

# NERD

**JFreeChart**


Diogo Carneiro  
Mauricio Ferreira  
Savio Sena



# NERD

## Introdução

- O que é JFreeChart ?
- Usabilidades: Applications, servlets, JSP e applets



## ● O que é JFreeChart ?

É uma biblioteca de classes Java feita para gerar gráficos. Pode gerar vários tipos de gráficos diferentes que serão vistos detalhadamente mais à frente.

## ● Usabilidades: aplicativos, applets, servlets e JSP

*Aplicativos Swing:* Aplicativos comuns em Java.

*Applets:* Miniaplicativo que roda dentro de uma página Web.

*Servlet:* é um programa que estende a funcionalidade de um web server, gerando conteúdo dinâmico e interagindo com os clientes, utilizando o modelo request/response.

*JSP:* é a abreviação de Java Server Pages. É uma tecnologia orientada a criar páginas web com programação em Java.

Veremos alguns exemplos com o JFreeChart mais à frente.

## Vantagens e Funcionalidades



- Licença GNU LGPL (GNU Lesser General Public License).
- Poder construir gráficos facilmente e com poucas linhas de códigos
- Os gráficos podem ser exportados para PNG, JPEG, PDF via iText e SVG via Batik.
- Interatividade com o usuário do aplicativo: Tool Tips, Zoom, Eventos de mouse, anotações
- Rotina de Impressão
- Gerador automático de image map para HTML

# NERD



## Desvantagens

- Não constrói gráficos com domínio em R3
- Esta API não oferece todas as ferramentas necessárias para se fazer gráficos extremamente impressionantes.  
(solução: classe Graphics do J2SE)

# NERD

## Instalação



Arquivo	Descrição
jfreechart-1.0.0-rc1.tar.gz	JFreeChart para Linux/Unix
jfreechart-1.0.0-rc1.zip	JFreeChart para Windows

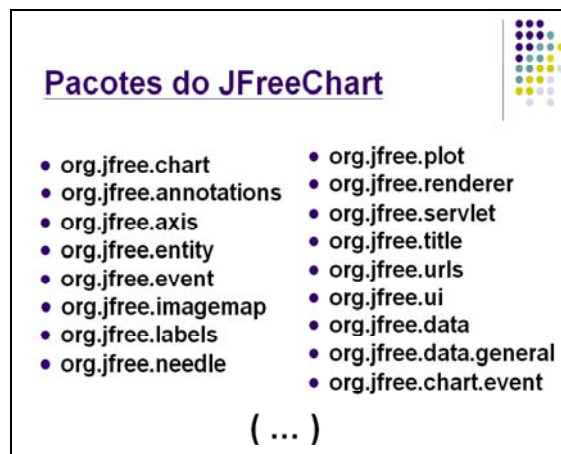
**Site:** <http://www.jfree.org/jfreechart/index.php>

**Javadoc:** <http://www.jfree.org/jfreechart/javadoc>

The two files contain the same source code.

JFreeChart uses the JCommon class library. The JCommon runtime jar file is included in the JFreeChart download, but if you require the source code (recommended) then you should also download JCommon from: <http://www.jfree.org/jcommon/index.html>

# MALE



**chart:** classes que fornecem a habilidade de exibir gráficos em suas aplicações

**annotations:** mecanismo para incluir imagem e textos nos gráficos

**axis:** trata os eixos

**entity:** trata entidades: pacote com url, tooltip, dataset...

**event:** Esse pacote fornece as notificações entre os data objects e os gráficos.

**imagemap:** gerar imagem map de HTML

**labels:** labels e tooltips

**needle:** gráficos com agulha (bússola,...)

**plot:** domínio, imagem, valores, renderer

**renderer:** responsavel por desenhar itens

**servlet:** trata arquivos e gráficos em dir temporários

**title:** trata titulos

**urls:** trata geracao de mapa de imagens para HTML

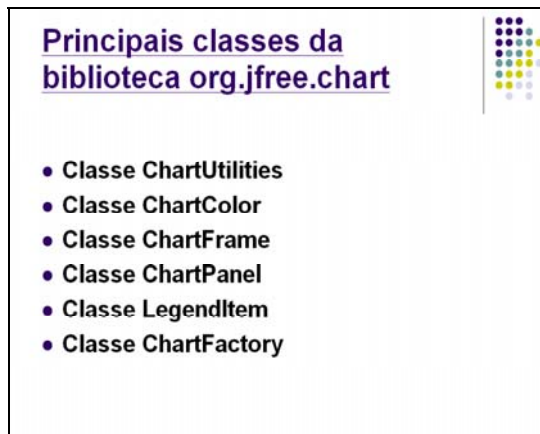
**ui:** classes que podem modificar os gráficos, painel de propriedades (botão dir)

**data:** representam os dados necessários para plotar o gráfico.

**Data.general:** implementações padrões das interfaces que representam os dados necessários para plotar o gráfico.

**Chart.event:** notificações entre os data objects e os gráficos.

# MALE



- **Classe ChartUtilities**

Uma coleção de métodos de conversão para formatos de imagens: PNG e JPEG, e criação de mapas de imagens para HTML.

- **Classe ChartColor**

Classe que estende java.awt.Color aumentando o número de cores disponíveis para os gráficos. Novas cores como: DARK\_CYAN, DARK\_GREEN, DARK\_MAGENTA, DARK\_YELLOW, DARK\_RED, etc.

- **Classe ChartFrame**

Classe que estende javax.swing.JFrame. Esse frame exibe o gráfico.

- **Classe ChartPanel**

Classe que estende javax.swing.JPanel.

- **Classe ChartFactory**

Coleção de métodos que criam a representação dos gráficos. Que veremos mais detalhadamente a seguir.

# MALF

## Etapas para gerar um gráfico



1. Gerar ou carregar os valores
2. Criar a representação
3. Customizar a aparência e o comportamento
4. Definir a apresentação: Aplicação Swing, salvar o gráfico para um arquivo ou exibir dinamicamente em aplicativos Web

O JFreeChart separa a representação dos gráficos individualmente. Mas tem algumas etapas para gerar um gráfico comum a todos os tipos de gráficos:

Vamos ver individualmente cada etapa com mais detalhes...



