

# Programando *Applets*

Rafael Santos

Instituto Nacional de Pesquisas Espaciais  
`www.lac.inpe.br/~rafael.santos`

# Tópicos

- 1 Introdução
- 2 Conceitos Básicos
- 3 *Applets*
- 4 Eventos
- 5 *Java2D*
- 6 Entrada e Saída
- 7 Extras

# Objetivos

- Entender o que é uma *applet*.
- Entender quais aplicações podem ser criadas como *applets*.
- Aprender a programar *applets* simples.
  - Aprender um pouco sobre programação orientada a eventos.
  - Aprender um pouco de *Java2D*.
  - Aprender a trabalhar com componentes de interfaces gráficas.

# Pré-requisitos

Não obrigatórios, mas ajuda bastante...

- Programação orientada a objetos e sintaxe de Java.
- Programação com elementos de interfaces gráficas (componentes).
- Programação orientada a eventos.
- Programação de gráficos com *Java2D*.
- Entrada e saída, acesso a bancos de dados.
- HTML.

# Conteúdo e material

- Exposição, sem laboratório.
- Não veremos segurança (*applets* assinadas, etc.).
- Não veremos aplicações complexas, uso de som, técnicas avançadas.
- Somente trechos significativos do código-fonte.
- Código-fonte e material de apresentação serão colocados no site [www.lac.inpe.br/~rafael.santos](http://www.lac.inpe.br/~rafael.santos).

# Applets...

- Pequenas aplicações copiadas de um servidor e executadas em um navegador.
- A partir de 1995, causaram grande interesse por Java.
  - WWW era muito estática!
  - Aplicações somente do lado do servidor (CGIs).
- Forma de enviar aplicações com interfaces ricas para clientes.
  - Sempre versões atualizadas da aplicação.
  - Ônus de processamento do lado do cliente.

# *Applets...*

- Interesse tem decaído:
  - Servidores mais eficientes, interfaces em HTML.
  - Estabelecimento de Java como linguagem de programação.
  - **Problemas de compatibilidade entre máquinas virtuais.**
- Ainda vale a pena aprender:
  - Fim da incompatibilidade, facilidade de *deployment*.
  - Criação de pequenas aplicações.
  - Tarefas como visualização, demonstrações, etc.
  - Existem outras abordagens, mas *applets* são em Java!

## Por que usar *applets*?

- Maneira mais fácil de fazer *deployment* de pequenas aplicações com interfaces ricas:
  - Usuário não precisa instalar nada no disco (exceto a VM).
  - Versões novas automaticamente carregadas.
  - Mais segurança do que aplicações inteiras.
- OK, existem algumas desvantagens...
  - Restrições à execução.
  - Restrições ao acesso a dados (exceto *applets* assinadas).
- Outras alternativas existem...
  - *Java Web Start*.



# Classes, campos e métodos

- Uma aplicação (ou *applet*) em Java é uma **classe**.
- Pedacos da aplicação podem estar distribuídos em várias classes.
- Usamos **instâncias** de classes em outras classes.
- Uma classe pode conter **campos** (que representam valores pertinentes àquela classe).
- Uma classe pode conter **métodos** (que descrevem que processamento deve ser feito naquela classe).
- Uma classe pode conter **construtores**, métodos especiais que são executados quando criamos instâncias das classes.

# Classes, campos e métodos

- Campos e métodos podem ser:
  - **Públicos**: métodos de outras classes podem acessar, não é uma boa idéia para campos.
  - **Privados**: somente métodos da mesma classe podem acessar, faz mais sentido para campos.
  - **Protegidos**: somente métodos da mesma classe e de classes herdeiras podem acessar<sup>1</sup>, melhor ainda.
- Comum: métodos públicos, campos privados ou protegidos: **encapsulamento**.

---

<sup>1</sup>Lembrar que isso só funciona com pacotes!

# Classes, campos e métodos: exemplo

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

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

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

# Herança

- Usamos uma classe já existente (**ancestral**) para derivar uma nova classe.
- Nova classe *herda* campos e métodos públicos e protegidos da classe ancestral.
- Nova classe (**herdeira**) pode sobrepor métodos e campos.
- Métodos na classe herdeira podem executar métodos na ancestral através de `super`.

# Herança: exemplo

```
public class CirculoPreenchido extends Circulo
{
    protected Color corP;

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

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

# Interfaces

- Diferente do conceito de interfaces gráficas!
- Tipos de classes onde métodos são declarados mas não implementados.
- Servem como *contratos* para classes que implementarão estas interfaces.
- Classes que implementam as interfaces **devem** implementar os métodos declarados.

# *Applets*

- *Applets* herdam da classe `JApplet`.
  - Alguns métodos pré-definidos, podem ser sobrepostos.
- Ciclo de Vida:
  - Método `init`: executado quando a *applet* é carregada pela primeira vez.
  - Método `start`: executado quando o navegador carrega ou volta à página com a *applet*.
  - Método `stop`: executado quando o navegador deixa a página com a *applet*.
  - Método `destroy`: executado quando o navegador é fechado.

# *Applets*

- Métodos “mágicos”, executados automaticamente pelo navegador.
- Herdam também de `Component/Container`, podemos sobrepor método `paint`.
- Herdam também de `Component/Container`, podemos sobrepor método `paint`.
- Outro método interessante: `resize` garante tamanho pedido (mas não no navegador!).



# Applet mínima

```
package applets;  
  
import javax.swing.JApplet;  
  
public class Applet0 extends JApplet  
{  
}
```



# Applet mais interessante

```
package applets;

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

public class Applet1 extends JApplet
{

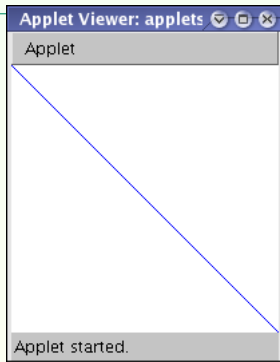
    public void paint(Graphics g)
    {
        Graphics2D g2d = (Graphics2D)g;
        g2d.setColor(Color.BLUE);
        g2d.drawString("Olá, Mundo",5,15);
    }
}
```



# Applet ainda mais interessante

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

public class Applet2 extends JApplet
{
    private int largura, altura;
    public void init()
    {
        largura = getWidth(); altura = getHeight();
    }
    public void paint(Graphics g)
    {
        Graphics2D g2d = (Graphics2D)g;
        g2d.setColor(Color.BLUE);
        g2d.drawLine(0,0, largura, altura);
    }
}
```



# *Applets*

até agora...

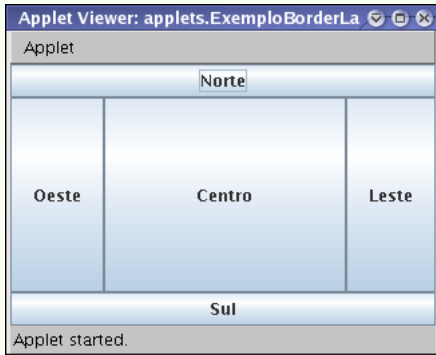
- Sobreposição de método `init` para inicializar atributos da *applet*.
- Sobreposição de método `paint` para desenhar algo na *applet*.
- Isso é simples mas pouco eficiente!
- Como desenhar uma interface com usuário?
- Como *reagir* à interação com usuário?

# *Layouts*

- **Como desenhar uma interface com usuário?**
- Uma *applet* contém uma única janela.
- Podemos usar vários componentes de interfaces com usuários em uma *applet*.
- Devemos ter uma forma de organizar estes componentes → *Layouts*.
- Vários tipos de *layouts* em Java, combinações possíveis.

# BorderLayout

- O *layout* mais comum, é o *default*.
- Cinco áreas nomeadas.

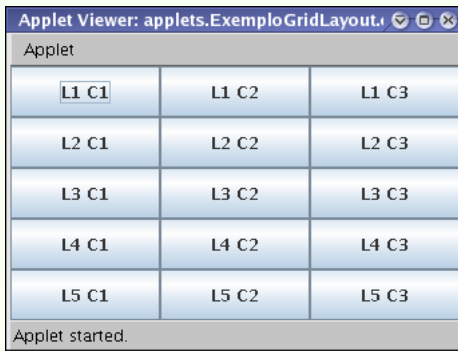


# BorderLayout

```
public class ExemploBorderLayout extends JApplet
{
    public void init()
    {
        setLayout(new BorderLayout());
        add(new JButton("Norte"),BorderLayout.NORTH);
        add(new JButton("Sul"),BorderLayout.SOUTH);
        add(new JButton("Leste"),BorderLayout.EAST);
        add(new JButton("Oeste"),BorderLayout.WEST);
        add(new JButton("Centro"),BorderLayout.CENTER);
    }
}
```

# GridLayout

- Várias áreas iguais, grade regular.





