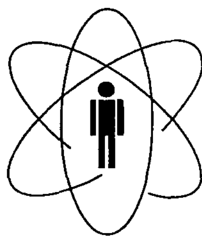
ISSN 0101-9201



CBPF - CENTRO BRASILEIRO DE PESQUISAS FÍSICAS
Rio de Janeiro

Notas Técnicas

CBPF-NT-006/00
agosto 2000

Introdução à Linguagem Java

Deyse M. Peixoto, Marcelo Portes de Albuquerque
e Márcio Portes de Albuquerque

CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico

Sumário

1. INTRODUÇÃO 2

2. LINGUAGEM ORIENTADA A OBJETOS 3

2.1 CARACTERÍSTICAS DA LINGUAGEM *JAVA* 3
2.2 CONCEITOS DA ORIENTAÇÃO A OBJETOS 3
2.3 CLASSE 4

3. APLICAÇÕES E APPLETS 5

3.1 APPLETS 5
3.3 COMPARANDO A EXECUÇÃO DE APLICAÇÕES E APPLETS............................ 7

4. ESCREVENDO EM ARQUIVOS 8

5. MANIPULANDO STRING 10

6. INTERFACE GRÁFICA 14

7. ANIMAÇÃO... 17

7.1 DOUBLE BUFFERING 18

8. CONCLUSÃO 21

9. BIBLIOGRAFIA... 22

1. Introdução

A linguagem *Java* foi desenvolvida pela Sun MicroSystems há aproximadamente seis anos atrás, objetivando-se aplicações voltadas para produtos eletrônicos de grande consumo, tais como televisões, videocassetes e outros eletrodomésticos. A escolha desse ramo não obteve o sucesso desejado, no entanto, com a popularização da Internet e suas páginas Web, surgia uma nova e interessante aplicação para a linguagem -as A*pplets*- proporcionando animação e interatividade aos até então estáticos documentos HTML. As possibilidades para artes gráficas, multimídia e interação passaram a ser maiores, pois a linguagem *Java* permite desenhar figuras, fazer cálculos, enviar mensagem ao usuário, exibir janelas e gráficos, etc.

Java é uma linguagem que herdou muitas de suas características do C++ e implementa o paradigma da Programação Orientada a Objetos. É uma linguagem interpretada, o que a torna independente da plataforma, ou seja, um mesmo programa pode ser executado em qualquer sistema que possua seu interpretador.

Em uma rede com vários computadores diferentes, esta independência de arquitetura mostra seu valor. O formato da arquitetura de *Java* concede sólidos benefícios tanto ao cliente quanto ao desenvolvedor: você compra uma vez e o executa em qualquer máquina; escreve uma vez e vende para qualquer plataforma.

Neste documento serão apresentados, de um modo prático, os conceitos básicos da Programação Orientada a Objetos, fornecendo exemplos para melhor utilização da linguagem *Java*.

2. Linguagem Orientada a Objetos

Java é uma linguagem orientada a objetos, onde seu grande segredo é a simplicidade. Para entendê-la, precisamos conhecer os conceitos da Programação Orientada a Objetos (POO). Antes porém, serão apresentadas algumas características da linguagem *Java*.

2.1 Características da Linguagem *Java*

i) Segura: um programa em *Java* não pode ler ou escrever arquivos locais quando é chamado em um carregador de classes seguro, como um browser Web e nem usar a memória além do permitido.

ii) Simples: *Java* tem uma sintaxe muito simples que permite o usuário programar facilmente de forma clara e orientada a objetos.

iii) Robusta: *Java* tem por finalidade a criação de programas que sejam confiáveis, eliminando situações de erro.

iv) Multitarefa: em um mesmo programa podemos ter vários processos rodando de forma concorrente.

v) Universal e Interpretada: *Java* é universal, pois independe da plataforma. Seu código é compilado para o processador virtual (*Java Virtual Machine*) e transformado em uma seqüência de instruções chamada *bytecode.* Essas instruções são interpretadas para o processador real da máquina. A figura 2.1 ajuda a entender como acontece a interpretação do código *Java*.

Para mais detalhes sobre cada uma das características citadas acima, veja [Fla97a, Fla97b].

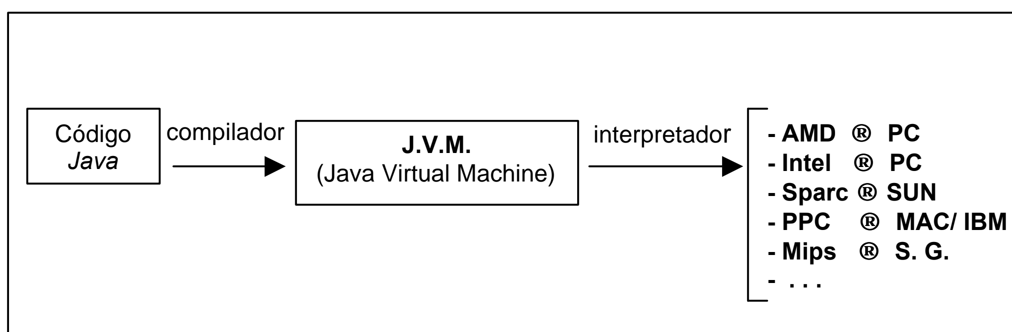


Figura 2.1: interpretação do código *Java* - O código *Java* é transformado em uma seqüência de instruções que são interpretadas para o processador real da máquina.

2.2 Conceitos da Orientação a Objetos

O conceito de orientação a objetos é o ponto central da linguagem *Java*. É importante entender como estes conceitos são traduzidos na programação. Depois de se familiarizar com este conceito, você estará pronto para começar a programar em *Java*.

