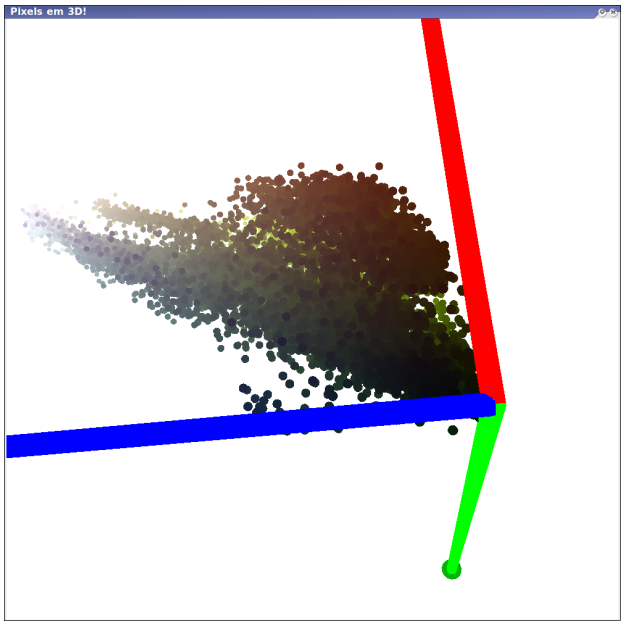
Java3D: Exemplo de Aplicação

```
private static AmbientLight luzAmbiente()
  {
  // Criamos uma luz ambiente branca
  Color3f white = new Color3f(Color.WHITE);
  AmbientLight luz = new AmbientLight(white);
  // A luz terá uma área de influência
  luz.setInfluencingBounds(abrangência);
  return luz;
  }

public static void main(String[] args) throws IOException
  {
  BufferedImage image =
    ImageIO.read(new File("/home/rafael/Docs/ELAC/dsc06447.jpg"));
  new Plot3DPixels(image);
  }
}
```