# MALF

## 1. Gerar / Carregar valores



Implementar a interface `org.jfree.data.Dataset`

- `PieDataset`
- `CategoryDataset`
- `XYDataset`
- `IntervalXYDataset`
- `HighLowDataset`
- `IntervalCategoryDataset`
- `JDBC Datasets`

A primeira parte para gerar um gráfico é gerar ou carregar os valores. Isso significa escolher os dados que serão lidos pelo “dataset”. Todos os dados que estão presentes no JFreeChart devem implementar a interface `org.jfree.data.Dataset`, ou seja, implementar a subinterface do tipo de gráfico a ser desenhado.

Algumas interfaces....

# MALF

## 2. Criar representação



- Classe ChartFactory
- Ex.: `JFreeChart chart ChartFactory.createPieChart ( "Chart" , dataset, true, true, false );`
- Dataset é passado por referência para a fábrica.

Para criar uma instância do gráfico desejado é preciso escolher um ou mais métodos estáticos da classe `org.jfree.chart.ChartFactory`.

Exemplos....

# MALE

## 3. Métodos de customização



- Borda
- Título e legendas
- Cor ou imagem de fundo
- Rendering Hints

Next, it is time to customize the behavior and appearance of the chart. This can be done in many ways, but the most powerful is to ask the JFreeChart object for its associated `org.jfree.chart.plot.Plot`. Several plots exist for the various chart types and their functionality is thus dependent.

# MALF

## Métodos de customização Borda



Desenha uma borda ao redor do gráfico

- Método `setBorderVisible( )`
- Controlado pelos métodos: `setBorderPaint( )`  
e `setBorderStroke( )`

# MALE

## Métodos de customização Título



- Método setTitle( )
- Posições: top, bottom, left ou right
- Ex.: chart.getTitle( ).setPosition( RectangleEdge.**BOTTOM** );

# MALF

## Métodos de customização Legenda



- Método addSubtitle ( )
- Ex.: `TextTitle subtitle1 = new TextTitle( "A Subtitle" );`  
`chart.addSubtitle( subtitle1 );`

# MALE

## Métodos de customização Cor de Fundo



- Método setBackgroundPaint( )
- Ex.: Paint p = new GradientPaint( 0, 0, Color.white,  
1000, 0, Color.green);  
chart.setBackgroundPaint( p );

# MALF

## Métodos de customização Imagem de Fundo

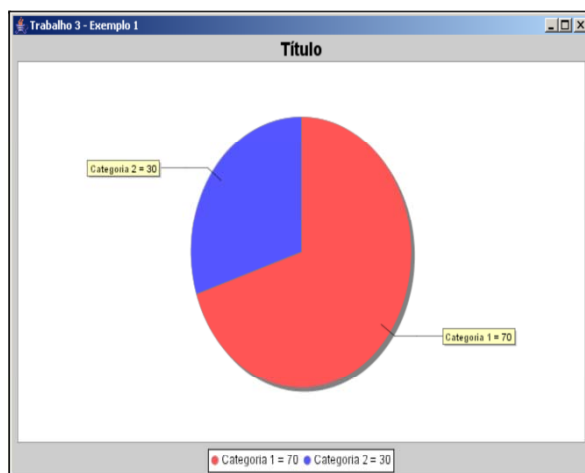


- Método `chart.setBackgroundImage( )`
- Ex.: `chart.setBackgroundImage( JFreeChart.INFO.getLogo( ));`  
`chart.setBackgroundImageAlignment( Align.TOP LEFT );`



**MALE**

## Alguns Tipos de Gráficos Pie Charts





## Pie Charts

### Exemplo simples:

```
public static void main(String[ ] args)
{
    /* (1) Gerar valores */
    DefaultPieDataset data = new DefaultPieDataset( );
    data.setValue( "Categoria 1" , new Double(70.0));
    data.setValue( "Categoria 2" , new Double(30.0));

    /* (2) Criar representação */
    JFreeChart chart;
    chart = ChartFactory.createPieChart( "Título" , data, true, true, false );

    /* (3) Customizando a aparência e comportamento */
    chart.setAntiAlias( true );

    /* (4) Definir a apresentação */
    ChartFrame frame = new ChartFrame( "Trabalho 3 – Exemplo 1" , chart );
    frame.pack( );
    frame.setVisible( true );
}
```

# MALF

## Pie Charts Exemplo simples:



Etapa 1: Gerar valores

Implementar a Interface PieDataset. Uma opção de classe que implementa essa interface é **Classe DefaultPieDataset** (org.jfree.data.general). Ela adiciona métodos bem práticos como o "setValue" que seta valores para o dataset.

```
DefaultPieDataset data = new DefaultPieDataset( );  
data.setValue( "Categoria 1" , new Double(70.0));  
data.setValue( "Categoria 2" , new Double(30.0));
```

# MALF

## Pie Charts Exemplo simples:



Etapa 2: Criar representação

Utilizar os métodos da classe ChartFactory para criar uma representação. No caso do PieChart os métodos para criá-lo são createPieChart ou createPieChart3D

```
public static JFreeChart createPieChart ( java.lang.String title, PieDataset dataset,  
                                         boolean legend, boolean tooltips, boolean urls )
```

```
JFreeChart chart;  
chart = ChartFactory.createPieChart( "Exemplo" , data, true, true, false );
```

# MALF

## Pie Charts

### Exemplo simples:



Etapa 3: Customizando a aparência e comportamento

Utilizar os métodos de classes como JFreeChart, PiePlot e PiePlot3D para customizar o gráfico.

Exemplos:

```
chart.setBackgroundImage  
chart.setAntiAlias
```

```
plot.setExplodePercent  
plot.setLabelFont  
plot.setCircular  
plot.setForegroundAlpha
```

```
chart.setAntiAlias( true );
```

# MALF

## Pie Charts

### Exemplo simples:



Etapa 4: Definir a apresentação

No exemplo usamos Swing. Mais a frente mostraremos outras apresentações usando JFreeChart.

```
ChartFrame frame = new ChartFrame( "Teste" , chart );  
frame.pack( );  
frame.setVisible( true );
```