2.3 Classe

No mundo real, temos muitos objetos de mesmo tipo. Por exemplo: seu automóvel é apenas um, dos vários automóveis existentes no mundo. Em POO, dizemos que seu automóvel é uma instância da classe de objetos conhecida como automóvel, ou seja, o objeto criado a partir de uma classe é uma instância dessa classe.

As fábricas de automóveis tiram vantagem do fato que automóveis compartilham características. Estas podem fabricar muitos automóveis de mesmo molde, pois seria muito ineficiente produzir um novo molde para cada automóvel fabricado.

Em POO, também é possível ter muitos objetos de mesmo tipo: botões têm as mesmas características, assim como as janelas, etc. Semelhante a fabricação de automóveis, podemos levar vantagem no fato de que objetos de mesmo tipo são parecidos e assim criarmos um molde para cada um desses objetos. Um “molde”, que define as propriedades e métodos comuns para todos os objetos criados, é chamado classe (figura 2.2).

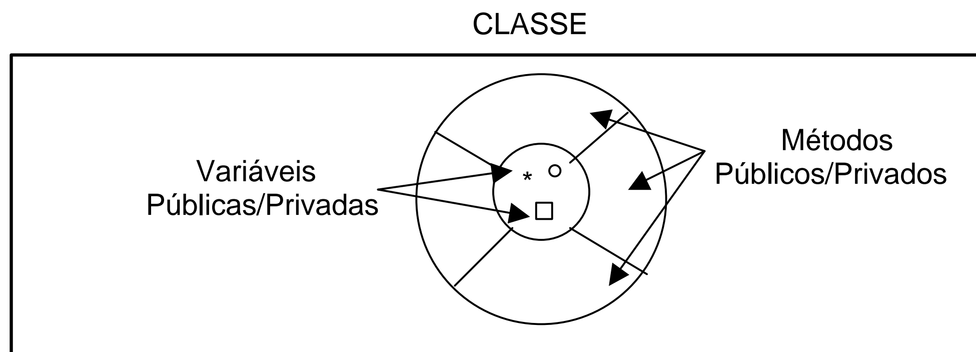


Figura 2.2: classe é um molde que define as propriedades (variáveis) e métodos comuns para todos os objetos criados.

Podemos criar uma classe automóvel que possua variáveis de instância e contenha características comuns como: acelerador, marcha, freio, etc. A classe também fornece implementações para os métodos, permitindo a alteração das variáveis: velocidade, trocar de marcha, intensidade do freio, etc.

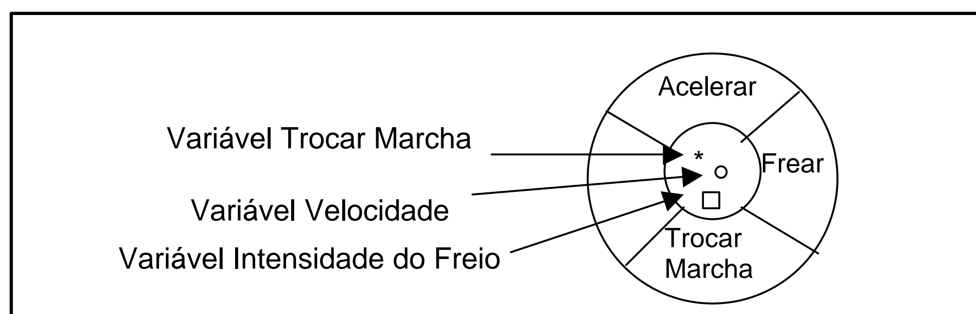


Figura 2.3: no exemplo do automóvel cada variável só pode ser modificada pelo método correspondente, já os métodos podem ser acessados por outros métodos.

No exemplo acima cada variável tem uma classe correspondente, assim podemos chamar o método que pertencente a essa classe para alterar os valores das variáveis. As variáveis da figura 2.3 são variáveis privadas, isto é, só podem ser modificadas pelos métodos da classe que ela pertence. Já os métodos são públicos, isto é, podem ser acessados por outros métodos externos àclasse.

3. Aplicações e Applets

Existem duas maneiras diferentes de rodar um programa em *Java*: como uma Applet ou uma Aplicação.

3.1 Applets

São programas em *Java* que rodam dentro de um browser. As applets são consideradas seguras, dinâmicas, independente da plataforma e rodam em ambiente de rede. Elas podem reagir àentrada do usuário tornando as páginas Web mais dinâmicas.

A figura 3.1 apresenta o código do programa onde será escrita a frase `"Hello Cat!"` na tela. Primeiramente precisamos importar duas bibliotecas: `java.awt` (*AWT-abstract window toolkit*), sendo responsável pelos recursos gráficos do programa e `java.applet`, que contém os recursos necessários para a construção de uma applet.

O nome da classe deve ser o mesmo do arquivo .java, que neste exemplo é **`hello`**. Para iniciar o programa as applets utilizam o método "init". O comando `System.out.println("Hello Cat!")` escreve a frase na applet .

```
------------------------------------------------------------------------------------------------------
//Programa  hello.java :                                                              (a)
//Obs : Escreve a frase 'Hello Cat !' na Applet
------------------------------------------------------------------------------------------------------
import java.awt.*;
import java.applet.*;
  public class hello extends Applet {
       public void paint(Graphics g) {
              g.drawString("Hello Cat!");
       }
  }
```

```
------------------------------------------------------------------------------------------------------
//Arquivo  hello.html :                                                               (b)
//Obs : Disponibiliza o programa hello.java na Internet
------------------------------------------------------------------------------------------------------
<html>
<APPLET CODE="hello.class" width=300 height=300>
</APPLET>
</html>
```

Figura 3.1: (a) applet que escreve na tela a frase `"Hello Cat!"`. (b) disponibiliza o arquivo na Internet.