# GridLayout

```
public class ExemploGridLayout extends JApplet
{
    public void init()
    {
        setLayout(new GridLayout(5,3));
        for(int l=0;l<5;l++)
            for(int c=0;c<3;c++)
                add(new JButton("L"+(l+1)+" C"+(c+1)));
    }
}
```

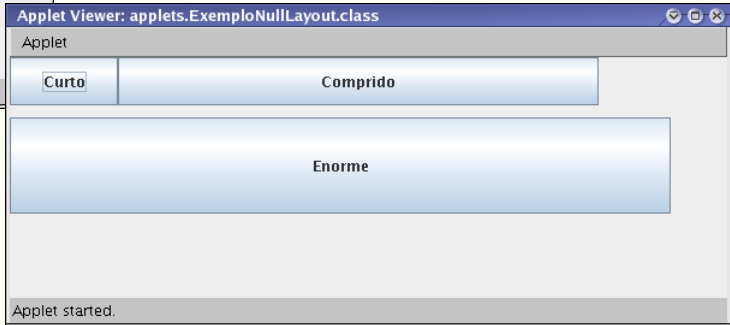
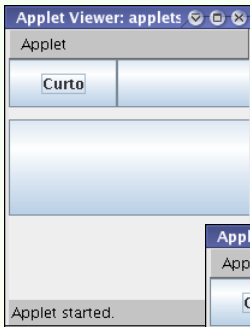
## Sem *layouts*

- Podemos usar `null` para `setLayout`.
- Posicionamento de componentes deve ser feito com chamadas a `setBounds` dos componentes.
- Desaconselhado:
  - *Look-and-feel* pode causar diferenças!
  - Será necessário calcular o tamanho da *applet* manualmente.
  - Trabalho pode ser muito manual.
- Mas dá grande flexibilidade...

# Sem layouts

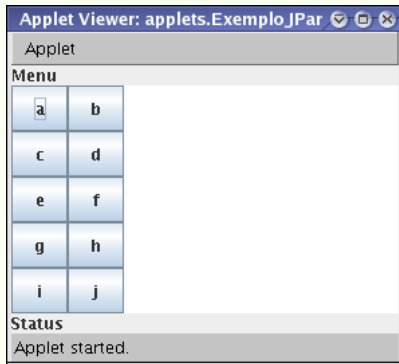
```
public class ExemploNullLayout extends JApplet
{
    public void init()
    {
        setLayout(null);
        JButton b1 = new JButton("Curto");
        JButton b2 = new JButton("Comprido");
        JButton b3 = new JButton("Enorme");
        b1.setBounds(0,0,90,40);
        b2.setBounds(90,0,400,40);
        b3.setBounds(0,50,550,80);
        add(b1);
        add(b2);
        add(b3);
    }
}
```

# Sem *layouts*



# Layouts mistos com JPanel

- Podemos usar instâncias de JPanel para agrupar componentes.
- Cada instância de JPanel pode ter seu próprio *layout*.



# Layouts mistos com JPanel

```
public class ExemploJPanel extends JApplet
{
    public void init()
    {
        setLayout(new BorderLayout());
        JPanel painel = new JPanel(new GridLayout(5,2));
        for(int i=0;i<10;i++)
            painel.add(new JButton(""+(char)('a'+i)));
        add(new JLabel("Menu"),BorderLayout.NORTH);
        add(new JLabel("Status"),BorderLayout.SOUTH);
        add(painel,BorderLayout.WEST);
        add(new JTextArea(10,30),BorderLayout.CENTER);
    }
}
```

## Outros *layouts*

- *CardLayout* permite a organização de páginas de componentes.
- *FlowLayout* monta componentes um do lado do outro, de acordo com tamanho da janela.
- *GridBagLayout* permite alinhamento preciso mas não absoluto.
- *SpringLayout* permite alinhamento de componentes relativamente a outros.

# Componentes básicos

- Muitos componentes já prontos para uso direto.
- Todos configuráveis/customizáveis quanto à aparência e funcionalidade.
- Não veremos todos.
- Primeiros exemplos mostram somente aparência; funcionalidade somente com [eventos](#).

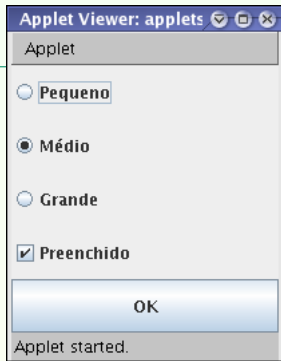


# Botões

- Servem para seleção de opções simples (sim/não) e executar ações.
- **Botões Comuns**: instâncias de `JButton`, podem conter texto e/ou ícone.
- **Botões de Rádio**: instâncias de `JRadioButton`, agrupados em `ButtonGroups`, permitem seleção de uma entre várias opções.
- **Checkboxes**: instâncias de `JCheckBox`, permitem escolher ou não uma opção.

# Botões

```
public class ExemploBotoes extends JApplet
{
    public void init()
    {
        setLayout(new GridLayout(5,1));
        JRadioButton peq =
            new JRadioButton("Pequeno",false);
        JRadioButton med =
            new JRadioButton("Médio",true);
        JRadioButton gde =
            new JRadioButton("Grande",false);
        ButtonGroup grupo = new ButtonGroup();
        grupo.add(peq); grupo.add(med); grupo.add(gde);
        JCheckBox preenchido = new JCheckBox("Preenchido",true);
        JButton ok = new JButton("OK");
        add(peq); add(med); add(gde);
        add(preenchido); add(ok);
    }
}
```

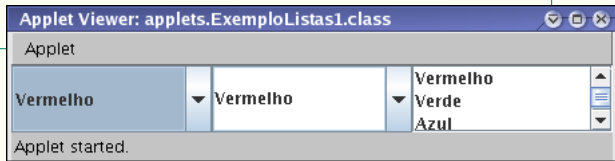


## Listas, *sliders* e *spinners*

- Servem para seleção de opções já existentes (ordenadas ou não).
- *Combo Boxes*: instâncias de `JComboBox`, listas com somente uma opção visível, pode ser editável.
- *Listas*: instâncias de `JList`, opções mostradas em uma ou mais colunas.
- *Sliders*: instâncias de `JSlider`, escolha de valores ordenados.
- *Spinners*: instâncias de `JSpinner`, escolha de valores ordenados.
- Listas e *Combo Boxes* permitem múltiplas seleções.

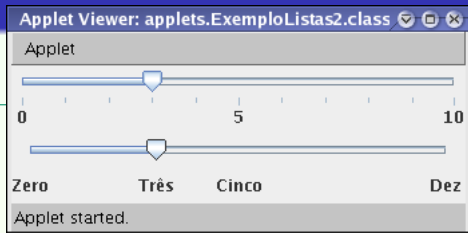
# Listas

```
public class ExemploListas extends JApplet
{
    public void init()
    {
        setLayout(new GridLayout(1,3));
        String[] opções = {"Vermelho","Verde","Azul","Amarelo"};
        JComboBox combo1 = new JComboBox(opções);
        combo1.setEditable(false);
        JComboBox combo2 = new JComboBox(opções);
        combo2.setEditable(true);
        JList list1 = new JList(opções);
        add(combo1); add(combo2); add(new JScrollPane(list1));
    }
}
```



# Sliders

```
public class ExemploSliders
    extends JApplet
    {
    public void init()
        {
        setLayout(new GridLayout(2,1));
        JSlider slider1 = new JSlider(0,10,3);
        slider1.setMajorTickSpacing(5); slider1.setMinorTickSpacing(1);
        slider1.setPaintLabels(true); slider1.setPaintTicks(true);
        JSlider slider2 = new JSlider(0,10,3);
        Hashtable<Integer,JLabel> labels = new Hashtable<Integer,JLabel>();
        labels.put(new Integer(0),new JLabel("Zero"));
        labels.put(new Integer(3),new JLabel("Três"));
        labels.put(new Integer(5),new JLabel("Cinco"));
        labels.put(new Integer(10),new JLabel("Dez"));
        slider2.setLabelTable(labels);
        slider2.setPaintLabels(true); slider2.setPaintTicks(true);
        add(slider1); add(slider2);
        }
    }
}
```

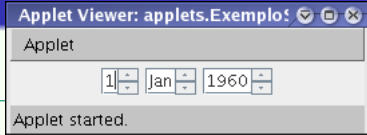


# Spinners

```

public class ExemploSpinners extends JApplet
{
    public void init()
    {
        setLayout(new FlowLayout());
        Integer[] dias = new Integer[31];
        for(int i=0;i<dias.length;i++) dias[i] = new Integer(i+1);
        SpinnerListModel modeloDia = new SpinnerListModel(dias);
        JSpinner dia = new JSpinner(modeloDia);
        String[] meses = {"Jan", "Fev", "Mar", "Abr", "Mai", "Jun",
                          "Jul", "Ago", "Set", "Out", "Nov", "Dez"};
        SpinnerListModel modeloMês = new SpinnerListModel(meses);
        JSpinner mês = new JSpinner(modeloMês);
        Integer[] anos = new Integer[105];
        for(int i=0;i<anos.length;i++) anos[i] = new Integer(i+1900);
        SpinnerListModel modeloAno = new SpinnerListModel(anos);
        JSpinner ano = new JSpinner(modeloAno);
        ano.setValue(new Integer(1960));
        add(dia); add(mês); add(ano);
    }
}