# MALF

## Pie Charts - Exemplo 2:



## Pie Charts - Exemplo 2:



```
public class Exemplo2 extends JPanel
{
    private JFreeChart chart1, chart2, chart3, chart4;
    private ChartPanel panel1, panel2, panel3, panel4;
    private DefaultPieDataset dataset = new DefaultPieDataset();

    public PieChartExample()
    {
        dataset.setValue( "Informática", new Double( 10.0 ) );
        dataset.setValue( "Administração", new Double( 8.0 ) );
        dataset.setValue( "Ciências Econômicas", new Double( 8.0 ) );
        dataset.setValue( "Engenharia", new Double( 40.0 ) );
        dataset.setValue( "Psicologia", new Double( 8.0 ) );
        dataset.setValue( "Direito", new Double( 20.0 ) );

        chart1 = ChartFactory.createPieChart ( "Alunos de graduação (Flat)",
                                                dataset, true, true, false);

        chart2 = ChartFactory.createPieChart ( "Alunos de graduação (Exploded)",
                                                dataset, true, true, false);

        PiePlot plot = ( PiePlot )chart2.getPlot();
        plot.setExplodePercent( 3, 0.25 );
    }
}
```



## Pie Charts - Exemplo 2:



```
chart3 = ChartFactory.createPieChart3D ( "Alunos de graduação (3D)",  
dataset, true, true, false);
```

```
chart4 = ChartFactory.createPieChart3D ( "Alunos de graduação (3D transp.)",  
dataset, true, true, false);
```

```
PiePlot3D plot4 = ( PiePlot3D )chart4.getPlot();  
plot4.setForegroundAlpha( 0.6f);
```

```
this.setLayout( new GridLayout( 2, 2 ) );  
this.panel1 = new ChartPanel( chart1 );  
this.panel2 = new ChartPanel( chart2 );  
this.panel3 = new ChartPanel( chart3 );  
this.panel4 = new ChartPanel( chart4 );
```

```
this.add( panel1 );  
this.add( panel2 );  
this.add( panel3 );  
this.add( panel4 );
```

```
}
```

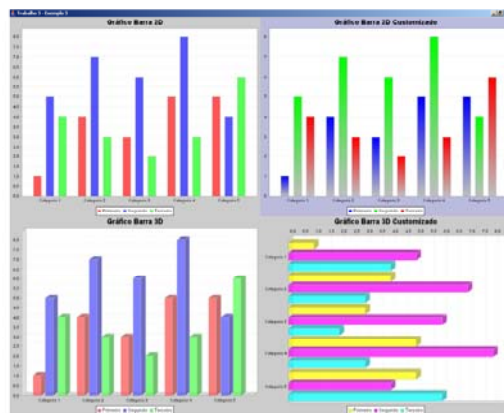
## Pie Charts - Exemplo 2:



```
public static void main( String[] args )
{
    JFrame frame = new JFrame( "Trabalho 3 - Exemplo 2" );
    PieChartExample chart = new PieChartExample();
    frame.getContentPane().add( chart, BorderLayout.CENTER );
    frame.setSize( 800, 800 );
    frame.setVisible( true );
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
}
```

**MALE**

## Alguns Tipos de Gráficos Bar Charts



# MALF

## Alguns Tipos de Gráficos Bar Charts



- 2D e com efeito 3D
- Interface CategoryDataset
- Zoom Interativo
- Customizações: Cor de fundo, mudar cores das barras, cores gradientes, espaçamento, etc.

## Bar Charts - Exemplo:



```
public class GraficosBarra extends JPanel
{
    private JFreeChart chart1, chart2, chart3, chart4;
    private ChartPanel panel1, panel2, panel3, panel4;
    private DefaultCategoryDataset dataset = new DefaultCategoryDataset ();

    public GraficosBarra()
    {
        // Linhas
        String series1 = "Primeiro" ; String series2 = "Segundo" ; String series3 = "Terceiro" ;

        // Colunas
        String category1 = "Categoria 1" ; String category2 = "Categoria 2" ;
        String category3 = "Categoria 3" ; String category4 = "Categoria 4" ;
        String category5 = "Categoria 5" ;

        // Criando Valores
        dataset.addValue(1.0, series1, category1); dataset.addValue(4.0, series1, category2);
        dataset.addValue(3.0, series1, category3); dataset.addValue(5.0, series1, category4);
        dataset.addValue(5.0, series1, category5); dataset.addValue(5.0, series2, category1);
        dataset.addValue(7.0, series2, category2); dataset.addValue(6.0, series2, category3);
        dataset.addValue(8.0, series2, category4); dataset.addValue(4.0, series2, category5);
        dataset.addValue(4.0, series3, category1); dataset.addValue(3.0, series3, category2);
        dataset.addValue(2.0, series3, category3); dataset.addValue(3.0, series3, category4);
        dataset.addValue(6.0, series3, category5);
    }
}
```

**MALE**

## Bar Charts - Exemplo:



```
// Criando representação
chart1 = ChartFactory.createBarChart( "Gráfico Barra 2D", null, null,
dataset, PlotOrientation.VERTICAL, true, false, false);

chart2 = ChartFactory.createBarChart( " Gráfico Barra 2D Customizado", null, null,
dataset, PlotOrientation.VERTICAL, true, false, false);

// Customizações do Chart 2
chart2.setBackgroundPaint( new Color(0xBBBDD));

// Pegando a referência do Plot para customização
CategoryPlot plot = chart2.getCategoryPlot();

// Pegando a referência do Number Axis
NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());

// Pegando a referência do renderer
BarRenderer renderer = (BarRenderer) plot.getRenderer();

GradientPaint gp0 = new GradientPaint( 0.0f, 0.0f, Color.blue, 0.0f, 0.0f, Color.lightGray);
GradientPaint gp1 = new GradientPaint( 0.0f, 0.0f, Color.green, 0.0f, 0.0f, Color.lightGray);
GradientPaint gp2 = new GradientPaint( 0.0f, 0.0f, Color.red, 0.0f, 0.0f, Color.lightGray);

renderer.setSeriesPaint(0, gp0);
renderer.setSeriesPaint(1, gp1);
renderer.setSeriesPaint(2, gp2);
```

**MALE**

## Bar Charts - Exemplo:



```
chart3 = ChartFactory.createBarChart3D( "Gráfico Barra 3D", null, null,  
dataset, PlotOrientation.VERTICAL, true, false, false);
```

```
chart2 = ChartFactory.createBarChart3D( " Gráfico Barra 3D Customizado", null, null,  
dataset, PlotOrientation.HORIZONTAL, true, false, false);
```

*// Customizações do Chart 4*

```
CategoryPlot plot4 = chart4.getCategoryPlot();
```

```
BarRenderer renderer4 = (BarRenderer) plot4.getRenderer();  
renderer.setDrawBarOutline( true );
```

```
CategoryAxis axis4 = plot4.getDomainAxis();  
axis4.setLowerMargin( 0.02 );  
axis4.setCategoryMargin( 0.10 );  
axis4.setUpperMargin( 0.02 );
```

```
renderer4.setSeriesPaint(0, Color.yellow);  
renderer4.setSeriesPaint(1, Color.magenta);  
renderer4.setSeriesPaint(2, Color.cyan);
```