3.2 Aplicações

São programas em *Java* que rodam independente do browser. A figura 3.2 mostra o código de um programa onde também será escrita a frase "Hello Cat!". Para iniciar o programa utilizamos o método "main". O comando System.out.println("Hello Cat!") escreve a frase.

```
---------------------------------------------------------------------------------------------------------
//Programa  aplica.java :
//Obs : Escreve a frase 'Hello Cat !'
---------------------------------------------------------------------------------------------------------
import java.awt.*;
public class aplica {
      public static void main(String args[]) {
            System.out.println("Hello Cat!");
         }
}
```

Figura 3.2: aplicação que escreve a frase "Hello Cat!".

A figura 3.3 mostra o procedimento para compilarmos uma aplicação, a criação do arquivo .class e a execução, mostrando a frase "Hello Cat!".

Figura 3.3: compilação e execução de uma aplicação. Na primeira linha temos a complilação do arquivo .java, logo depois a criação do arquivo .class e por fim a chamada ao interpretador mostrando a frase "Hello Cat!".

3.3 Comparando a execução de Aplicações e Applets

As aplicações e as applets diferem nos privilégios de execução que possuem e também na maneira como iniciar a execução. A figura 3.3 analisa as duas execuções.

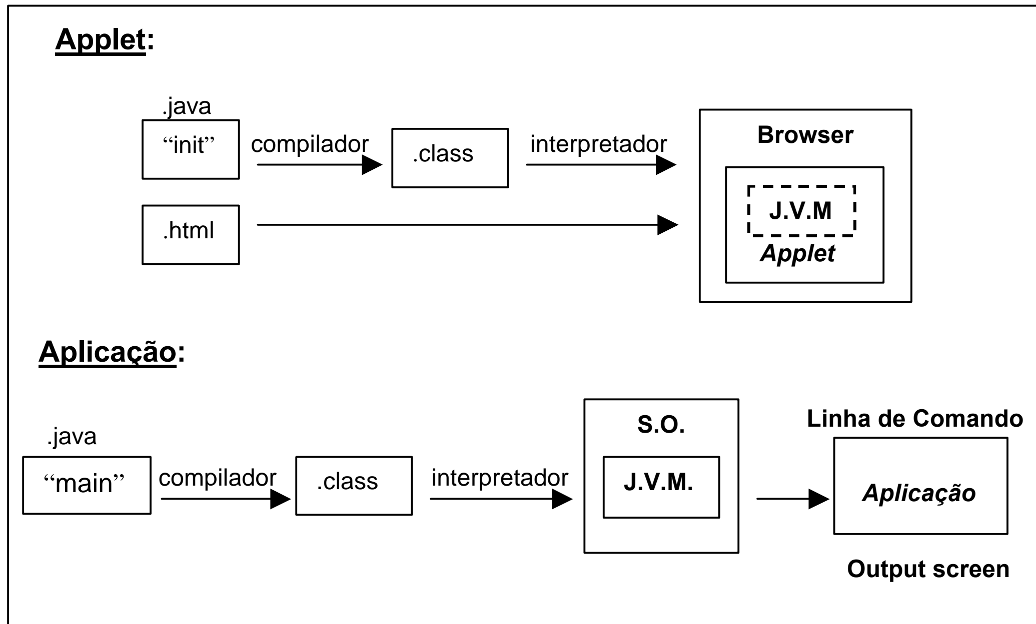


Figura 3.3: compara a execução de uma Applet e uma Aplicação. No caso de uma applet, após a compilação do código .java é criado o arquivo .class. Criamos o arquivo .html e chamamos o interpretador para que a applet possa ser vista. No caso de uma aplicação, o código também é compilado criando assim o arquivo .class, chamamos o interpretador e através de uma linha de comando do sistema operacional a aplicação poderá ser vista.

4. Escrevendo em Arquivos

Neste capítulo veremos como manipular arquivos em *Java*. A figura 4.1 mostra o código de um programa onde um arquivo texto é criado, aberto e em seguida escrevemos os ângulos de 0° a 360° num intervalo de 45° e os respectivos cossenos.

```
-----------------------------------------------------------------------------------------------------------------------------
//Programa  escreve_arq.java :
//Obs : Calcula o cosseno dos ângulos de 0°  a  360° num intervalo de 45° e escreve o
// resultado em um arquivo texto criado.
-------------------------------------------------------------------------------
import java.applet.*;
import java.io.*;

  public class escreve_arq extends Applet {
   String FileName, str; //declaração das variáveis
   double ang, ResCos;

   public void init() {
//--------------------- método init------------------------------------------
    FileName = new String("cos");
    File to_file = new File(FileName+".txt"); // criação do arquivo txt
    try {
      FileOutputStream  to  =  new  FileOutputStream(to_file  );  //  fluxo  de  saída
      DataOutputStream dt = new DataOutputStream(to);// de dados para o arquivo

        for (int x  = 0; x <= 360; x = x + 45){
           ang = Math.PI * x /180;
           ResCos=Math.cos(ang);
           str ="Ang:"+ang*180/Math.PI+"-> Cos:"+ResCos+"\n";
           dt.writeBytes(str); // escreve no arquivo
         }
      }
      catch(IOException exc){
         System.out.println("Erro ao abrir ou escrever no arquivo\n");
       }
    }
  }

//--------------------End of file escreve_arq.java ----------------------------
```

Figura 4.1: manipulação de arquivo. Cria um arquivo txt, abre e escreve os ângulos de 0° a 360° num intervalo de 45° e o respectivo cosseno.