```

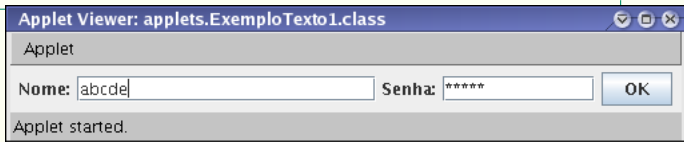


# Campos para texto

- Servem para entrada de strings e variações.
- **Campos para texto**: instâncias de `JTextField`, somente uma linha.
- **Campos para senhas**: instâncias de `JPasswordField`, não mostra caracteres digitados.
- **Áreas para texto**: instâncias de `JTextArea`, várias linhas.
- **Labels**: instâncias de `JLabel`, texto não modificável.

# Campos para texto

```
public class ExemploTexto1 extends JApplet
{
    public void init()
    {
        setLayout(new FlowLayout());
        JLabel l1 = new JLabel("Nome:");
        JTextField n = new JTextField(20);
        JLabel l2 = new JLabel("Senha:");
        JPasswordField p = new JPasswordField(10);
        JButton b = new JButton("OK");
        add(l1); add(n); add(l2); add(p); add(b);
    }
}
```





# Campos para texto

```
public class ExemploTexto2 extends JApplet
{
    public void init()
    {
        setLayout(new BorderLayout());
        JLabel l1 = new JLabel("Digite algo abaixo");
        JTextArea ta = new JTextArea(5,50);
        JButton b = new JButton("OK");
        add(l1,BorderLayout.NORTH);
        add(new JScrollPane(ta),BorderLayout.CENTER);
        add(b,BorderLayout.SOUTH);
    }
}
```



# Desenvolvendo componentes específicos

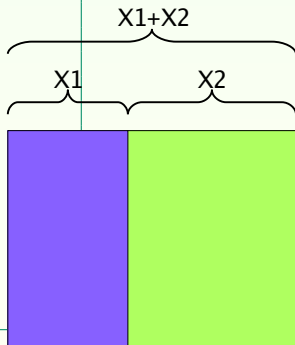
- É simples desenvolver alguns tipos de componentes específicos:
  - Escreva uma classe que herde de `JComponent`.
  - Implemente o construtor da classe para passar informações para suas instâncias.
  - Sobreescreva o método `paintComponent` para desenhar o componente.
- Sobreescrever outros métodos pode ser interessante, dependendo da aplicação.

## Desenvolvendo componentes específicos

- Esta receita básica serve para componentes que exibem dados ou informações.
- Componentes que aceitam entrada de dados interativa são mais complexos.
- Basta usar componente na *applet*, sem se preocupar com método *paint* da *applet*.

# Desenvolvendo componentes específicos

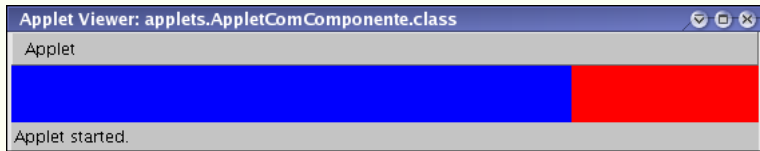
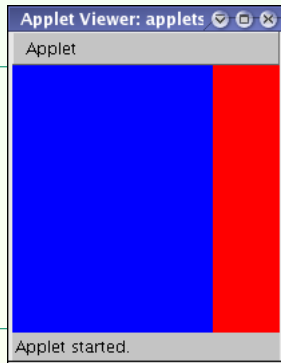
```
public class ComponenteSimples extends JComponent
{
    private int x1,x2;
    public ComponenteSimples(int x1,int x2)
    {
        this.x1 = x1; this.x2 = x2;
    }
    protected void paintComponent(Graphics g)
    {
        float w = getWidth()*x1/(x1+x2);
        g.setColor(Color.BLUE);
    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



# Desenvolvendo componentes específicos

```
public class AppletComComponente extends JApplet
{
    private ComponenteSimples c;

    public void init()
    {
        c = new ComponenteSimples(75,25);
        add(c);
    }
}
```

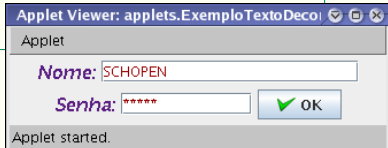


## Mais sobre componentes...

- Componentes podem ter atributos modificados.
  - Fontes.
  - Cores de frente e fundo.
- Botões (e outros componentes) podem conter ícones!
- *Labels* (e outros componentes) podem conter HTML!

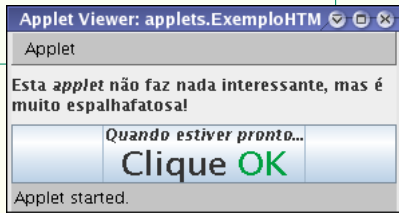
# Mais sobre componentes...

```
public class ExemploTextoDecorado extends JApplet
{
    public void init()
    {
        setLayout(new FlowLayout());
        Font fontLabels = new Font("Arial", Font.ITALIC|Font.BOLD, 16);
        Color colorLabels = new Color(90,30,130);
        Color colorInputs = new Color(150,0,0);
        JLabel l1 = new JLabel("Nome:");
        l1.setFont(fontLabels); l1.setForeground(colorLabels);
        JTextField n = new JTextField(20); n.setForeground(colorInputs);
        JLabel l2 = new JLabel("Senha:");
        l2.setFont(fontLabels); l2.setForeground(colorLabels);
        JPasswordField p = new JPasswordField(10);
        p.setForeground(colorInputs);
        ImageIcon iconOK = new ImageIcon("ok.png");
        JButton b = new JButton("OK",iconOK);
        add(l1); add(n); add(l2); add(p); add(b);
    }
}
```



# Mais sobre componentes...

```
public class ExemploHTMLComponentes extends JApplet
{
    public void init()
    {
        setLayout(new GridLayout(2,1));
        JLabel l =
            new JLabel("<html>Esta <i>applet</i> não faz nada interessante,"+
                " mas é <b>muito</b> espalhafatosa!</html>");
        JButton b =
            new JButton("<html><center><i>Quando estiver pronto...</i><br>"+
                "<font size=+2>Clique <font color=\\\"#00A040\\\">OK"+
                "</font></center></html>");
        add(l); add(b);
    }
}
```





## Applets até agora...

- Como desenhar uma interface com usuário?  $\Rightarrow$  *layouts* e componentes.
- **Como reagir à interação com usuário?**  $\Rightarrow$  *programação com eventos*.
- Basicamente devemos:
  - Implementar interface correspondente a um *listener*.
  - Criar componentes e registrar neles *listeners* para a própria classe.
  - Implementar métodos requeridos pela interface.
- Os métodos requeridos serão executados automaticamente quando o evento ocorrer.

# Eventos

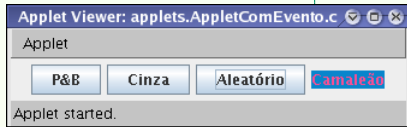
- Diversos tipos de eventos: ação, ajuste, mudança, seleção de opções, clique e movimento de mouse, etc.
- Alguns tipos de eventos não são aplicáveis a *applets*: modificação de tamanho de janelas, minimização/maximização, etc.
- Cada tipo de componente pode “escutar” um ou mais tipos de eventos.
- Primeiro exemplo básico: um botão que ao ser clicado modifica uma cor em um *label* na *applet*.

# Exemplo básico de evento