Java3D: Exemplo de Aplicação



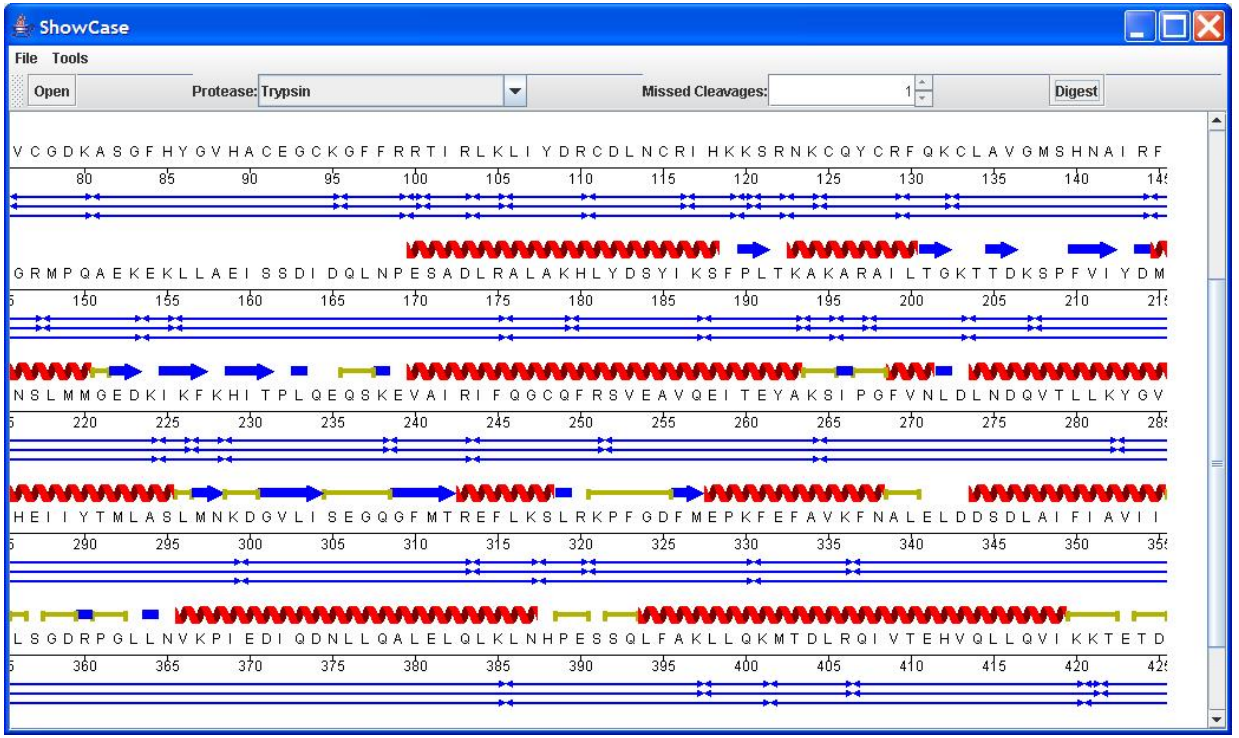




Comentários sobre outras *APIs*

BioJava

- Projeto de software livre para permitir desenvolvimento de aplicações para bioinformática usando Java.
  - "Parente" de BioPython, BioPerl, BioSQL.
  - http://biojava.org/wiki/Main_Page
  - Extensão: BioJavaX (http://biojava.org/wiki/BioJava:BioJavaXDocs).



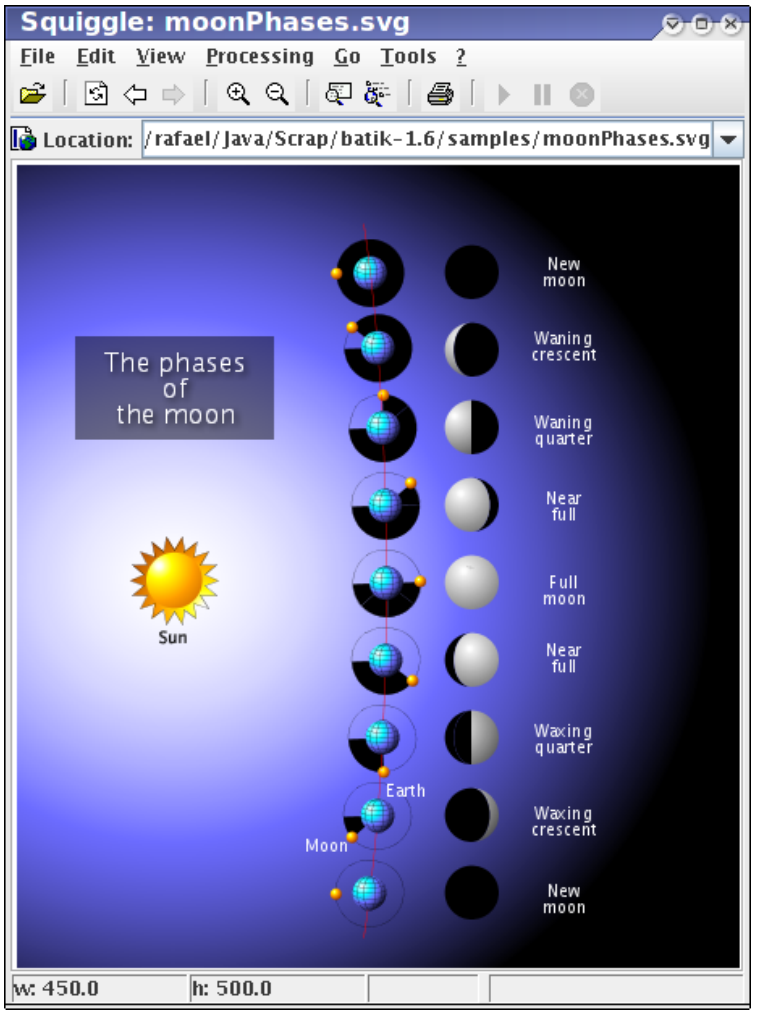
JGAP e JSimul

- Algoritmos Genéticos / Programação Genética usando Java.
- Muita flexibilidade com alguma complexidade.
- http://jgap.sourceforge.net/
- Vários documentos, tutoriais, etc. no sítio.