Esta applet chama-se **`calc1`** e é uma extensão da classe `Applet`. Inicialmente temos as declarações das variáveis usadas e logo depois criamos e nomeamos o arquivo texto como `"cos"`.

```
FileName = new String("cos");
File to_file = new File(FileName+".txt");
```

No bloco **try** está uma parte do código que deve estar protegida contra todas as exceções (erros). Uma exceção é uma condição anormal que ocorre em uma seqüência de código em tempo de execução, como por exemplo: divisão por zero, arquivo não encontrado, etc. Para mais detalhes sobre tratamento de exceções, veja [Fla97a, Fla97b]. Neste caso temos o fluxo de saída de dados para o arquivo:

```
FileOutputStream to = new FileOutputStream(to_file );
DataOutputStream dt = new DataOutputStream(to);
```

Logo após segue um loop de 0° a 360° com intervalo de 45°, onde são calculados os ângulos e os respectivos cossenos. Por fim escrevemos no arquivo o ângulo e o correspondente cosseno, com o comando:

```
dt.writeBytes(str);
```

Imediatamente após um bloco **try**, incluímos uma cláusula **catch** que especifica o tipo de exceção que se deseja capturar. Neste caso se acontecer algum erro o usuário receberá uma mensagem.

```
catch(IOException exc){
   System.out.println("Erro ao abrir ou escrever no arquivo\n");
}
```

5. Manipulando String

Uma string é uma seqüência de caracteres. Como strings formam a estrutura de dados básica das palavras, *Java* tem diversos recursos incorporados que facilitam o tratamento de strings e também possui uma biblioteca padrão chamada: `java.lang` que está presente em todos os programas. Existe uma classe embutida no pacote `java.lang` com nome de *`String`*.

`String` usa outra classe embutida, chamada `StringBuffer`, para ajudá-la na sua manipulação. `StringBuffer` difere de `String` pelo fato de oferecer suporte às seqüências de caracteres que podem ser anexadas e inseridas. Ela não tem qualquer suporte para pesquisa de caracteres individuais ou substring. Usaremos a classe `String` com mais freqüência do que `StringBuffer`.

A seguir são apresentados alguns métodos importantes da classe `String` usados para manipulação de strings:

<table>
<tr>
<td>length</td>
<td>
É um método usado em uma <code>String</code> para retornar o número de caracteres da string. Este trecho de código imprimirá 3, uma vez que há três caracteres na string, s.
<pre>String s = “cat’;
System.out.println(s.length());</pre>
O interessante em Java é que uma instância de objeto é criada para todos os literais <code>String</code>, assim podemos chamar os métodos diretamente em uma string citada como se ela fosse uma referência ao objeto. Este exemplo imprimirá 3 novamente.
<pre>System.out.println(“cat”.length());</pre>
</td>
</tr>
</table>

Usando os métodos a seguir pode-se modificar uma `String` fazendo uma cópia dela em um `StringBuffer` construindo uma cópia nova da `String` com suas modificações realizadas.

substring	É usado para extrair um trecho de uma `String`, fazendo uma cópia distinta de todos os caracteres que forem solicitados ao método. Pode-se especificar só o índice inicial, o que retornará uma cópia daquele caractere e do restante da `String`, ou pode-se especificar o caractere inicial e final, obtendo assim uma `String` contendo todos os caracteres do intervalo.
	
	`"Hello Cat".substring(6)→ "Cat"`
	`"Hello Cat".substring(3,8)→ "lo Ca"`

concat	A concatenação de strings, pelo método `concat` de `String`, simplesmente cria um objeto novo com o conteúdo atual de sua `String` com os caracteres do parâmetro `String` anexados no final. `"Hello".concat("Cat")→ "Hello Cat"`
replace	O método replace toma dois caracteres como parâmetros. Todas as ocorrências do primeiro caractere na `String` são substituídas pela segunda. `"Hello".replace('l','w')→ "Hewwo"`

toLowerCase
 	
e
 	
toUpperCase
	A seguir temos um par de métodos, toLowerCase e toUpperCase que convertem todos os caracteres de uma `String` de maiúsculas para minúsculas e vice-versa.
	
	`"Hello".toLowerCase() → "hello"`
	`"Hello".toUpperCase() → "HELLO"`
	
	Finalmente, o método `trim` retorna uma cópia da `String` sem nenhum espaço em branco inicial ou final.
	
	`"   Hello Cat     ".trim()→ "Hello Cat"`

Para comparação de `Strings` usamos os seguintes métodos:

equals	É usado para comparar duas strings, sabendo se elas são iguais. Ele vai retornar `true` se os caracteres das duas strings forem exatamente os mesmos. Uma forma alternativa de `equals` chamada `equalsIgnoreCase` ignora as maiúsculas e minúsculas de cada caractere das strings na comparação. Considerando A-Z sendo a mesma coisa que a-z.

	`class equalsDemo {`
	`public static void main(String args[]) {`
	`String s1= "Hello";`
	`String s2= "HELLO";`
	`String s3= "Hello";`
	`System.out.println(s1+"equals"+s2+"→"+s1.equals(s2));`
	`System.out.println(s1+"equals"+s3+"→"+s1.equals(s3));`
	`System.out.println(s1+"equalsIgnoreCase"+s2+"→"+s1.equalsIgnoreCase(s2));`
	`}`
	`}`
	
	O resultado deste exemplo mostra que `equals` e `equalsIgnoreCase` funcionam conforme o esperado.
	
	C:\> Java equalsDemo
	`Hello equals HELLO → false`
	`Hello equals Hello → true`
	`Hello equalsIgnoreCase HELLO → true`

regionMatches	A classe `String` expõe alguns métodos utilitários que são versões especializadas de `equals`. O método `regionMatches` é usado para comparar uma região específica dentro de uma `String`, com outra região específica em outra `String`. Há uma opção para ignorar as maiúsculas e minúsculas em tais comparações. Isto permitiria fazer comparações sofisticadas entre strings de origem arbitrária e padrão.
	
	Há duas variantes de `regionMatches`. Uma permite controlar se as maiúsculas e minúsculas são ou não significativas; a outra considera-se que são significativas. Ambas permitem sintonia sobre onde uma comparação deve iniciar-se na `String` de origem.
	