```
public class AppletComEvento extends JApplet implements ActionListener
{
    private JButton pb,cinza,aleat;
    private JLabel label;
    public void init()
    {
        setLayout(new FlowLayout());
        pb = new JButton("P&B"); pb.addActionListener(this);
        cinza = new JButton("Cinza"); cinza.addActionListener(this);
        aleat = new JButton("Aleatório"); aleat.addActionListener(this);
        label = new JLabel("Camaleão"); label.setOpaque(true);
        add(pb); add(cinza); add(aleat); add(label);
    }
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == pb)
        {
            label.setForeground(Color.WHITE);
            label.setBackground(Color.BLACK);
        }
    }
}
```

# Exemplo básico de evento

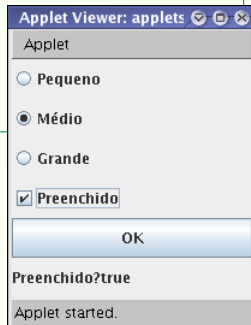
```
if (e.getSource() == cinza)
{
    label.setForeground(new Color(192,192,192));
    label.setBackground(new Color(64,64,64));
}
if (e.getSource() == aleat)
{
    int r1 = (int)(Math.random()*256);
    int g1 = (int)(Math.random()*256);
    int b1 = (int)(Math.random()*256);
    label.setForeground(new Color(r1,g1,b1));
    int r2 = (int)(Math.random()*256);
    int g2 = (int)(Math.random()*256);
    int b2 = (int)(Math.random()*256);
    label.setBackground(new Color(r2,g2,b2));
}
}
```



# Eventos para botões e *checkboxes*

- ActionListener para JButton e JCheckBox, ItemListener para JRadioButton

```
public class ExemploEventoBotoes extends JApplet
    implements ItemListener, ActionListener
{
    private JRadioButton peq,med,gde;
    private JCheckBox preenchido;
    private JButton ok;
    private JLabel status;
```



# Eventos para botões e *checkboxes*

```
public void init()
{
    setLayout(new GridLayout(6,1));
    peq = new JRadioButton("Pequeno", false);
    peq.addItemListener(this);
    med = new JRadioButton("Médio", true);
    med.addItemListener(this);
    gde = new JRadioButton("Grande", false);
    gde.addItemListener(this);
    ButtonGroup grupo = new ButtonGroup();
    grupo.add(peq);    grupo.add(med);    grupo.add(gde);
    preenchido = new JCheckBox("Preenchido", true);
    preenchido.addActionListener(this);
    ok = new JButton("OK");
    ok.addActionListener(this);
    status = new JLabel("");
    add(peq); add(med); add(gde);
    add(preenchido); add(ok); add(status);
}
```

# Eventos para botões e *checkboxes*

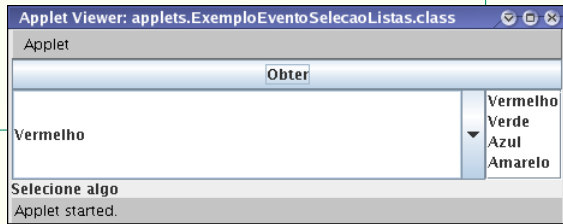
```
public void itemStateChanged(ItemEvent e)
{
    if (peq.isSelected()) status.setText("Pequeno");
    if (med.isSelected()) status.setText("Médio");
    if (gde.isSelected()) status.setText("Grande");
}

public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == ok) status.setText("Botão OK");
    if (e.getSource() == preenchido)
        status.setText("Preenchido?" +preenchido.isSelected());
}
}
```

# Eventos de seleção em listas

- ListSelectionListener para JLists, ItemListener ou ActionListener para JComboBoxes.

```
public class ExemploEventoSelecaoListas extends JApplet
    implements ListSelectionListener,
               ItemListener,
               ActionListener
{
    private JComboBox combo;
    private JList list;
    private JButton obter;
    private JLabel label;
}
```





# Eventos de seleção em listas

```
public void init()
{
    String[] opções = {"Vermelho", "Verde", "Azul", "Amarelo"};
    combo = new JComboBox(opções);
    combo.setEditable(true);
    combo.addActionListener(this); combo.addItemListener(this);
    list = new JList(opções); list.addListSelectionListener(this);
    obter = new JButton("Obter"); obter.addActionListener(this);
    label = new JLabel("Selecione algo");
    add(combo, BorderLayout.CENTER);
    add(new JScrollPane(list), BorderLayout.EAST);
    add(obter, BorderLayout.NORTH);
    add(label, BorderLayout.SOUTH);
}

public void valueChanged(ListSelectionEvent e)
{
    label.setText("Lista: selecionou "+list.getSelectedValue());
}
```

# Eventos de seleção em listas

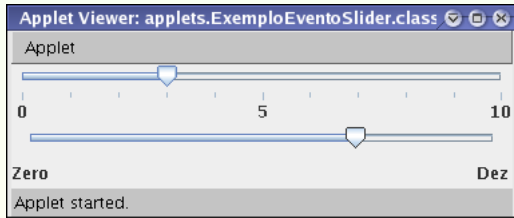
```
public void itemStateChanged(ItemEvent e)
{
    label.setText("Combo box: selecionou "+combo.getSelectedItemAt());
}

public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == combo)
        label.setText("Combo box: selecionou novo "+
                      combo.getSelectedItemAt());
    if (e.getSource() == obter)
        label.setText("Lista: selecionou "+list.getSelectedValue()+" "+
                      "Combo box: selecionou "+combo.getSelectedItemAt());
}
}
```

# Eventos de seleção em *sliders*

- ChangeListener para JSliders.

```
public class ExemploEventoSlider extends JApplet
                                implements ChangeListener
{
    private JSlider slider1, slider2;
```

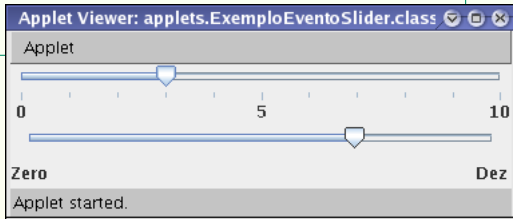


# Eventos de seleção em *sliders*

```
public void init()
{
    setLayout(new GridLayout(2,1));
    slider1 = new JSlider(0,10,3);
    slider1.setMajorTickSpacing(5);
    slider1.setMinorTickSpacing(1);
    slider1.setPaintLabels(true); slider1.setPaintTicks(true);
    slider1.addChangeListener(this);
    slider2 = new JSlider(0,10,7);
    Hashtable<Integer,JLabel> labels = new Hashtable<Integer,JLabel>();
    labels.put(new Integer(0),new JLabel("Zero"));
    labels.put(new Integer(10),new JLabel("Dez"));
    slider2.setLabelTable(labels);
    slider2.setPaintLabels(true); slider2.setPaintTicks(true);
    slider2.addChangeListener(this);
    add(slider1); add(slider2);
}
```

# Eventos de seleção em *sliders*

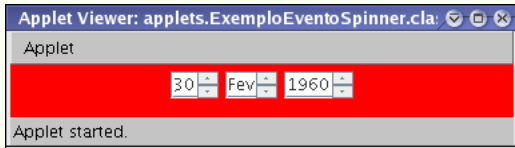
```
public void stateChanged(ChangeEvent e)
{
    JSlider source = (JSlider)e.getSource();
    if (source.getValueIsAdjusting()) return;
    if (e.getSource() == slider1)
        slider2.setValue(10-slider1.getValue());
    if (e.getSource() == slider2)
        slider1.setValue(10-slider2.getValue());
}
}
```



# Eventos de seleção em *spinners*

- `ChangeListener` para `JSpinners`.

```
public class ExemploEventoSpinner extends JApplet
    implements ChangeListener
{
    private JSpinner dia,mês,ano;
    private String[] meses = {"Jan","Fev","Mar","Abr","Mai","Jun",
        "Jul","Ago","Set","Out","Nov","Dez"};
```

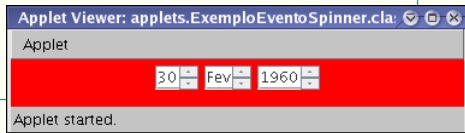


# Eventos de seleção em *spinners*

```
public void init()
{
    setLayout(new FlowLayout());
    Integer[] dias = new Integer[31];
    for(int i=0;i<dias.length;i++) dias[i] = new Integer(i+1);
    SpinnerListModel modeloDia = new SpinnerListModel(dias);
    dia = new JSpinner(modeloDia);
    dia.addChangeListener(this); add(dia);
    SpinnerListModel modeloMês = new SpinnerListModel(meses);
    mês = new JSpinner(modeloMês);
    mês.addChangeListener(this); add(mês);
    Integer[] anos = new Integer[105];
    for(int i=0;i<anos.length;i++) anos[i] = new Integer(i+1900);
    SpinnerListModel modeloAno = new SpinnerListModel(anos);
    ano = new JSpinner(modeloAno);
    ano.setValue(new Integer(1960));
    ano.addChangeListener(this); add(ano);
}
```

# Eventos de seleção em *spinners*

```
public void stateChanged(ChangeEvent e)
{
    int dd = (Integer)(dia.getValue());
    String tempm = (String)mês.getValue();
    int mm = 0;
    for(mm=0;mm<12;mm++) if (tempm.equals(meses[mm])) break;
    int aa = (Integer)ano.getValue();
    Calendar c = Calendar.getInstance();
    c.clear(); c.setLenient(false);
    c.set(Calendar.DAY_OF_MONTH,dd); c.set(Calendar.MONTH,mm);
    c.set(Calendar.YEAR,aa);
    try { c.getTime(); }
    catch (Exception exc)
    {
        getContentPane().setBackground(Color.RED);
        repaint();
    }
}
```





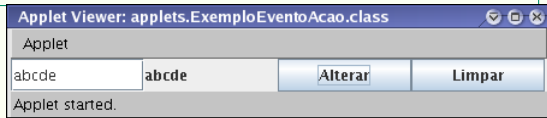
# Eventos para campos de texto

- ActionListener para campos de texto (acionados quando Enter for pressionado).