## Bar Charts - Exemplo:

*// Criando a apresentação*

```
this.setLayout( new GridLayout( 2, 2 ) );
this.panel1 = new ChartPanel( chart1 );
this.panel2 = new ChartPanel( chart2 );
this.panel3 = new ChartPanel( chart3 );
this.panel4 = new ChartPanel( chart4 );
this.add( panel1 );
this.add( panel2 );
this.add( panel3 );
this.add( panel4 );
}

public static void main( String[ ] args )
{
    JFrame frame = new JFrame( "Trabalho 3 - Exemplo 3" );

    GraficosBarra chart = new GraficosBarra();

    frame.getContentPane().add( chart, BorderLayout.CENTER );
    frame.setSize( 800, 800 );
    frame.setVisible( true );
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
}
}
```



# SAVIO

## Alguns Tipos de Gráficos Gráficos Dinâmicos



Rodar o código e explicar

# SAVIO



## Outros recursos

- JDBC:
  - API para gerar datasets através de banco de dados relacionais
  - Datasets: JDBC PieDataset, JBCCategoryDataset, JDBCXYDataset

# SAVIO

## Outros recursos

- Gráficos combinados:

- Tipos: Domínio compartilhado  
Imagem compartilhada
- Plots: CombinedDomainCategoryPlot  
CombinedRangeCategoryPlot  
CombinedDomainXYPlot  
CombinedRangeXYPlot

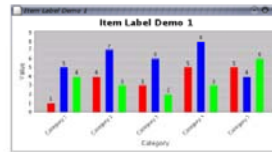


```
CategoryAxis domainAxis = new CategoryAxis( "Categoria" );  
CombinedDomainCategoryPlot plot = new CombinedDomainCategoryPlot(domainAxis);  
plot.add(subplot1, 2);  
plot.add(subplot2, 1);
```

# SAVIO

## Outros recursos

- Legenda em gráficos



- Exibir anotações dentro de gráficos

- Problemas na versão atual: Alguns Renderers não suportam, alguns eixos não são ajustados automaticamente.

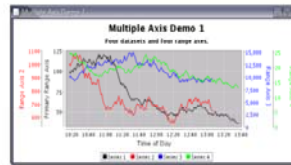
```
CategoryItemRenderer renderer = plot.getRenderer( );  
renderer.setLabelGenerator( new StandardCategoryItemLabelGenerator( ));  
renderer.setItemLabelsVisible( true );
```

# SAVIO

## Outros recursos

- Gráficos Múltiplos

- Múltiplos datasets, renderers, eixos de imagem e eixos de domínio



Métodos:

```
plot.setSecondaryRangeAxis(0, axis);  
plot.setSecondaryDataset(0, dataset);  
plot.setSecondaryRenderer(0, renderer);
```

# SAVIO

## Outros recursos



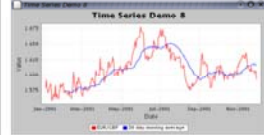
- ToolTips

- JFreeChart inclui mecanismos de exibir, gerar e coletar ToolTips
- **Gerar:** PieToolTipGenerator, CategoryToolTipGenerator, etc.
- **Coletar:** Classe ChartRenderingInfo
- **Exibir:** Classe ChartPanel: setDisplayToolTips(boolean flag);

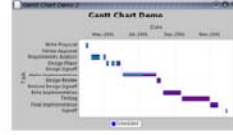
# SAVIO

## Outros Tipos de Gráficos

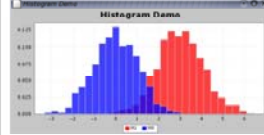
### Séries Temporais



### Gráfico de Gantt



### Histograma



### Gráfico Cascata



# SAVIO

## Outros Tipos de Gráficos

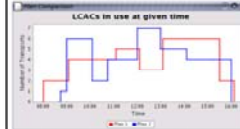
Gráfico Diferença



Gráfico de Área



Gráfico Escada



Outros



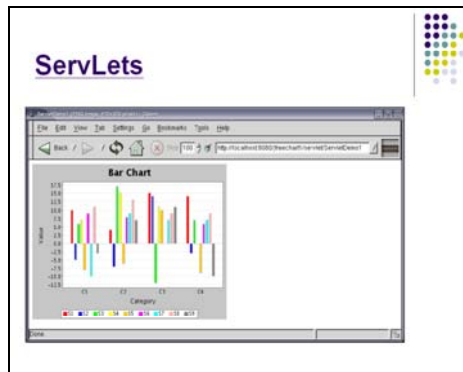




## 4. Tipos de Apresentações

- Aplicação Swing
- Servlets
- Applets
- JSP

# NERD



O "API Servlet" é uma extensão Java especialmente desenvolvida para aplicações que rodam em servidores. Os Servlets não são específicos para servidores WEB, mas como os servidores HTTP (servidores WEB) são importantes, criou-se uma versão específica de Servlet, o `HttpServlet`, o qual será tratado apenas como Servlet neste documento. Uma grande vantagem do Servlet é que este é carregado apenas uma vez pelo servidor, ficando residente na memória e prontamente disponível para processar chamadas, até mesmo a nível de threads concorrentes. Os Servlets são portáteis e provêm mecanismos de segurança, características herdadas do Java, além de contar com a orientação à objeto em sua forma de programação.

As classes do pacote `javax.servlet.*` (e sub-pacotes), usadas pelo exemplo `Servlet1`, não fazem parte do Java 2 Standard Edition (J2SE). Para compilar este código usando J2SE, você precisará obter o arquivo `servlet.jar`

A classe abstrata `HttpServlet` possui vários métodos:

- `doGet`, if the servlet supports HTTP GET requests
- `doPost`, for HTTP POST requests
- `doPut`, for HTTP PUT requests
- `doDelete`, for HTTP DELETE requests
- `init` and `destroy`, to manage resources that are held for the life of the servlet
- `getServletInfo`, which the servlet uses to provide information about itself

No nosso caso, este exemplo class `Servlet1` estende `HttpServlet` e usa o método `doGet`. Este método é chamado pela engine servlet quando uma chamada é feita pelo cliente.