	A seguir temos o protótipo de função desses dois métodos:
	
	`boolean regionMatches(int toffset, String other, int ooffset, int len);`
	
	`boolean regionMatches(Boolean ignoreCase, int toofset, String other, int ooffset, int len);`
	
	Em ambas as versões de `regionMatches`, o parâmetro `toffset` indica o deslocamento de caracteres para o objeto `String` no qual estamos chamando o método. A `String` com a qual estamos comparando chama-se `other` e o deslocamento para aquela `String` chama-se `ooffset`. Os caracteres `len` das duas strings são comparados começando nos dois deslocamentos.

<table>
<tr><td>startsWith

e

endsWith</td><td>Algumas rotinas de conveniência também são formas especializadas de <code>equals</code>. O método <code>startsWith</code> verifica se determinada <code>String</code> começa com a <code>String</code> passada. Da mesma forma, há um método chamado <code>endsWith</code>, que verifica se a <code>String</code> em questão termina exatamente com os mesmos caracteres do parâmetro.

<code>"Foobar".endsWith("bar")</code> e <code>"Foobar".startsWith ("Foo")</code> são ambas <code>true</code>. É possível especificar onde se deve começar a procurar dentro de uma string, passando um segundo parâmetro inteiro, por exemplo <code>"Foobar".startsWith("bar",3)</code> é <code>true.</code></td></tr>
<tr><td>IndexOf

e

lastIndexOf</td><td>A classe <code>String</code> suporta a busca de determinado caractere ou substring usando os métodos <code>indexOf e lastIndexOf</code>. Estes dois métodos são sobrecarregados de maneiras diferentes para suportar a busca de caracteres ou substrings e para iniciar nos deslocamentos diferentes do início e final da <code>String</code> de origem. Cada um destes métodos retorna o índice do caractere que você está procurando ou do índice do início da substring que você está procurando. Em todos os casos se a busca não teve sucesso, esses métodos retornam – 1.

Exemplos:

<code>int indexOf(int ch)</code>
<code>int lastIndexOf(int ch)</code>

Retorna o índice da primeira(última) ocorrência do caractere <code>ch</code>.

<code>int indexOf(String str)</code>
<code>int lastIndexOf(String str)</code>

Retorna o índice do primeiro caractere da primeira(última) ocorrência da substring <code>str</code>.

<code>int indexOf(int ch, int fromIndex)</code>
<code>int lastIndexOf(int ch, int fromIndex)</code>

Retorna o índice da primeira ocorrência depois(antes) <code>fromIndex</code> do caractere <code>ch</code>.

<code>int indexOf(String str, int fromIndex)</code>
<code>int lastIndexOf(String str, int fromIndex)</code>

Retorna o índice do primeiro caractere da primeira ocorrência depois(antes) de <code>fromIndex</code> da substring <code>str</code>.</td></tr>
</table>

6. Interface Gráfica

A interface gráfica permite ao usuário trocar informações com o programa de uma forma mais dinâmica.

A Figura 6.1 mostra o código de um programa que permite o usuário modificar o ângulo em um quadrante e selecionar a função trigonométrica que será calculada.

```
-------------------------------------------------------------------------------------------
//Programa  interg.java :
//Obs : O usuário modifica o ângulo em um quadrante com clique do mouse e seleciona
//a função trigonométrica que será calculada.-
-------------------------------------------------------------------------------------------
import java.awt.*;
import java.applet.*;

  public class interg extends Applet {
       //----------------- declaração das variáveis --------------------------
       double dx, dy;
       String NaN;
       double ang,a,b,s1,s2,s3;
       int redesenha = 0;
       int nx,ny,xref, yref,mx,my;
       Label l1,l2;
       TextField t1,t2,t3,t4;
       Checkbox cb1,cb2,cb3;
    //----------------------- método init ----------------------------------
       public void init() {
          setLayout(null);
          cb1 = new Checkbox("Sin");
          cb2 = new Checkbox("Cos");
          cb3 = new Checkbox("Tan");
          l1 = new Label("Ângulo:");
          l2 =new Label("INTERFACE GRÁFICA-Cálculo de Sin-Cos-Tan");
          t1 = new TextField(); t2 = new TextField();
          t3 = new TextField(); t4 = new TextField();
          add(cb1); add(cb2); add(cb3);
          add(l1); add(l2);
          add(t1); add(t2);
          add(t3); add(t4);
          cb1.reshape(200,60,50,30);
          cb2.reshape(200,110,50,30);
          cb3.reshape(200,160,50,30);
          l1.reshape(50,160,60,20);
          l2.reshape(45,20,400,10);
          t1.reshape(112,160,30,20);
          t2.reshape(250,65,85,20);
          t3.reshape(250,115,85,20);
          t4.reshape(250,165,85,20);
          xref=50; yref=150;
          mx=xref+50;my=yref;
          repaint();
       }
```

```
//--------------------------- método paint -----------------------------------
      public void paint(Graphics g) {
          g.setColor(Color.red);
          g.drawLine(xref,yref,nx+xref,yref-ny);
          g.fillOval(xref+nx-2,yref-ny-2,5,5);
          g.setColor(Color.black);
          g.drawLine(50,150,150,150);
          g.drawLine(50,50,50,150);
          g.drawArc(-50,50,200,200,0,90);

          if(redesenha == 1){
              g.setColor(Color.white);
              g.drawLine(xref,yref,mx+xref,yref-my);
              redesenha = 0;
              mx = nx; my = ny;

          }
      }

//--------------------------- método mouseDown --------------------------------
      public boolean mouseDown(Event e, int x, int y) {

          a = Math.abs(x - 50);
          b = Math.abs(y - 150);
          ang = (Math.atan2( b, a)*180/Math.PI); //cálculo do ângulo

          Double iz = new Double(ang);
          t1.setText(iz.toString());

          dx = Math.cos(ang*Math.PI/180)*100;
          dy = Math.sin(ang*Math.PI/180)*100;
          ny = (int)dy;
          nx = (int)dx;

          if (cb1.getState() == true) {
              double s = dy/100;
              iz = new Double(s);
              t2.setText(iz.toString());
           }
          if (cb2.getState() == true) {
              double s1 = dx/100;
              Double iz1 = new Double(s1);
              t3.setText(iz1.toString());
           }
          if (cb3.getState() == true) {
              double s2 = dy/dx;
              Double iz2 = new Double(s2);
              t4.setText(iz2.toString());
           }
            repaint();

            redesenha=1;
            return true;
       }

 }

//--------------------- End of file intergraf.java------------------------------
```