```
public class ExemploEventoAcao extends JApplet
                                implements ActionListener
{
    private JTextField entrada;
    private JLabel saída;
    private JButton alterar,limpar;
    public void init()
    {
        setLayout(new GridLayout(1,4));
        entrada = new JTextField(15); entrada.addActionListener(this);
        saída = new JLabel();
        alterar = new JButton("Alterar"); alterar.addActionListener(this);
        limpar = new JButton("Limpar"); limpar.addActionListener(this);
        add(entrada); add(saída); add(alterar); add(limpar);
    }
}
```

# Eventos para campos de texto

```
public void actionPerformed(ActionEvent e)
{
    if ((e.getSource() == entrada) || (e.getSource() == alterar))
        saída.setText(entrada.getText());
    if (e.getSource() == limpar)
    {
        entrada.setText("");
        saída.setText("");
    }
}
}
```



# Eventos para movimentos e cliques do mouse

- `MouseListener` para quando os botões do mouse forem clicados em um componente.
- `MouseMotionListener` para quando o mouse for movimentado sobre um componente.
- `MouseWheelListener` para quando o botão rotatório do mouse for usado sobre um componente.

# Eventos para movimentos e cliques do mouse

Receita básica para criação de componentes que usam eventos de mouse em *applets*:

- 1 Criar uma classe que herde de `JComponent` e implemente `MouseListener` e/ou `MouseMotionListener` e/ou `MouseWheelListener` de acordo com a necessidade.
- 2 Escrever código para os métodos necessários (nem todos precisam executar código!)
- 3 Ao criar instância deste componente na *applet*, registrar o *listener* de eventos usando a própria instância.

# Eventos para movimentos e cliques do mouse

- Exemplo simples: *applet* para rabiscos.
- Componente faz o processamento, *applet* somente exibe componente.
- Guardamos posições dos pontos em uma lista quando o mouse é pressionado ou arrastado (métodos `mousePressed` e `mouseDragged`).
- Pintamos os pontos guardados na lista com o método `paintComponent`.

# Eventos para movimentos e cliques do mouse

```
public class ComponenteDesenhoSimples extends JComponent
    implements MouseListener, MouseMotionListener
{
    private ArrayList<Point> pontos;
    private int size = 8; private int halFSIZE = size/2;
    public ComponenteDesenhoSimples()
    {
        pontos = new ArrayList<Point>(1024);
    }
    protected void paintComponent(Graphics g)
    {
        Graphics2D g2d = (Graphics2D)g;
        g2d.setColor(Color.WHITE);
        g2d.fillRect(0,0,getWidth(),getHeight());
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setColor(Color.BLUE);
        for(Point p:pontos)
        {
            g2d.fillOval(p.x-halFSIZE,p.y-halFSIZE,size,size);
        }
    }
}
```

# Eventos para movimentos e cliques do mouse

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

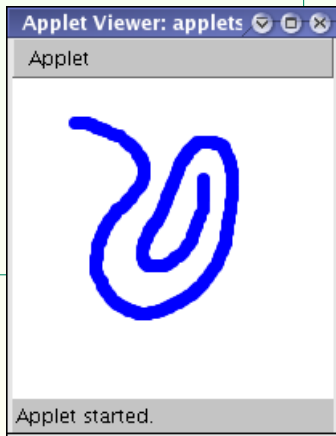
public void mouseDragged(MouseEvent e)
{
    pontos.add(e.getPoint());
    repaint();
}

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

# Eventos para movimentos e cliques do mouse

```
public class AppletComComponenteDesenhoSimples extends JApplet
{
    private ComponenteDesenhoSimples c;

    public void init()
    {
        c = new ComponenteDesenhoSimples();
        c.addMouseListener(c);
        c.addMouseMotionListener(c);
        add(c);
    }
}
```





# Introdução

- Como fazer com que algo ocorra independente de ações do usuário?
- *Threads* ou linhas de execução: execução concorrentemente com execução da *applet*.
- *Timers*: eventos disparados de tempos em tempos.

# Threads

- *Threads* permitem que parte de código de uma classe seja executado juntamente com a classe principal.
- Para uso com *applets*, precisamos implementar nos componentes a interface `Runnable` e criar uma *Thread* para executar a instância.
- Exemplo básico (receita de bolo):

```
public class ComponenteRunnable extends JComponent
                                implements Runnable
{
    private int x = 0;
    private Thread tt;
```

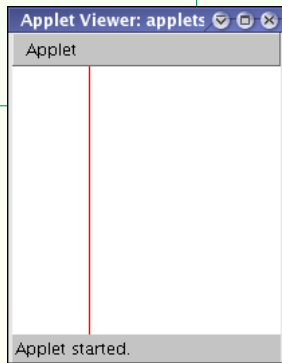
# Threads

```
public ComponenteRunnable()  
{  
    tt = new Thread(this);  
    tt.start();  
}  
  
protected void paintComponent(Graphics g)  
{  
    g.setColor(Color.WHITE); g.fillRect(0,0,getWidth(),getHeight());  
    g.setColor(Color.RED);    g.drawLine(x,0,x,getHeight());  
}  
  
public void run()  
{  
    Thread t = Thread.currentThread();  
    while(t==tt)  
    {  
        x++; if (x > getWidth()) x = 0;  
        repaint();  
    }  
}  
}
```

# Threads

```
public class AppletComComponenteRunnable extends JApplet
{
    public void init()
    {
        ComponenteRunnable c = new ComponenteRunnable();
        add(c);
    }
}
```

Devemos considerar métodos  
start e stop da *applet*!



# Exemplo mais complexo

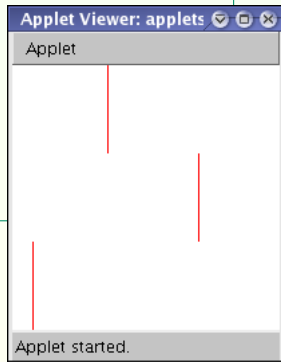
```
public class ComponenteRunnable2 extends JComponent
                                implements Runnable
{
    private int x = 0;
    private long pause;
    private Thread tt;
    public ComponenteRunnable2(long p)
    {
        tt = new Thread(this);
        tt.start();
        pause = p;
    }
    protected void paintComponent(Graphics g)
    {
        g.setColor(Color.WHITE);
        g.fillRect(0,0,getWidth(),getHeight());
        g.setColor(Color.RED);
        g.drawLine(x,0,x,getHeight());
    }
}
```

# Exemplo mais complexo

```
public void run()
{
    Thread t = Thread.currentThread();
    while(t==tt)
    {
        x++;
        if (x > getWidth()) x = 0;
        repaint();
        try
        {
            Thread.sleep(pause);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

# Exemplo mais complexo

```
public class AppletComComponenteRunnable2 extends JApplet
{
    public void init()
    {
        setLayout(new GridLayout(3,1));
        ComponenteRunnable2 c1,c2,c3;
        c1 = new ComponenteRunnable2(10);
        c2 = new ComponenteRunnable2(100);
        c3 = new ComponenteRunnable2(1000);
        add(c1); add(c2); add(c3);
    }
}
```



# Timers

- *Timers* registram um evento de ação que ocorrerá cada vez que o valor do temporizador for alcançado.
- Criamos uma classe que implementa `ActionListener`.
- Criamos uma instância de `javax.swing.Timer` e a registramos com a classe.
- Cada vez que o temporizador for disparado, o método `actionPerformed` será executado.
- Exemplo básico (receita de bolo):

```
public class ComponenteComTimer extends JComponent
                                implements ActionListener
{
    private Color c;
    private Timer timer;
```



# Timers

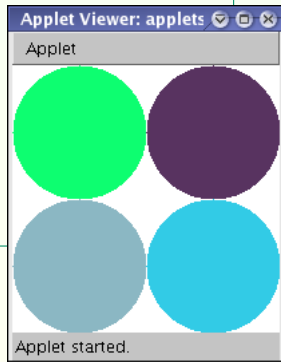
```
public ComponenteComTimer(int delay)
{
    timer = new Timer(delay, this);
    timer.setCoalesce(true);
    timer.start();
}

protected void paintComponent(Graphics g)
{
    g.setColor(Color.WHITE);
    g.fillRect(0,0,getWidth(),getHeight());
    g.setColor(c);
    g.fillArc(0,0,getWidth(),getHeight(),0,360);
}

public void actionPerformed(ActionEvent e)
{
    int r = (int)(Math.random()*256);
    int g = (int)(Math.random()*256);
    int b = (int)(Math.random()*256);
    c = new Color(r,g,b);
    repaint();
}
}
```

# Timers


```
public class AppletComComponenteComTimer extends JApplet
{
    public void init()
    {
        setLayout(new GridLayout(2,2));
        ComponenteComTimer c1,c2,c3,c4;
        c1 = new ComponenteComTimer(10);
        c2 = new ComponenteComTimer(50);
        c3 = new ComponenteComTimer(250);
        c4 = new ComponenteComTimer(1000);
        add(c1); add(c2); add(c3); add(c4);
    }
}
```



# Introdução

- Sabemos como montar o *layout* de uma *applet*.
- Sabemos como reagir a eventos para adicionar funcionalidade às *applets*.
- *Applets* servem como clientes ricos para a web: apresentação é importante!
  - Podemos modificar aparência de alguns componentes.
  - Como fazer desenhos mais complexos em componentes de exibição? ⇒ *Java2D*.
  - Foco será em desenhar componentes específicos.