`doGet(HttpServletRequest request, HttpServletResponse response)`

`request` – é um objeto `HttpServletRequest` que contém o que o servlet recebe do cliente.

`response` – objeto `HttpServletResponse` que contém a resposta que o servlet manda p/ o cliente.

`java.io.IOException` – se um input ou output error é detectado quando o servlet carrega o GET request.

[ServletException](#) – se o request para o GET não pode ser carregado.

# NERD

## ServLets

```
public class Servlet1 extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException
    {
        OutputStream out = response.getOutputStream();
        try
        {
            DefaultCategoryDataset dataset = new DefaultCategoryDataset();
            dataset.addValue(10.0, "S1", "C1"); dataset.addValue(4.0, "S1", "C2");
            dataset.addValue(15.0, "S1", "C3"); dataset.addValue(14.0, "S1", "C4");
            dataset.addValue(-5.0, "S2", "C1"); dataset.addValue(-7.0, "S2", "C2");
            dataset.addValue(14.0, "S2", "C3"); dataset.addValue(-3.0, "S2", "C4");
            dataset.addValue(6.0, "S3", "C1"); dataset.addValue(17.0, "S3", "C2");
            dataset.addValue(-12.0, "S3", "C3"); dataset.addValue(7.0, "S3", "C4");
            dataset.addValue(7.0, "S4", "C1"); dataset.addValue(15.0, "S4", "C2");
            dataset.addValue(11.0, "S4", "C3"); dataset.addValue(0.0, "S4", "C4");
            dataset.addValue(-8.0, "S5", "C1"); dataset.addValue(-6.0, "S5", "C2");
            dataset.addValue(10.0, "S5", "C3"); dataset.addValue(-9.0, "S5", "C4");
            dataset.addValue(9.0, "S6", "C1"); dataset.addValue(8.0, "S6", "C2");
            dataset.addValue(null, "S6", "C3"); dataset.addValue(6.0, "S6", "C4");
            dataset.addValue(-10.0, "S7", "C1"); dataset.addValue(9.0, "S7", "C2");
```

## ServLets

```
dataset.addValue(7.0, "S7", "C3"); dataset.addValue(7.0, "S7", "C4");
dataset.addValue(11.0, "S8", "C1"); dataset.addValue(13.0, "S8", "C2");
dataset.addValue(9.0, "S8", "C3"); dataset.addValue(9.0, "S8", "C4");
dataset.addValue(-3.0, "S9", "C1"); dataset.addValue(7.0, "S9", "C2");
dataset.addValue(11.0, "S9", "C3"); dataset.addValue(-10.0, "S9", "C4");

JFreeChart chart;
chart = ChartFactory.createBarChart("Bar Chart", "Category", "Value", dataset,
    PlotOrientation.VERTICAL, true, true, false);
response.setContentType("image/png");
ChartUtilities.writeChartAsPNG(out, chart, 400, 300);
}
catch (Exception e) {
    System.err.println(e.toString());
}
finally {
    out.close();
}
}
```

Depois disso o servlet realiza alguns passos:

- an OutputStream reference is obtained for returning output to the client; (OutputStream out = response.getOutputStream(); )

- a chart is created;

JFreeChart chart = ChartFactory.createBarChart(..);

- the content type for the response is set to image/png. This tells the client what type of data it is receiving;

response.setContentType("image/png");

- a PNG image of the chart is written to the output stream;

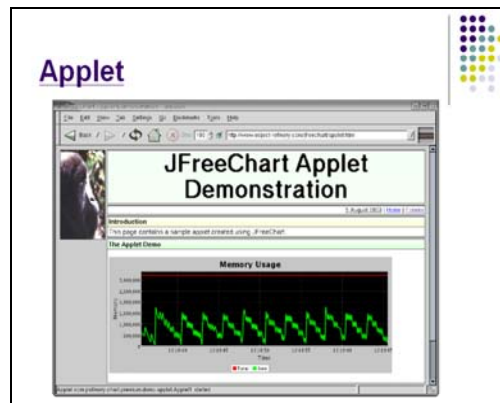
ChartUtilities.writeChartAsPNG(out, chart, 400, 300);

- the output stream is closed.

out.close();

Este exemplo retorna uma imagem png de um gráfico de barra usando JFreeChart. Quando este programa roda, o servlet retornara somente a imagem para o cliente( web browser ), sem algum HTML.

# NERD



Um Applet é um programa escrito em Java, o qual é executado no cliente, sendo o oposto ao Servlet. A restrição do uso do Applet é a forma pela qual ele é "baixado" na máquina cliente. Todo Applet é "baixado" por demanda, o que diminui a performance. Uma solução para esta perda de performance seria a utilização do "channel" como proposto mais à frente.

Primeiramente tem que ter em mentes os três problemas :

- **Browser support**

A maioria dos web browsers possuem suporte para a ultima versão do Java (JDK 1.4), ou seja, não terao problemas rodando o applets que usa JFreeChart( lembrando que o JFreeChart roda em qualquer versão do JDK a partir da 1.2.2 pra cima ).

Entretanto, A maioria dos usuários da internet usam "the one web browser" – Microsoft Internet Explorer ( MSIE ) – que suporta apenas a versão do Java ( JDK 1.1 ) que é agora uma versão inaceitável. Esse é um problema porque o applet que usa JFreeChart não irá trabalhar com a instalação padrão do Internet Explorer. A solução é baixar e instalar o Sun's Java plugin, mas isto é um inconveniente para muitas pessoas. Este problema acarretou com que muitos programadores abandonassem seus planos para fazer um applet e como alternativa escolheram o Java Servlets.

- **Security**

Quando um applet roda no seu web browser, ele esta restrito para algumas operações.

Por exemplo, num applet tipicamente não ira ser permitido leitura ou escrita para o local filesystem. Você devera saber que algumas funções do JFreeChart tais como, a opção para salvar o gráfico para o formato PNG via pop-up menu, não ira funcionar. Se você quiser que funcione, então vai precisar estudar os mecanismos de segurança em maiores detalhes.

## - Code size

Este problema é causado pelo tamanho do código requerido para rodar seu applet. Antes de rodar o applet, o código ( tipicamente compactado em arquivos .jar ) tem que ser baixado para a maquina do usuário. Como existem usuários com conexões ruins, tipo modem, o tamanho do código pode ser um problema.