Figura 6.1: interface gráfica, onde o usuário seleciona o ângulo e a função trigonométrica que será calculada.

A applet chama-se `intergraf`. No método `init` criamos e posicionamos os “widgets”(botões, TextField, Checkbox...) com a seqüência de instruções:

```
cb1 = new Checkbox("Sin"); // cria o Checkbox1
l1 = new Label("Ângulo:"); // cria o Label1
t1 = new TextField();// cria o TextField1 (.....)

add(cb1); // adiciona o Checkbox1
add(l1); // adiciona o Label1
add(t1); // adiciona o TextField1 (.....)

cb1.reshape(200,60,50,30); // posiciona o Checkbox1
l1.reshape(50,160,60,20); // posiciona o Label1
t1.reshape(112,160,30,20); // posiciona o TextField1 (.....)
```

As variáveis (mx, my) servem para posicionarmos o segmento no quadrante. O `repaint` é usado para chamar o método `paint` que “desenha” qualquer objeto que tenha uma representação gráfica (Button, TextField, Label ...) ou quando precisa aparecer na tela.

No método `paint` desenhamos os “widgets”, o quadrante e o segmento na posição inicial. A Figura 6.2 mostra a posição dos “widgets” e do quadrante após o `paint`.

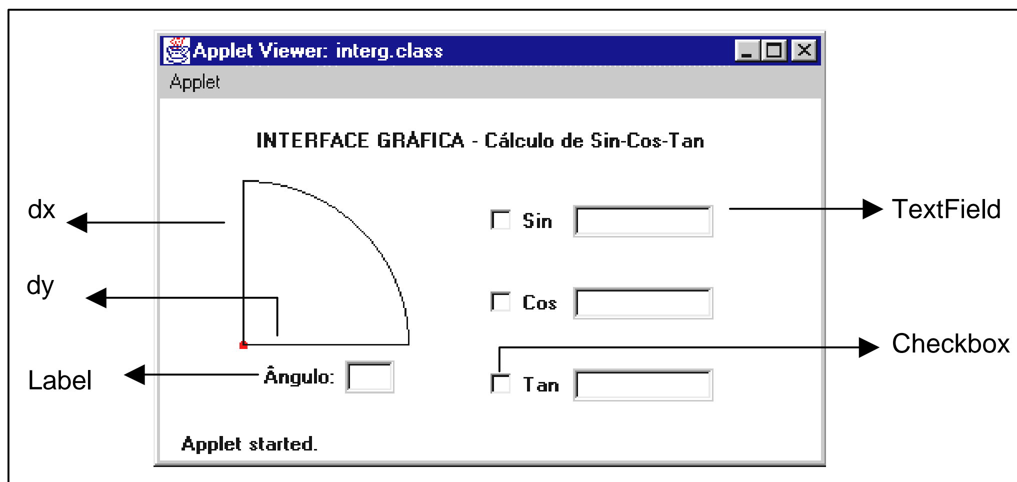


Figura 6.2: posição dos “widgets” e do quadrante na applet.

Quando o usuário “clicar” na applet, o método `mouseDown` é chamado, calculando o ângulo selecionado com o mouse e mostrando seu valor na caixa de texto (TextField) correspondente.

Após o cálculo do ângulo temos que verificar qual função foi selecionada, calcular o seu valor e mostrar o resultado no respectivo `TextField`. Essa condição é testada para todo `Checkbox` e realizada pelo teste do estado do `Checkbox(getState)`. Conforme mostramos no código abaixo.

```
if (cb1.getState() == true)
double s = dy/100;
iz = new Double(s);
t2.setText(iz.toString()); (.......)
```

Novamente precisaremos chamar o `repaint` para atualizar a tela com os novos valores calculados. Quando o usuário “clica” na applet para selecionar outro ângulo, é necessário apagar o segmento e redensenhá-lo na nova posição. Usamos portanto, um flag de controle (redesenha).

7. Animação

A animação em Java consiste em exibir diversas imagens em uma rápida sucessão, dando a ilusão de movimento. A Figura 7.1 ilustra a animação de dois objetos.

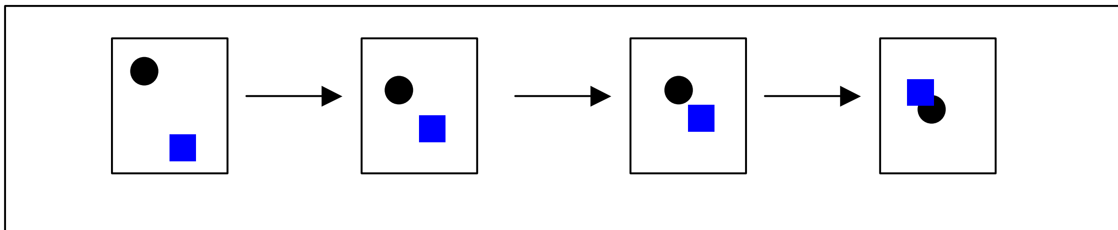


Figura 7.1: animação de dois objetos.