# Graphics2D

- Cada classe que herda de `JComponent` pode sobrescrever método `paintComponent`.
- Método `paintComponent` recebe como argumento uma instância de `Graphics` que é o contexto gráfico onde desenharemos.
- Podemos fazer um `cast` de `Graphics` para `Graphics2D` para maior funcionalidade.
- Exemplo: `ComponenteSimples` 

# Cores

- Mudamos cores em um contexto gráfico com método `setColor`.
- Argumento deve ser uma instância de `Color`.
- Exemplos:
  - `g2d.setColor(Color.BLACK)` (BLACK, BLUE, CYAN, DARK\_GRAY, GRAY, GREEN, LIGHT\_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW).
  - `g2d.setColor(new Color(10,80,200))`.
  - `g2d.setColor(new Color(10,80,200))`.
  - `g2d.setColor(new Color(10,80,200,120))`.
  - `g2d.setColor(new Color(0xff00ff))`.

# Desenhando formas básicas

- Métodos drawXXX para Arc, Image, Oval, Line, Polygon, Rect, etc.
- Métodos fillXXX para alguns destes.
- Métodos drawShape e fillShape para instâncias de classes que herdam de Shape: Arc2D, Ellipse2D, Line2D, Rectangle2D, RoundRectangle2D, etc.

# Desenhando formas básicas

```
public class ComponenteCarnavalesco extends JComponent
                                   implements ActionListener
{
    private Timer timer;
    public ComponenteCarnavalesco()
    {
        timer = new Timer(100,this);
        timer.setCoalesce(true);
        timer.start();
    }
    protected void paintComponent(Graphics g)
    {
        Graphics2D g2d = (Graphics2D)g;
        int rr = (int)(Math.random()*256);
        int gg = (int)(Math.random()*256);
        int bb = (int)(Math.random()*256);
        int aa = (int)(Math.random()*256);
        g2d.setColor(new Color(rr,gg,bb,aa));
    }
}
```

# Desenhando formas básicas

```
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,  
    RenderingHints.VALUE_ANTIALIAS_ON);  
g2d.setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,  
    RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);  
Shape shape = null;  
int x = (int)(Math.random()*getWidth());  
int y = (int)(Math.random()*getHeight());  
int w = (int)(Math.random()*getWidth());  
int h = (int)(Math.random()*getHeight());
```



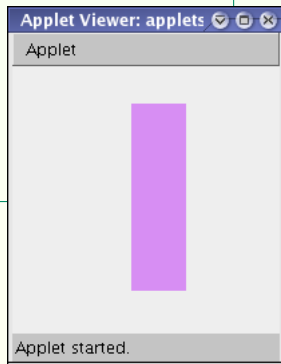
# Desenhando formas básicas

```
switch((int)(Math.random()*4))
{
  case 0:
    {
      shape = new Ellipse2D.Float(x,y,w,h);           break;
    }
  case 1:
    {
      shape = new Rectangle2D.Float(x,y,w,h);         break;
    }
  case 2:
    {
      shape = new RoundRectangle2D.Float(x,y,w,h,2,2); break;
    }
  case 3:
    {
      shape = new RoundRectangle2D.Float(x,y,w,h,15,15); break;
    }
}
g2d.fill(shape);
}
```

# Desenhando formas básicas

```
public void actionPerformed(ActionEvent e)
{
    repaint();
}
```

```
public class AppletComComponenteCarnavalesco extends JApplet
{
    private ComponenteCarnavalesco c;
    public void init()
    {
        c = new ComponenteCarnavalesco();
        add(c);
    }
}
```

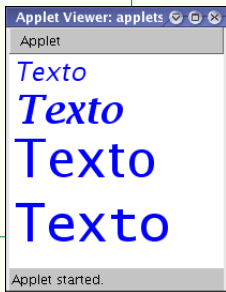


# Texto

- Usamos o método `drawString` da classe `Graphics2D`.
- Podemos criar instâncias de `Font` para fontes, estilos e tamanhos diferentes.
- Veremos somente as fontes lógicas: `Serif`, `SansSerif`, `Monospaced`, `Dialog`, `DialogInput`.

# Texto

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



# Imagens e *Sprites*

- Podemos usar `ImageIcon` para obter imagens do disco.
- Imagens podem ser transparentes.
- Método `drawImage` da classe `Graphics2D` pode ser usado para desenhar a imagem:
  - Instância de `Image`, obtida com `ImageIcon.getImage()`.
  - `X` e `Y` para desenhar a imagem (origem é canto superior esquerdo).
  - Instância de `ImageObserver`: pode ser `null`.

# Imagens e *Sprites*

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

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

# Imagens e *Sprites*

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

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

# Imagens e *Sprites*

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



# Imagens e *Sprites*

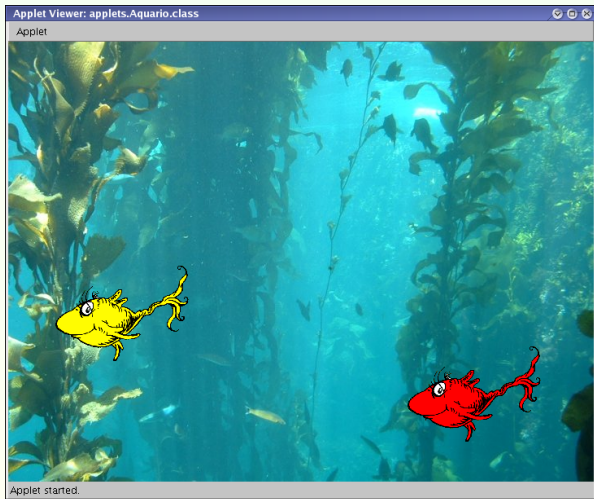
```
protected void paintComponent(Graphics g)
{
    Graphics2D g2d = (Graphics2D)g;
    g2d.drawImage(background.getImage(),0,0,null);
    vermelho.paint(g2d);
    amarelo.paint(g2d);
}
public void run()
{
    Thread t = Thread.currentThread();
    while(t==tt)
    {
        vermelho.move(); amarelo.move(); repaint();
        try
        {
            Thread.sleep(100);
        }
        catch (InterruptedException e)
        {
        }
    }
}
}
```

# Imagens e *Sprites*

```
public class Aquario extends JApplet
{
    private ComponenteAquario c;

    public void init()
    {
        c = new ComponenteAquario();
        add(c);
        resize(720,540);
    }
}
```

# Imagens e *Sprites*



# Restrições que afetam entrada e saída

Uma *applet* tem algumas restrições de segurança:

- Uma *applet* não pode ler ou escrever arquivos localizados no cliente.
- Uma *applet* não pode fazer conexões de rede exceto para o servidor de onde foi carregada.
- Uma *applet* não pode iniciar novos programas no cliente.

É importante observar que *applets* executadas com *appletviewer* são consideradas seguras!

## Lendo dados em *applets*

Para fazer leitura de um arquivo em uma *applet*, precisamos:

- 1 Ter certeza de que o arquivo é visível na Internet e que vem do mesmo servidor da *applet*.
- 2 Criar uma instância de URL com o endereço desejado.
- 3 A partir da URL, obter uma instância de `InputStream` com `openStream()`.
- 4 A partir da instância de `InputStream` construir uma instância de `InputStreamReader`.
- 5 A partir da `InputStreamReader` criar uma instância de `BufferedReader`.
- 6 Ler strings do servidor.