- Projeto relacionado: JSimul, *JAVA-based simulated annealing package.*
  - http://www.theblueplanet.org/JSimul_readme.html

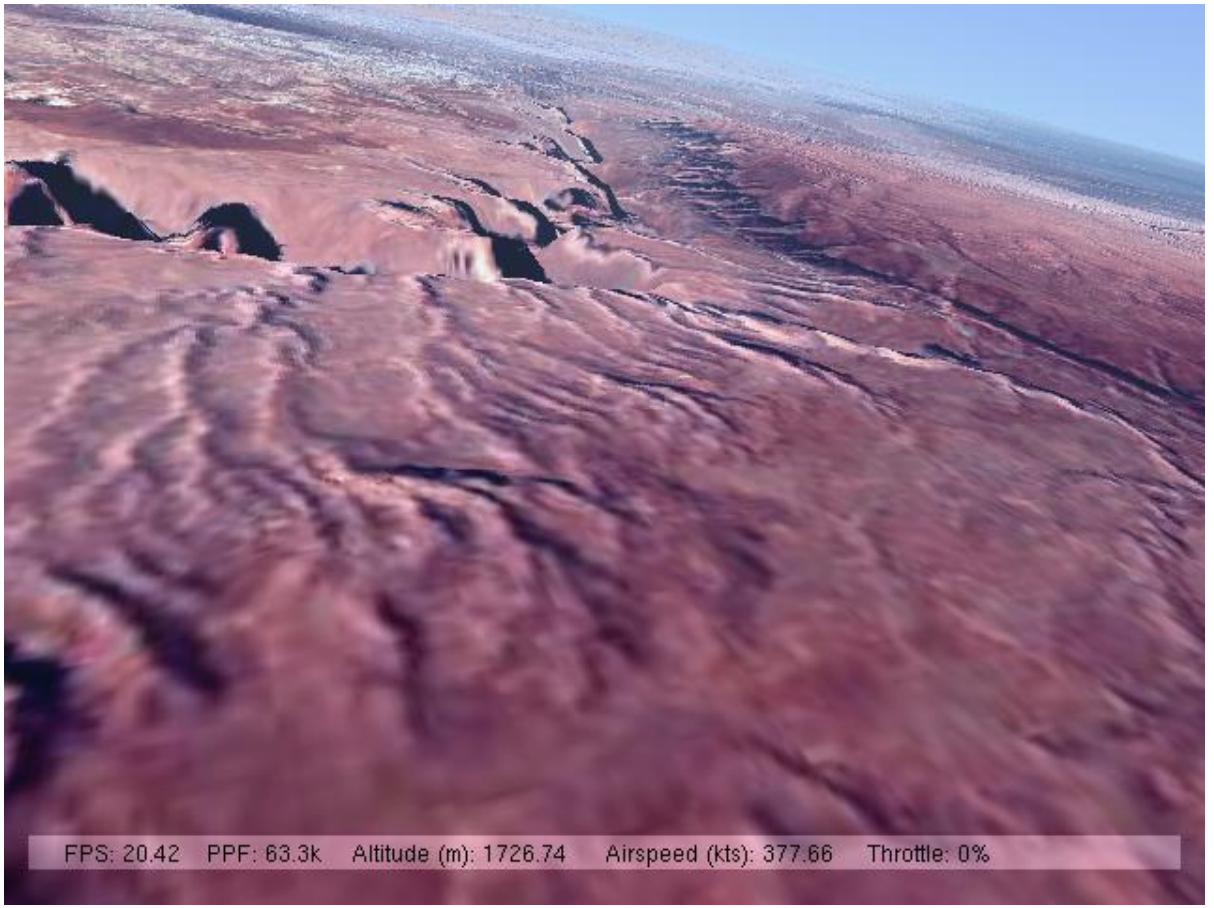
Batik

- Toolkit para manipulação do formato gráfico *Scalable Vector Graphics (SVG)*, inclusive geração e visualização.
- Projeto maduro, http://xmlgraphics.apache.org/batik/
- Vários documentos, exemplos, etc. no sítio.