O código do JFreeChart é distribuído em um arquivo .jar que possui por volta de 500KB de tamanho. Isso não é muito grande, especialmente quando consideramos o numero de variedades de gráficos que o JFreeChart possui. Mas ao mesmo tempo, isso não é exatamente ótimo para usuários que se conectam na internet por modem. E alem disso precisara do arquivo JCommon.jar ( que possui por volta de 170KB ) somado ao código do applet.

Dois aspectos deste exemplo de applet são interessantes, o código fonte que é usado para criar o applet e o arquivo HTML que é usado para chamar o applet.

## HTML

```
<APPLET ARCHIVE="jfreechart-0.9.4-premium-demo-applets.jar,jfreechart-0.9.4.jar,
jcommon-0.7.1.jar" CODE="com.jrefinery.chart.premium.demo.applet.Applet1"
width=640 height=260 ALT="You should see an applet, not this text.">
</APPLET>
```

Podemos notar que existem três arquivos .jar sendo referenciados. O primeiro contem a classe applet ( código fonte que sera visto mais a frente ), enquanto que os outros dois arquivos .jar são as bibliotecas de classes JFreeChart e o JCommon.

<http://www.object-refinery.com/jfreechart/applet.html>

```
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JApplet;
```

### 16.2.1 What is SVG?

Scalable Vector Graphics (SVG) is a standard language for describing twodimensional graphics in XML format. It is a Recommendation of the World Wide Web Consortium (W3C).

### 16.2.2 Batik

Batik is an open source toolkit, written in Java, that allows you to generate SVG content. Batik is available from:

<http://xml.apache.org/batik>

At the time of writing, the latest stable version of Batik is 1.5.

# NERD

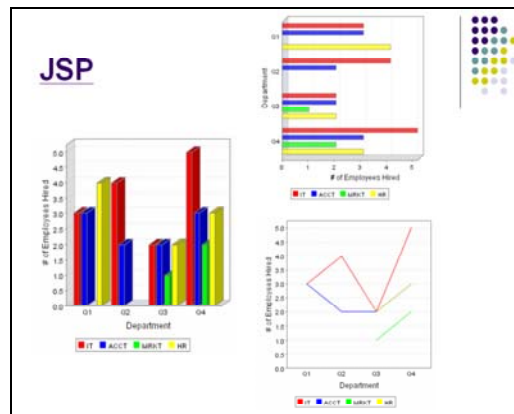
## Applet

Problemas:

- Suporte a diferentes browsers
- Segurança
- Tamanho do código fonte



## NERD



The content of [categorychart.jsp](#) should look familiar as it uses many of the same constructs as [piechart.jsp](#). It declares a `DatasetProducer` instance, and then lists a set of JSP tags to manage the charts. This JSP file contains separate tags for chart types 'horizontalBar3D', 'verticalBar3D', and 'line'. All three charts represent the same `CategoryDataset` (provided by the `DatasetProducer`). Only two new tag attributes are introduced: `axislabel` and `yaxislabel`. Figure 2 shows the resulting Web page. Below are the three chart images produced in greater detail:

<http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/index.html>

What is JSP?

Java Server Pages (JSP) is a technology that lets you mix regular, static HTML with dynamically-generated HTML. Many Web pages that are built by CGI programs are mostly static, with the dynamic part limited to a few small locations. But most CGI variations, including servlets, make you generate the entire page via your program, even though most of it is always the same. JSP lets you create the two parts separately

JavaServer Pages (JSP) is a Sun Microsystems specification for combining Java with HTML to provide dynamic content for Web pages. When you create dynamic content, JSPs are more convenient to write than HTTP servlets because they allow you to embed Java code directly into your HTML pages, in contrast with HTTP servlets, in which you embed HTML inside Java code. JSP is part of the Java 2 Enterprise Edition (J2EE).

JSP enables you to separate the dynamic content of a Web page from its presentation. It caters to two different types of developers: HTML developers, who are responsible for the graphical design of the page, and Java developers, who handle the development of software to create the dynamic content.

Because JSP is part of the J2EE standard, you can deploy JSPs on a variety of platforms, including WebLogic Server. In addition, third-party vendors and application developers can provide JavaBean components and define custom JSP tags that can be referenced from a JSP page to provide dynamic content.

In `piechart.jsp`, there are two parent tags at work: `chart` and `img`. The `chart` tag is responsible for overall chart configuration. Included are tags and attributes for specifying the type of chart to be rendered, a title for the chart, background color, and more. The most important piece of information within the `chart` tag is recorded in a nested tag:

```
<cewolf:data>
  <cewolf:producer id="pieChartView" />
</cewolf:data>
```

The data tag specifies the Dataset to be used when rendering the chart. Object pieChartView is an instance of EmployeesByDept that you declare at the top of the page:

```
<jsp:useBean id="pieChartView" class="examples.EmployeesByDept"/>
```

The other parent tag just below chart is named img. This tag specifies how the chart is to be *displayed*. It contains attributes to control the graphic: height, width, border, and so on. The chart and img tags are linked to one another through the chart 'id' attribute and the img 'chartid' attribute. In this case, you've given your chart an id of 'pieExample'. These tag definitions produce the following pie chart image:



This is a good start, but you can improve upon this example. Many charts like to include text (in addition to the text inside the legend) to describe each data series. You have the ability to add text to the image itself. Another capability you have with JFreeChart and Cewolf is to include text as a tool tip. Now, add a tool tip that displays when a user's mouse hovers over a particular series on the image. To accomplish this, you need to obtain an instance of type PieToolTipGenerator that implements a method called generateToolTip. The easiest way to do this is to add an inner class to EmployeesByDept.java. [Here](#) is the updated Java file, in which you added the following code:

```
/**Inner Class to generate tool tips for data.*/
PieToolTipGenerator pieTG = new PieToolTipGenerator() {
    public String generateToolTip(PieDataset dataset,
        Comparable section, int index) {
        return String.valueOf(dataset.getValue(index) +
            " employees total" );
    }
};
public PieToolTipGenerator getPieTG() {
    return this.pieTG;
}
```

The new and improved [piechart.jsp](#) includes a second pie chart with many enhancements. Near the top of the page, you obtain an instance of your PieToolTipGenerator:

```
<% pageContext.setAttribute("pieChartViewToolTips",
    pieChartView.getPieTG()); %>
```

An additional tag, map, needs to be included within the img tag in order to use tool tips:

```
<cewolf:map tooltipgeneratorid="pieChartViewToolTips"/>
```

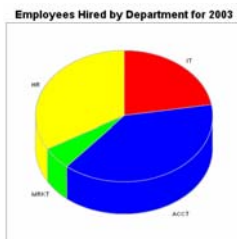
While adding tool tips to this second pie chart, I modified the JSP tags to include additional features. The chart image is now of type 'pie3d' and the chart's legend is now generated separately from the chart. More importantly, you now pass an input parameter, inputYear, using the producer tag (nested within the chart tag). The ability to parameterize a chart is the key to providing users with dynamic charts. In a typical application, these parameter values might be input from an HTML form that a user submits to customize his or her chart:

```
<cewolf:producer id="pieChartView" >
    <cewolf:param name="year" value="<%= (Serializable)inputYear%>" />
</cewolf:producer>
```

In EmployeesByDept, you check for an input parameter. If a parameter is supplied, you filter our data by year. Figure 1 shows the JSP page displaying both pie charts. Note that the second chart displays the tool tip text '6 employees total' over the series that represents HR employees.