# Lendo dados em *applets*

```
public class AppletIO extends JApplet
{
    private JLabel read1,read2;

    public void init()
    {
        setLayout(new GridLayout(2,1));
        String s1 = readLineFrom("http://localhost:8080/rafael-jug");
        JLabel l1 = new JLabel(s1);
        add(l1);
        String s2 = readLineFrom("http://www.cnn.com");
        JLabel l2 = new JLabel(s2);
        add(l2);
    }
}
```

# Lendo dados em *applets*

```
private String readLineFrom(String url)
{
    String texto = null;
    try
    {
        URL u = new URL(url);
        BufferedReader reader =
            new BufferedReader(new InputStreamReader(u.openStream()));
        texto = reader.readLine();
    }
    catch (MalformedURLException e) { texto = "Erro na URL."; }
    catch (AccessControlException e) { texto = "Acesso inválido."; }
    catch (IOException e) { texto = "Erro de I/O."; }
    if (texto == null) texto = "ERRO!!!";
    return texto;
}
}
```

# Lendo dados em *applets*

Applet Viewer: applets.AppletIO.class

Applet

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"><html lang="en"><head><title>CNN.com</title> ...
```

Applet started.

Entrada e Saída - Mozilla Firefox

File Edit View Go Bookmarks Tools Help



file:///home/rafael/Java/Misc,



Java Docs Webmail Utils News Fun Research Game Info Home Pages

## Entrada e Saída

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

Acesso inválido.

Applet applets.AppletIO started



# Conexão com Bancos de Dados

- Além dos conceitos de conexão, temos o problema do *classloader*.
  - Exemplo,

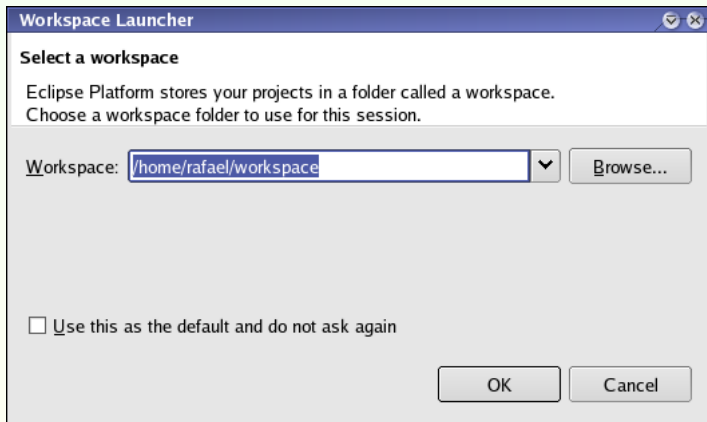
```
Class.forName("oracle.jdbc.driver.OracleDriver");
```
- É necessário estabelecer uma **política de acesso** para a *applet*.
- Conceito avançado, não veremos neste curso.

# Eclipse

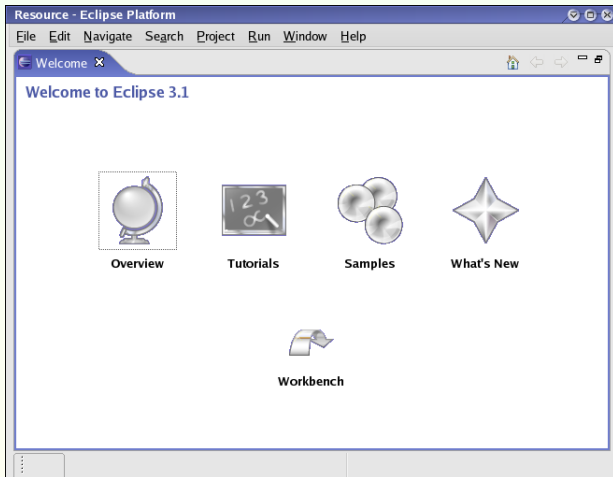
- Ferramenta sugerida para desenvolvimento.
- Gratuita, disponível para várias plataformas, extensível.
- [www.eclipse.org](http://www.eclipse.org).

Mostro aqui apenas um tutorial básico do Eclipse.  
Tutoriais mais avançados existem na Internet.

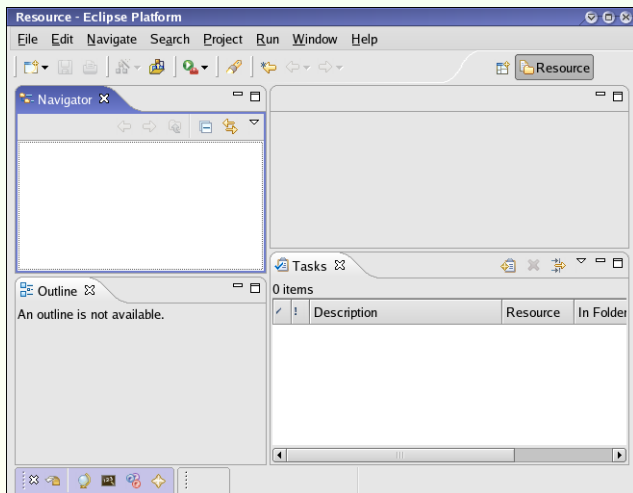
# Eclipse: Seleccionando *Workspace*



# Eclipse: Selecionar *Workbench*



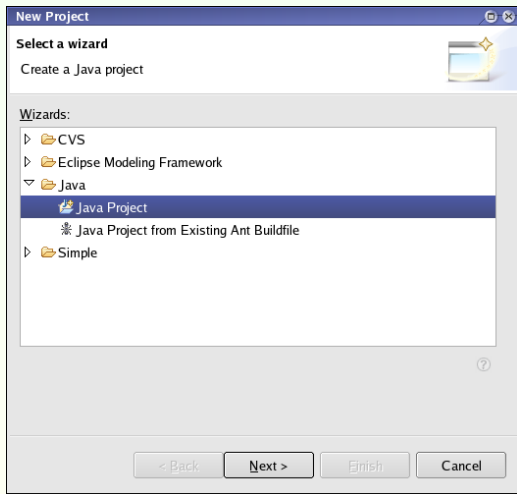
# Eclipse: Aparência Inicial



# Eclipse: Criando um Projeto

Criando um novo projeto: File → New → Project.

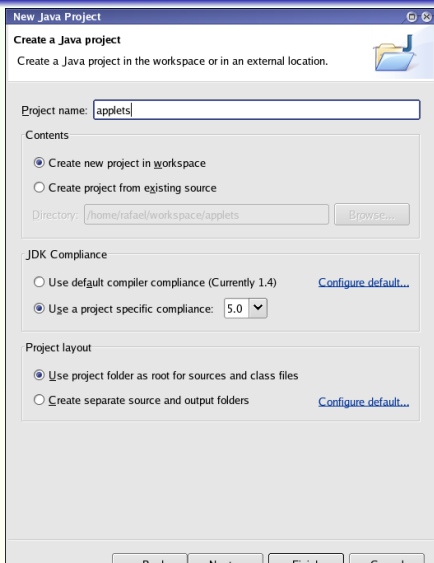
Escolher Java → Java Project.



# Eclipse: Criando um Projeto

Escolher um nome para o projeto, usar *project specific compliance 5.0* se for adequado.

Clicar em *Finish*, clicar *Yes* para diálogo *Open Associated Perspective*.

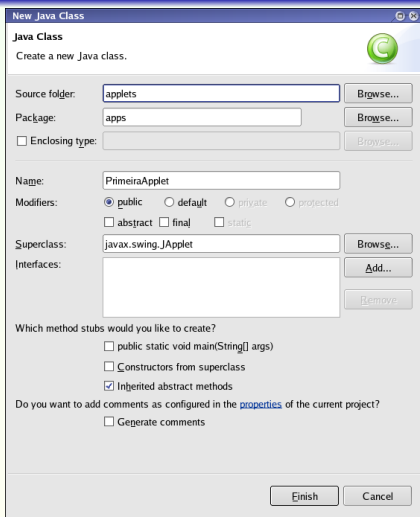


# Eclipse: Criando uma Classe

Clicar em File →  
New → Class.

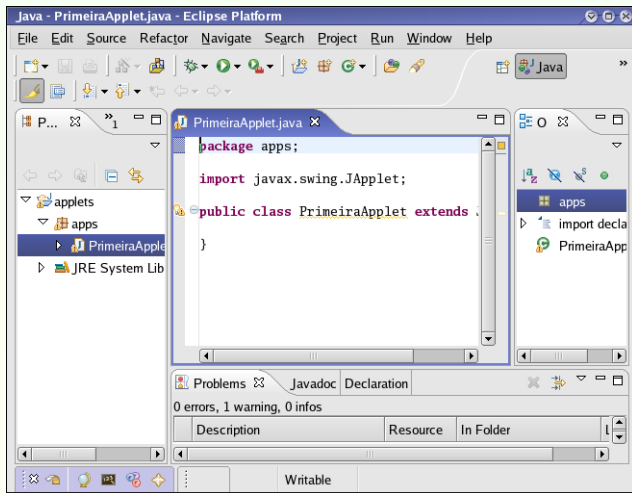
Preencher dados da  
classe, em especial  
Package, Name e  
Superclass.

Clicar em Finish.





# Eclipse: Ambiente com Nova Classe



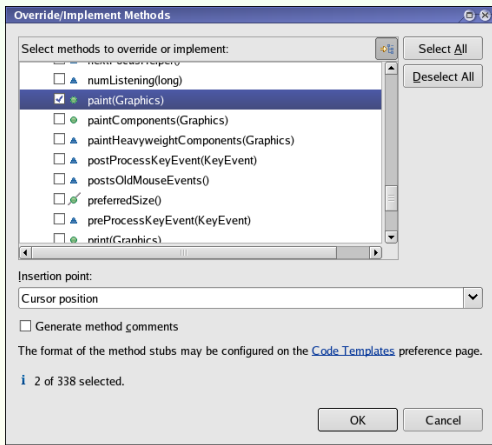
# Eclipse: Escrevendo Métodos

Clicar com botão direito  
no editor, escolher  
Source →  
Override/Implement  
Methods.

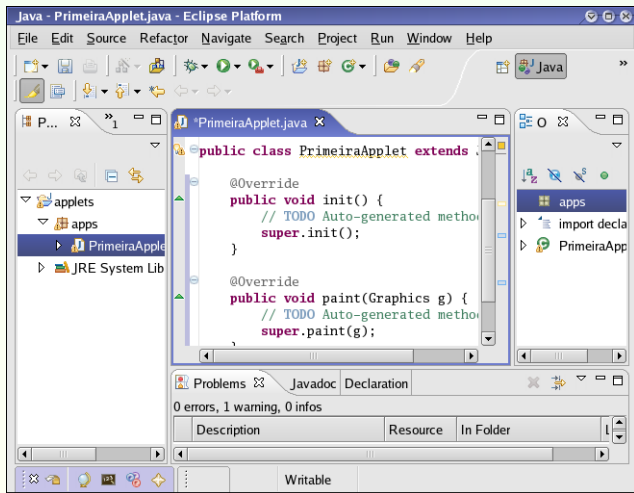
Selecionar:

- init de Applet
- paint de Container

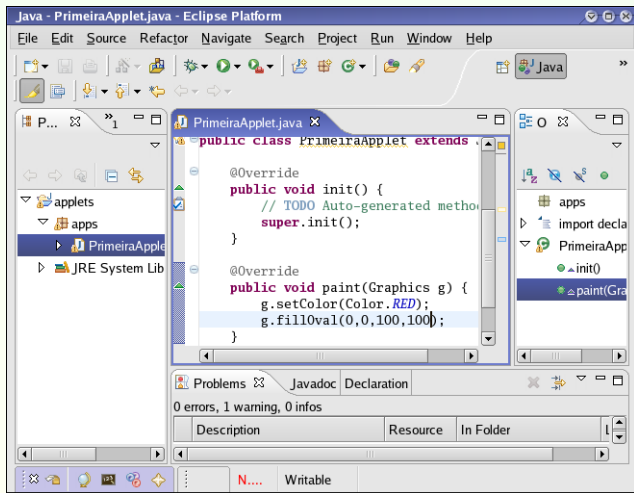
Clicar em OK.



# Eclipse: Classe com Mais Métodos

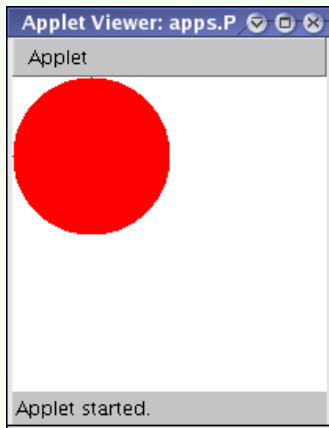


# Eclipse: Classe com Mais Métodos (Reescritos)



# Eclipse: Executando a *Applet*

No menu, selecionar Run →  
Run As → Java Applet.



# Deployment

- Vamos ver como fazer o *deployment* completo de uma *applet* em um arquivo *jar*.
- Arquivos *jar*: contém classes e arquivos adicionais que podem ser usados pela *applet*.
- Exemplo: aquário.
- Condições:
  - Assumimos que as classes não estão em pacotes (isto é, não tem declaração *package*).
  - Assumimos que os arquivos *.class* e os *resources* (imagens) estão em um mesmo diretório.

# Deployment

- Faremos ligeiras modificações nas classes.
- Imagens não podem ser carregadas com `new ImageIcon(nomeDaImagem)`: devem ser carregadas via uma URL.
- Para demonstrar parâmetros para *applets*, peixes terão velocidades passadas como parâmetros.

# Deployment: aquário modificado