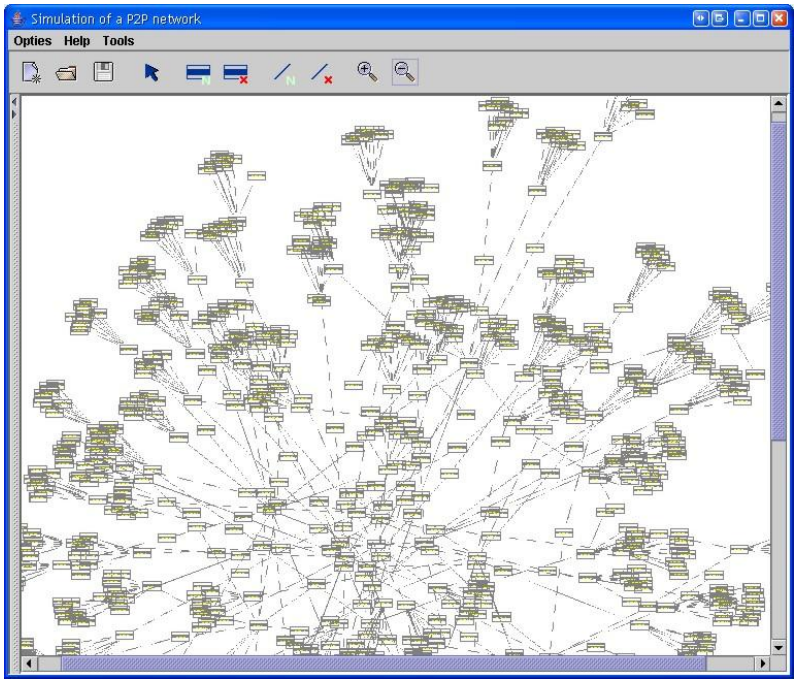
jogl

- *Bindings* de OpenGL para Java.
- Concorrente do Java3D? **Não**.
- https://jogl.dev.java.net/