Here's the 3-dimensional pie chart graphic (without a legend, which was a separate image in this example):



Using `<jsp:useBean>`

To use a JavaBeans component, the first thing you need to do is to enable the use of a bean within your current template, through a process called *instantiation*. You use the `<jsp:useBean>` action to instantiate beans. Table 1 shows the attributes of this action.

**Table 1 Attributes of the `<jsp:useBean>` Action**

Attribute	Use
id	This attribute specifies the name of the bean and how you will refer to it in the page.
scope	This attribute specifies the scope in which you want to store the bean instance. It can be set to page (the default), session, request, or application.
class	This attribute specifies the Java class that the bean is drawn from. If you have specified beanName, you do not have to specify class.
beanName	This attribute specifies the name of a bean that is stored on the server. You refer to it as you would a class (for example, com.projectalpha.PowerBean). If you have specified class, you do not need to specify beanName.
type	This attribute specifies the type of scripting variable returned by the bean. The type must relate to the class of the bean.

The following is a simple example of using `<jsp:useBean>` to instantiate `java.util.Date` as an entity bean:

```
<jsp:useBean id="today" class="java.util.Date" />
```

After a bean is instantiated, you can use it in two ways. First, two actions, `<jsp:getProperty>` and `<jsp:setProperty>`, allow you to set values and retrieve values in a bean. Second, you can directly access the methods within a bean by using Java code in scriptlets.

```
<%@ taglib uri="http://www.jspcentral.com/tags" prefix="public" %>
<public:loop>      ...      </public:loop>
```

## Cewolf

Cewolf can be used inside a Servlet/JSP based web application to embed complex graphical charts of all kinds (e.g. line, pie, bar chart, plots, etc.) into a web page. Therefore it provides a full featured tag library to define all properties of the chart (colors, strokes, legend, etc.). Thus the JSP which embeds the chart is not polluted with any java code. Everything is described with XML conform tags.

Paginas :

[http://nettrace.blogspot.com/archive/2004/09/09/java\\_draw\\_plot\\_graph\\_chart\\_and\\_report.html](http://nettrace.blogspot.com/archive/2004/09/09/java_draw_plot_graph_chart_and_report.html)

# NERD

## JSP



```
<%@page contentType= "text/html" %>
<%@page import= "org.jfree.data.*" %>
<%@page import= "java.io.Serializable" %>

<%@taglib uri= ' /WEB-INF/cewolf.tld ' prefix='cewolf' %>

<jsp:useBean id="barChartView" class="examples.EmployeesByDeptAndQtr"/>

<% String yAxis = "# of Employees Hired" ; String xAxis = "Department" ; %>

<html>
<body>
<table align= "center" border= "0" width= "60%" >
<tr><td colspan="3">
<H2 ALIGN= "CENTER" >Employees by Department and Quarter</H2></td></tr>
<tr><td>
<cewolf: chart id="barExample3DH" type="horizontalBar3D"xaxislabel=" <%=xAxis%> "
yaxislabel="<%=yAxis%> ">
  <cewolf: data>
    <cewolf:producer id= "barChartView" />
  </cewolf:data>
  <cewolf: colorpaint color= "#FFFFFF" />
</cewolf:chart>
<cewolf: img chartid ="barExample3DH" renderer="cewolf" border="0" width="300"
height="300" />
</td>
```

# NERD

## JSP




```
<td>
  <cewolf: chart id="barExample3DV" type="verticalBar3D" xaxislabel=" <%=xAxis%> "
  yaxislabel=" <%=yAxis%> " >
    <cewolf:data>
      <cewolf: producer id="barChartView" />
    </cewolf:data>
    <cewolf: colorpaint color="#FFFFFF" />
  </cewolf:chart>
  <cewolf: img chartid="barExample3DV" renderer="cewolf" border="0" width="300"
  height="300" />
</td>

<td>
  <cewolf:chart id="lineExample" type="line"
  xaxislabel="<%=xAxis%>" yaxislabel="<%=yAxis%>">
    <cewolf:data>
      <cewolf:producer id="barChartView" />
    </cewolf:data>
    <cewolf:colorpaint color="#FFFFFF" />
  </cewolf:chart>
  <cewolf: img chartid="lineExample" renderer="cewolf" border="0" width="300"
  height="300" /> </td> </tr>

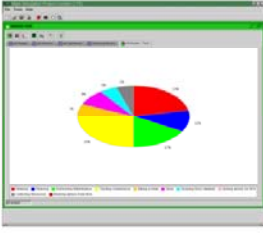

</body>
</html>
```

# SAVIO

**Softwares que utilizam JFreeChart**



- The Mars Simulation Project  
<http://mars-sim.sourceforge.net/>



<http://mars-sim.sourceforge.net/>

The Mars Simulation Project is a free software Java project to create a simulation of future human settlement of Mars.

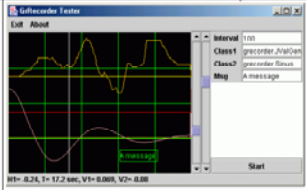
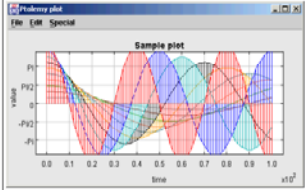
O projeto é um programa que simula o futuro dos humanos em marte.  
Existem pessoas, veículos, latitude, longitude, location, ...

Torta → Gender, Location, Personality, Job, Task, Mission, Health  
Barra → Hunger, Fatigue, Stress, Performance

Na barra pode ser selecionado mais de uma coluna, e atualiza em tempo real de Marte.