```
public class ComponentePeixe
{
    private ImageIcon fish;
    private int x,y; // posição
    private int wf,hf; // tamanho peixe
    private int wa,ha; // tamanho aquário
    private float speed;
    public ComponentePeixe(String filename,Dimension aq,float s)
    {
        URL iconURL = getClass().getResource(filename);
        fish = new ImageIcon(iconURL);
        wf = fish.getIconWidth(); hf = fish.getIconHeight();
        wa = aq.width; ha = aq.height;
        speed = s;
        x = (int)(Math.random()*wa);
        y = (int)(Math.random()*(ha-hf));
    }
}
```



# Deployment: aquário modificado

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

```
public class ComponenteAquario extends JComponent
                                implements Runnable
{
    private ComponentePeixe vermelho;
    private ComponentePeixe amarelo;
    private ImageIcon background;
    private Thread tt;
```

# Deployment: aquário modificado

```
public ComponenteAquario(float vv,float va)
{
    URL imageURL = getClass().getResource("aquarium_bg.jpg");
    background = new ImageIcon(imageURL);
    Dimension area = new Dimension(background.getIconWidth(),
                                    background.getIconHeight());
    vermelho = new ComponentePeixe("red_fish.png",area,vv);
    amarelo = new ComponentePeixe("yellow_fish.png",area,va);
    tt = new Thread(this);
    tt.start();
}
protected void paintComponent(Graphics g)
{
    Graphics2D g2d = (Graphics2D)g;
    g2d.drawImage(background.getImage(),0,0,null);
    vermelho.paint(g2d);
    amarelo.paint(g2d);
}
```

# Deployment: aquário modificado

```
public void run()
{
    Thread t = Thread.currentThread();
    while(t==tt)
    {
        vermelho.move();
        amarelo.move();
        repaint();
        try
        {
            Thread.sleep(100);
        }
        catch (InterruptedException e)
        {
        }
    }
}
```

# Deployment: aquário modificado

```
public class Aquario extends JApplet
{
    private ComponenteAquario c;
    public void init()
    {
        float vv = getValor("velVermelho");
        float va = getValor("velAmarelo");
        c = new ComponenteAquario(vv,va);
        add(c);
        resize(720,540);
    }
}
```

# Deployment: aquário modificado

```
private float getValor(String parâmetro)
{
    float valor = 1;
    String val = getParameter(parâmetro);
    if (val != null)
        try
        {
            valor = Float.parseFloat(val);
        }
        catch (NumberFormatException e)
        {
        }
    return valor;
}
```

## Deployment: criando o arquivo .jar

Assumimos que os arquivos .class e os *resources* (imagens) estão em um mesmo diretório:

```
jar cvf aquario.jar Aquario.class ComponenteAquario.class
      ComponentePeixe.class aquarium_bg.jpg red_fish.png
      yellow_fish.png

adding: Aquario.class(in = 439) (out= 307)(deflated 30%)
adding: ComponenteAquario.class(in = 1673) (out= 986)(deflated 41%)
adding: ComponentePeixe.class(in = 1304) (out= 757)(deflated 41%)
adding: aquarium_bg.jpg(in = 80273) (out= 79502)(deflated 0%)
adding: red_fish.png(in = 3412) (out= 3368)(deflated 1%)
adding: yellow_fish.png(in = 4047) (out= 4017)(deflated 0%)

ls -la aquario.jar
-rw-rw-r-- 1 rafael rafael 90256 Sep 13 18:11 aquario.jar
```

# Deployment: criando o arquivo .html

```
<html>
<head><title>Aquário Virtual</title></head>
<body bgcolor="#a0e8ff">
<center>
<h3>Aquário Virtual em Java</h3>
<APPLET CODE="Aquario.class" ARCHIVE="aquario.jar"
          WIDTH=720 HEIGHT=540>
<PARAM NAME=velAmarelo VALUE=2.5>
<PARAM NAME=velVermelho VALUE=6.5>
Se você está lendo isso é porque seu navegador não reconhece
<i>applets</i>!
</APPLET>
</center>
</body>
</html>
```

# Deployment: abrindo o arquivo .html