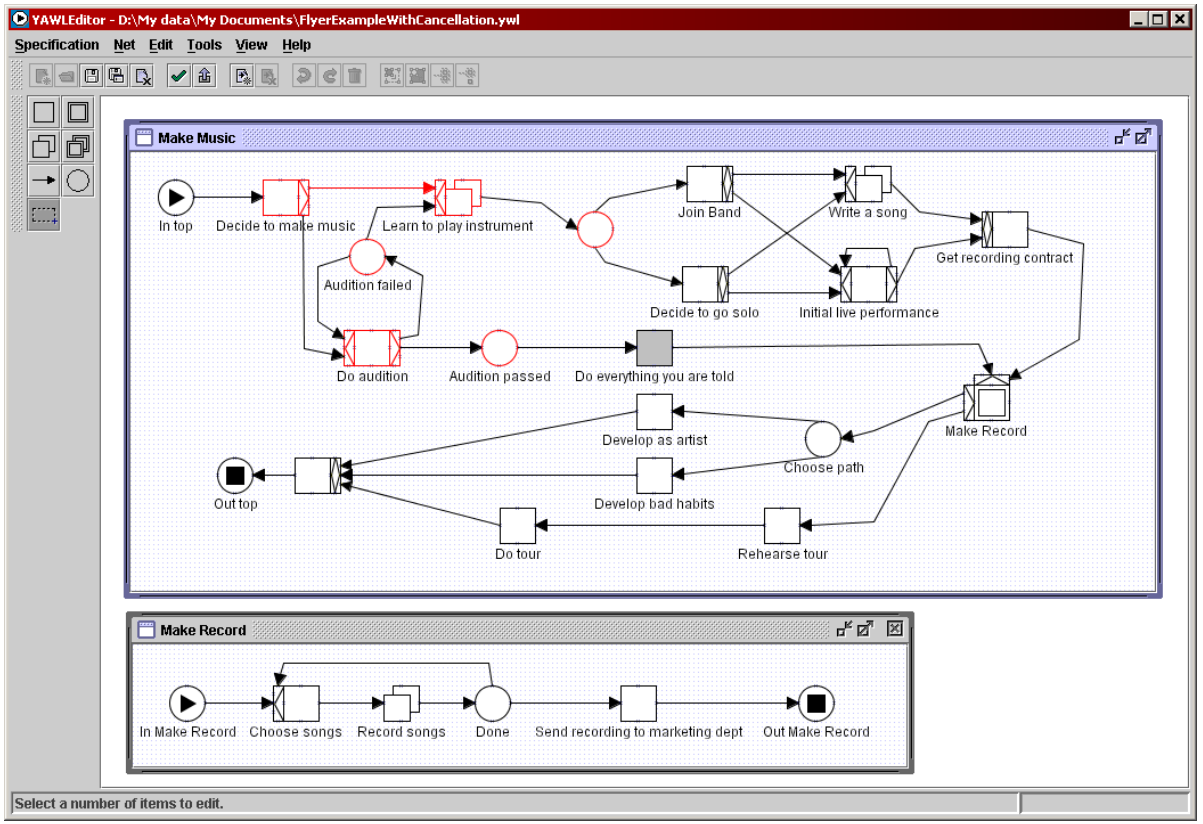


JGraph

- *Toolkit* para visualização e edição de grafos em Java.
- Parte *open source*, parte comercial.
- http://www.jgraph.com/



Visualização de uma rede P2P, http://web.ulyssis.org/~trappie/andy/main.php



YAWL Editor, editor de *workflow*, http://www.citi.qut.edu.au/yawl/index.jsp

Colt e outros

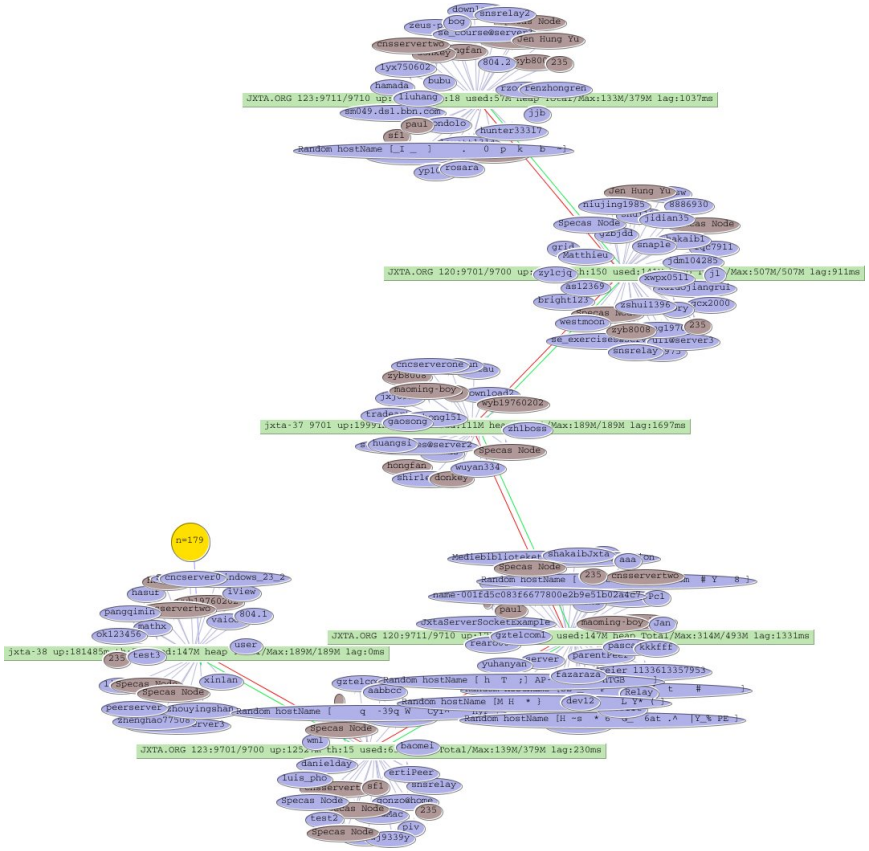
- Conjunto de bibliotecas *open source* para computação científica e técnica de alta performance.
- http://dsd.lbl.gov/~hoschek/colt/

- *International Workshop on Java and Components for Parallelism, Distribution and Concurrency*
  - Parte do *IEEE International Parallel & Distributed Processing Symposium*
  - http://www.labri.fr/perso/chaumett/conferences/iwjpdc/iwjpdc.html
  - Anual desde 1999.