A Figura 7.2 mostra o código de um programa que anima dois objetos (círculo e quadrado).

```
-----------------------------------------------------------------------------------------------------------
//Programa  anim.java :
//Obs: Animação onde todas as operaçõe de desenho são observadas pelo usuário
-----------------------------------------------------------------------------------------------------------
import java.applet.*;
import java.awt.*;

      public class anim extends Applet implements Runnable    {
         //-------------------- declaração das variáveis -------------------
         Image offscreen;
         Thread animator = null;
         boolean please_stop = false;
         int x=10,y=10;
         int a=200,b=200;

         //------------------- método paint ------------------------------
         public void paint(Graphics g)  {
          g.setColor(Color.blue);
          g.fillRect(0,0,300,300);
          g.setColor(Color.black);
          g.drawRect( x, y, 100, 100);
          g.fillRect(x,y,100,100);
          g.drawOval(a,b,80,80);
          g.setColor(Color.yellow);
          g.fillOval(a,b,80,80);
        }

        //----------------------- método start ----------------------------
        //Start the animation

        public void start()  {
          animator = new Thread(this);
          animator.start();
       }
```

```
        //--------------------------- método stop ------------------------
        //Stop it
         public void stop()   {
           if (animator != null) animator.stop();
           animator = null;
         }

          //Stop and start animating on mouse clicks.
          //-----------------------método run ---------------------------
           public void run()   {
              int i,j;
              for(i=0; i<10; i++){
                  for(j=0; j<10; j++){
                  try  {
                     Thread.sleep(100);
                     x= x+10; y= y+10;
                     a= a-10; b= b-10;
                     repaint();
                   }
                   catch(InterruptedException e) { }
                 }
              }
           }
        }
//---------------------- end of file anim.java----------------------------
```

Fig 7.2: animação onde todas as operações de desenho são acompanhadas pelo usuário.

A classe `mov` implementa a multitarefa `Runnable` e utiliza o método `run` que possibilita a animação. O método `start` é chamado toda vez que uma applet é declarada. Portanto, se um usuário sair de uma página web e voltar, a applet recomeça a execução. O método `stop` é chamado quando saimos do documento HTML contendo a applet. O método `run` executa a animação, neste método atualizamos as posições dos objetos e usamos o bloco `try/catch` a fim de parar o programa 100 ms a cada atualização.

7.1 Double Buffering

A animação do programa acima apresenta uma desagradável sensação de cintilamento (flickering), devido as repetitivas operações de apagar e redesenhar os objetos na tela, sendo essas operações acompanhadas pelo usuário.

Double Buffering consiste em fazer todas estas operações de desenho em uma área gráfica na memória. Quando a imagem é concluída, ela é “jogada” para tela em uma só operação. A figura 7.3 mostra como é feita uma animação sem usar o recurso Double Buffering e outra animação usando esse recurso. Ao observar a animação usando o Double Buffering, notamos uma significativa melhora.

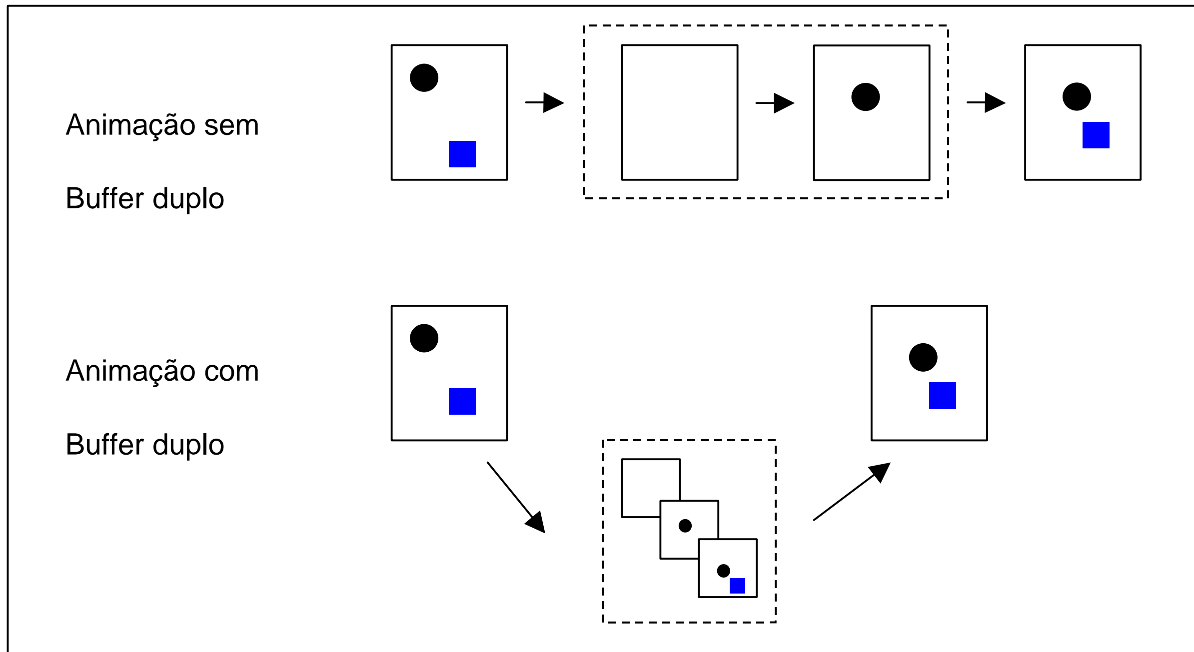


Figura 7.3: representação de uma animação usando o Buffer Duplo e outra não usando esse recurso.