# SAVIO

## Concorrentes

<b>GrRecorder</b> <a href="http://pages.infinet.net/bigfeet/grrecorder_.html">http://pages.infinet.net/bigfeet/grrecorder_.html</a> Licença: GNU Lesser GPL Tipo: Aplicativo Produtor: Marcel St-Amant	
<b>PtPlot</b> <a href="http://ptolemy.eecs.berkeley.edu/java/ptplot/">http://ptolemy.eecs.berkeley.edu/java/ptplot/</a> Licença : GNU Lesser GPL Tipo: Component Produtor: Ptplot	

<http://www.java2s.com/Product/GUI-Tools/Chart.htm>

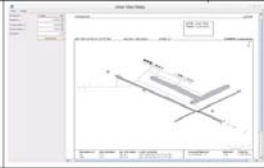

## GrRecorder

"GrRecorder (ou Graph Recorder Component) é usado para mostrar gráficos de tempo variável com valores analógicos. Isso simula instrumentos de laboratório tipo um osciloscópio.

## Ptplot

"Ptplot 5.3 é um 2D data plotter e um histograma implementado em Java. Ptplot pode ser usado com um applet ou aplicação , ou pode ser usado em seu próprio applet ou aplicação."

# SAVIO

Concorrentes	
<b>Chart Maker</b> <a href="http://www.xineo.net/chartmaker.jsp">http://www.xineo.net/chartmaker.jsp</a> Licença: Xineo Freeware License Tipo: Componente Produtor: Xineo.net	
<b>KavaChart</b> <a href="http://www.ve.com/kavachart/index.html">http://www.ve.com/kavachart/index.html</a> Licença: Commercial Tipo: Componente Produtor: Visual Engineering	

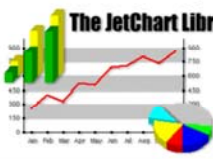
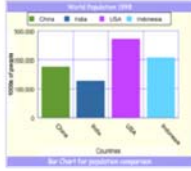
## Chart Maker

"Xineo Chart Maker é capaz de gerar gráficos de aeroporto no formato PDF."

## KavaChart

"KavaChart tem tudo que você precisa para fazer um gráfico industrial e gráficos para seu software. Mesmo que você queira criar um aplicativo cliente que requer interatividade."

# SAVIO

Concorrentes	
<b>JetChart</b> <a href="http://www.jinsight.com/jetchart/index.html">http://www.jinsight.com/jetchart/index.html</a> Licença: Comercial Tipo: Componente Produtor: Jinsight Informatica	 <p>The JetChart Library</p>
<b>AgileBlox Chart</b> <a href="http://www.elansoft.com/web/home.html">http://www.elansoft.com/web/home.html</a> Licença: Comercial Tipo: Componente Produtor: Elansoft	 <p>AgileBlox Chart</p>

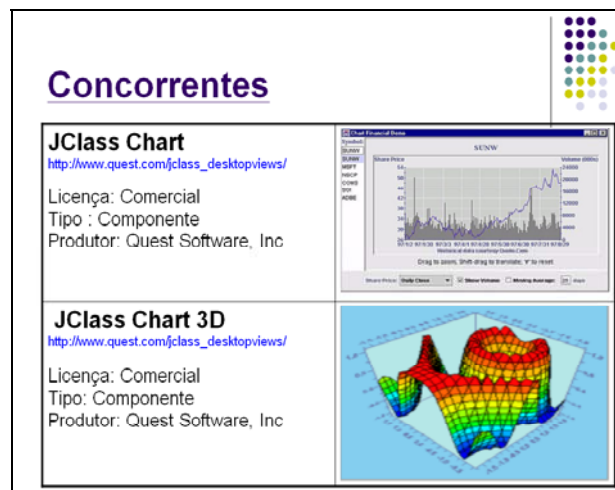
## JetChart

"The JetChart Library possui uma visualização de dados e análises em diferentes formas de gráficos. Os dados podem ser representados em diferentes maneiras, tais como, uma sequência de pontos conectados, barras, colunas, áreas preenchidas e mais."

## AgileBlox Chart

"AgileBlox Chart serve para gerar gráficos de aplicativos comercial."

# SAVIO



## JClass Chart



"JClass Chart serve para construir gráficos comerciais e científicos que usa o formato de texto ( rich text format ) para customizar labels ou misturar imagens e URLs com texto."

## JClass Chart 3D

"JClass Chart 3D automaticamente fornece todas as rotações, escalas, anotações e cálculos de perspectiva."



# SAVIO

<h2>Concorrentes</h2>	
<b>EasyCharts</b> <a href="http://www.objectplanet.com/EasyCharts/">http://www.objectplanet.com/EasyCharts/</a> Licença: Comercial Tipo : Componente Produtor: ObjectPlanet, Inc	
<b>Chart2D</b> <a href="http://chart2d.sourceforge.net/">http://chart2d.sourceforge.net/</a> Licença: OpenSource Tipo: Componente Produtor: Chart2D	

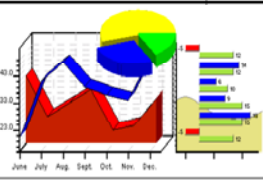
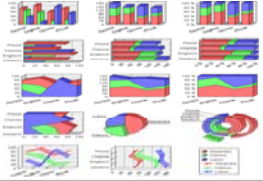
## EasyCharts

"EasyCharts serve para applets e servlets, que habilita programadores para adicionar gráficos e grafos em java e em aplicações web com poucas linhas de código."

## Chart2D

"Biblioteca usada para visualizar gráficos de 2-dimensões."

# SAVIO

<h2>Concorrentes</h2>	
<b>RChart</b> <a href="http://www.java4less.com/charts_e.htm">http://www.java4less.com/charts_e.htm</a> Licença: Comercial Tipo : Componente Produtor: J4L	
<b>ChartCat</b> <a href="http://www.netcat.it/java-graph-and-chart-engine/">http://www.netcat.it/java-graph-and-chart-engine/</a> Licença: Comercial Tipo: Componente Produtor: NetCat Inc	

## RChart

"RChart serve para adicionar gráficos para sua aplicativos Java[TM] ou websites (pode ser usado com ASP, Php, JSP[TM] ). RChart é um dos mais populares pois é muito simples mas poderoso e muito útil."

## ChartCat

"ChartCat é uma bliblioteca de gráficos orientada a objeto para java que fornece o máximo de flexibilidade, funcionalidade e reusabilidade."