JXTA e outros

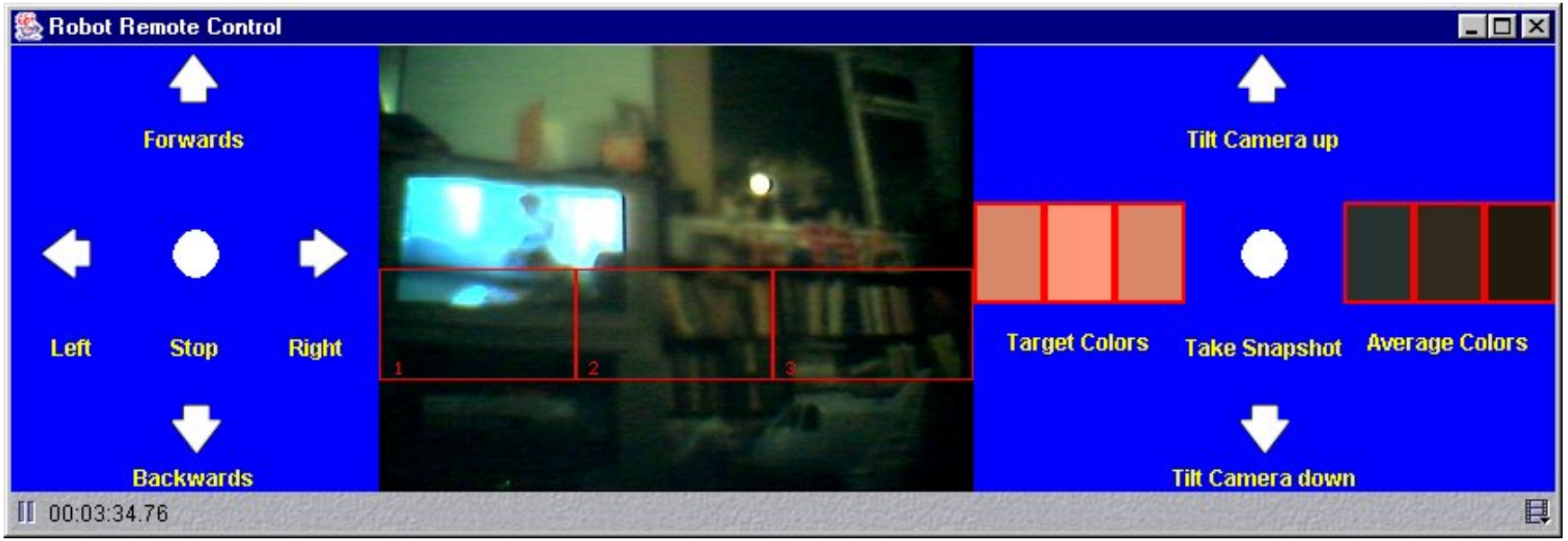
- Conjunto de protocolos abertos para comunicação/colaboração P2P de dispositivos em redes.
- http://www.jxta.org/

- Firefish:
  - Serviço P2P para acesso a dados dinâmicos.
  - http://dsd.lbl.gov/firefish/index.html
- Scishare
  - P2P escaláve, seguro
  - http://dsd.lbl.gov/P2P/file-share/



leJOS

- Máquina Virtual para “tijolos” RCX usados no *Lego Mindstorms*.
- http://lejos.sourceforge.net/
- Contém subprojetos como IDEs e um subsistema de visão!



JMF

- *Java Media Framework API*, possibilita uso de mídia baseada em tempo (áudio, vídeo) em aplicações em Java.
- http://java.sun.com/products/java-media/jmf/
- Permite reprodução de áudio ou vídeo diretamente ou programaticamente.
- Algumas aplicações de análise de vídeo são integradas com JAI para análise de *frames*.

Agentes

- Várias *APIs* e *frameworks*!
- JADE (Java Agent DEvelopment Framework)
  - http://jade.tilab.com/
  - Compatível com especificações da FIPA (*IEEE Foundation for Intelligent Physical Agents*)
- Aglets: Projeto da IBM para agentes móveis
  - http://www.trl.ibm.com/aglets/
  - *Abandonware*?