Na figura abaixo apresentamos o código da mesma animação usando o Buffer Duplo:

--
//Programa screen.java :
//Obs : Minimizando o flickering da animação, usando o recurso Buffer Duplo.
--

```
import java.applet.*;
import java.awt.*;

public class screen extends Applet implements Runnable    {
  //-------------------- declaração das variáveis ----------------------------
  Image offscreen;
  Thread animator = null;
  boolean please_stop = false;
  int a=10; int b=10;
  int r=300; int s=300;
  int imagewidth, imageheight;
  int x =0;
  int y =0;

//----------------------- método start --------------------------------------
//Start the animation
    public void start()  {
      animator = new Thread(this);
      animator.start();
    }

//----------------------- método stop ----------------------------------------
//Stop it
   public void stop()   {
     if (animator != null) animator.stop();
     animator = null;
   }
```

```
//----------------------- método mouseDown ----------------------------------
// Stop and start animating on mouse clicks.
   public boolean mouseDown(Event e, int x, int y) {
    // if running, stop it.  Otherwise, start it.
       if (animator != null) please_stop = true;
       else { please_stop = false; start(); }
       return true;
  }
   //------------------------méto do run -----------------------------------
   public void run()   {
      while(!please_stop) {
      Dimension d = this.size();

      // Make sure the offscreen image is created and is the right size.
      if ((offscreen == null) ||
         ((imagewidth != d.width) || (imageheight != d.height))) {
       // if (offscreen != null) offscreen.flush();
          offscreen = this.createImage(d.width, d.height);
          imagewidth = d.width;
          imageheight = d.height;
      }
      int i,j;
      for(i=0; i<10; i++){
         for(j=0; j<10; j++){
             // Use this rectangle as the clipping region when
            // drawing to the offscreen image, and when copying
            // from the offscreen image to the screen.

             Graphics g = offscreen.getGraphics();
            // Draw into the off-screen image.
               g.setColor(Color.blue);
               g.fillRect(0,0,600,600);
               g.setColor(Color.black);
               g.drawRect(a, b, 100, 100);
               g.fillRect(a, b,100,100);
               g.setColor(Color.yellow);
               g.drawOval(r,s,80,90);
              g.fillOval(r,s,80,90);
               paint(g);
               g = this.getGraphics();
               g.drawImage(offscreen, 0, 0, this);

            try  {
                   Thread.sleep(100);
                  // Copy it all at once to the screen, using clipping.

                  a=a+10; b=b+10;
                  r=r-10; s=s-10;  }
             catch(InterruptedException e) { }
         }
      }
    }
  }
}
//-------------------end of file screen.java ---------------------------------
```

Figura 7.4: animação usando Buffer Duplo.

Observando a figura7.4, temos o método `run` dividido em 3 etapas: na primeira definimos uma imagem na memória chamada `offscreen` (Buffer Duplo).

Na segunda, cada objeto é criado no `offscreen`. Na terceira, usamos `paint(g)` para desenhar os objetos no `offscreen` e por fim copiamos a imagem do `offscreen` para a applet. No bloco `try` o programa pára 100 ms para que a applet possa ser vista e as posições dos objetos atualizadas.

8. Conclusão

Apresentamos os conceitos fundamentais da linguagem *Java* com exemplos simples, a fim de que o leitor tenha condições de começar a programar em *Java*. Os programas apresentados possibilitam a familiarização com a sintaxe da linguagem e estão disponíveis para download no endereço (http://www.cbpf.br/cat/download).

Uma das grandes vantagens da linguagem *Java* é ser interpretada e universal, permitindo que o usuário trabalhe em qualquer plataforma. É também muito utilizada na internet, pois provê uma quantidade de métodos ou subrotinas para tratar com a rede. Outra vantagem é o tamanho em bytes do código final, sendo relativamente pequeno para transferência via internet.

Por outro lado, um grande problema na utilização desta linguagem vem da biblioteca gráfica AWT, pois em todas as plataformas testadas esta responde de maneira diferente. Dependendo do tamanho do arquivo a ser executado, *Java* pode ser considerada uma linguagem lenta. Para solucionar esse problema utilizamos o compilador Just-in-Time (J.I.T.) que compila uma vez os bytecodes em código nativo. Em seguida lêem o resultado e repetem a compilação, caso necessário. Desta forma o código não será mais universal, porém a execução será mais rápida.

Com este trabalho pretendemos motivar a utilização da linguagem *Java* e mostrar a facilidade na utilização da sintaxe. O conhecimento das linguagens C ou C++ ajudarão bastante para um bom aproveitamento deste documento.

9. Bibliografia

- ✓ [Cad97] - Cadenhead, Rogers. *" Teach Yourself Java 1.1 Programming in 24 Hours".* Sams Net, 1997
- ✓ [Cor97] - Cornell, Gary, Horstmann, Cay S. *"Core Java".* Makron Books, 1997.
- ✓ [Dam96] - Damasceno Júnior, Américo. *" Aprendendo JAVA: programação na Internet".* Érica, 1996.
- ✓ [Fla97a] - Flanagan, David. *"JAVA in a Nutshell."* O'Reilly, 1997.
- ✓ [Fla97b] - Flanagan, David. *"JAVA Examples in a Nutshell."* O'Reilly, 1997.
- ✓ [Gra97a] - Grand, Mark. *"Java Language Reference".* O'Reilly, 1997.
- ✓ [Gra97b] - Grand, Mark, Knudsen, Jonathan. *"Java Fundamental Classes Reference".* O'Reilly, 1997.
- ✓ [Lin97] - Linden, Peter van der. *"Just Java".* Makron Books, 1997.
- ✓ [Nau96] - Naughton, Patrick. *"Dominando o JAVA".* Makron Books, 1996.
- ✓ [Nie97] - Niemeyer, Patrick, Peck, Joshua. *"Exploring JAVA".* O'Reilly, 1997.
- ✓ [Zuk97] - Zukowski, John. *"Java AWT Reference".* O'Reilly, 1997.