Number Crunching

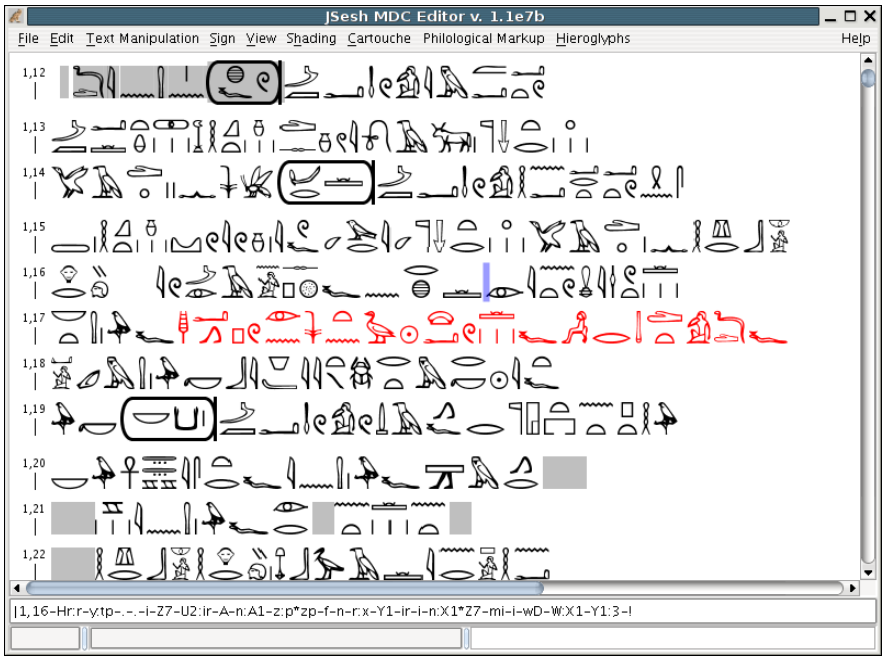
- Projeto Ninja (*Numerically Intensive Java*) da IBM
  - Técnicas de otimização para remoção de *bottlenecks*.
  - Objetivo é fazer Java competitivo com C++ e Fortran.
  - http://www.research.ibm.com/ninja/
- JScience
  - Quer prover uma biblioteca abrangente para a comunidade científica.
  - Módulos para unidades e quantidades, coordenadas, estruturas matemáticas, álgebra linear, cálculos simbólicos, precisão arbitrária e garantida, números racionais, cálculos monetários, etc.
  - http://www.jscience.org/

Informações Técnicas

- **JAI:** Instalação com bibliotecas nativas (*system-wide*) ou com alguns JARs no classpath.
- **Weka:** Um único JAR contém a API e aplicações.
- **JOONE:** Instaladores automáticos ou ZIP, alguns arquivos JAR.
- **GeoTools:** Muitos, muitos arquivos JAR!
  - No Eclipse podemos criar uma *User Library* com todas estes JARs para facilitar desenvolvimento.
- **Java3D**: Instalação depende de bibliotecas nativas, então é *system-wide*.

Referências

- *Wicked Cool Java: Code Bits, Open Source Libraries, and Project Ideas* by Brian D. Eubanks (http://www.wickedcooljava.com/)
- *Technical Java: Applications for Science and Engineering* by Grant Palmer (http://www.phptr.com)
- *Swing Sightings* (http://java.sun.com/products/jfc/tsc/sightings/)



Referências

- *JavaTech, an Introduction to Scientific and Technical Computing with Java*, de Clark S. Lindsey, Johnny S. Tolliver e Thomas Lindblad (http://www.cambridge.org/9780521821131)
- JAI: tutorial em http://www.lac.inpe.br/~rafael.santos/Java/JAI/

Perguntas?

Sobre este mini-curso

Será melhorado, possivelmente ampliado e oferecido em outras ocasiões.

Sua opinião é muito importante!
Preencha o formulário e devolva ao instrutor.