UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
CURSO DE ESPECIALIZAÇÃO EM INFORMÁTICA

Ambientes de Programação

Notas de Aula

Roberto da Silva Bigonha

UFMG

13 de fevereiro de 2017

OBJETIVOS DO CURSO AMBIENTES DE PROGRAMAÇÃO

- Apresentar uma linguagem de programação de propósito geral no desenvolvimento de aplicações para ambientes Internet/intranet.
- O curso apresenta uma visão geral da linguagem incluindo a sintaxe dos principais comandos, conceitos básicos, características marcantes da linguagem, histórico, tendências, programação de interface gráficas do usuário, etc.
- Introduzir conceitos fundamentais de linguagens orientadas por objetos.
- No fim do curso, o aluno com bom aproveitamento será capaz de desenvolver pequenas e médias aplicações nesta linguagem.

PROGRAMA

1. Java Básico
2. Classes e Objetos
3. Hierarquia de Tipos
4. Polimorfismo
5. Tratamento de Falhas
6. Biblioteca Básica
7. Classes Aninhadas
8. Pacotes
9. Agrupamento de Classes
10. Entrada e Saída
11. Genericidade
12. Coleções
13. Interfaces Gráficas
14. Processos Concorrentes
15. Applets
16. Reflexão Computacional

BIBLIOGRAFIA BÁSICA

- Bigonha, Roberto S., Programação Modular, Volume I: A Linguagem Java, Segunda Edição, DCC/UFMG, 2015.

BIBLIOGRAFIA SUPLEMENTAR

- Arnold, Ken & Gosling, James, A Linguagem de Programação Java, Bookman, Quarta Edição, 2007, ISBN 978-85-60031-64-1.
- Arnold, Ken & Gosling, James, The Java Programming Language, Addison-Wesley, 4th Edition, 2006, ISBN 0-321-34980-6.
- Cay Horstmann, Big Java, Bookman, 2004.
- Dietel, H.M. & Dietel P.J., Java Como Programar, 6a. Edition, Artmed Editora Ltda, Porto Alegre, 2005.
- Gary Conell & Cay S. Horstmann, Core Java - Fundamentos, Makron Books, Volume I e Volume II, 2001.
- James Gosling, Bill Joy, Guy Steele, Gilad Bracha & Alex Buckley. The Java™ Language Specification - Java SE 8 Edition. Oracle, 2014.

AMBIENTES DE PROGRAMAÇÃO
JAVA BÁSICO

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

19 de março de 2015

SUMÁRIO

- Introdução
- Primeiro Programa
- Tipos Básicos e Valores
- Expressões
- Arranjos
- Comandos

INTRODUÇÃO

IDENTIFICADORES DE JAVA

- Começa com letra.
- Após letra inicial pode ter letras, digitos, $ ou caractere de sublinhado (_).
- Letras são do conjunto UNICODE (16 bits).
- O conjunto ASCII ocupa as primeiras posições do conjunto UNICODE.
- Letras maiúsculas, minúsculas, acentuadas, não-acentuadas são distintas entre si na formação de identificadores.

PALAVRAS-CHAVES

- Em declarações:
  char byte short boolean int long
  float double void volatile
- Em expressões:
  null new this super true false
- Em comandos de seleção:
  if else switch case break default
- Em comandos de iteração:
  for continue do while
- Em comandos de transferência de controle:
  return throw try catch finally

PALAVRAS-CHAVES ...

- Em declarações de controle de escopo, visibilidade, tempo de vida :
  static final private
  protected public transient
- Em classes:
  class instanceof throws native interface
  abstract synchronized
- Em módulos:
  extends implements package import
- Reservadas para futuro uso:
  cast const future generic goto
  inner operator outer rest var

PRIMEIRO PROGRAMA EM JAVA

```
public class Programa {
   public static void main(String[ ] args) {
      System.out.println("Bom Dia");
   }
}
```

```
>javac Programa.java
>java Programa
SAIDA: Bom Dia
```

- Arquivo com nome da classe classe pública e extensão .java
- A menor unidade de programa compilável separadamente é a classe.
- Arquivo compilado tem extensão .class

PROGRAMA PRINCIPAL COM PARÂMETROS

```
public class Eco {
  public static void main(String[ ] args) {
     for (int i=0; i<args.length; i++)
           System.out.print(args[i] + " ");
     System.out.println(args.length);
  }

}
>javac Eco.java
>java Eco Os "parâmetros da linha de comando" sao

SAIDA: Os parâmetros da linha de comando sao 3
```

TIPOS BÁSICOS E VALORES

TIPOS EM JAVA

- **Tipos Primitivos**
  - **numéricos**
    - **inteiros: byte, char, short, int, long**
    - **fracionários: float, double**
  - **booleanos**
- **Referências**
  - **classes**
  - **interfaces**
  - **arranjos**

TIPO `boolean`

- **constantes:** `true` **e** `false`
- **operações:** `&& || ! & | ^ = == !=`
- **restrição: não se pode converter boolean em um outro tipo.**

```
boolean b, b1, b2, b3;
int i;
...
i = (b? 1 : 0); ...
b = (i == 0 ? false: true); ...
if (b) i = 10 else i = 11; ...

b = ! (b1 && b2 || b3); ....
```


TIPO `int`

- **intervalo de valores: -2.147.483.648 a 2.147.483.647 (32 bits)**
- **representação: 32 bits, complemento de 2**
- **constantes:**
  - **decimal: -256**
  - **octal: 07777 (sempre começa com zero)**
  - **hexadecimal:** `0xA5, 0xa5` **(sempre começa com** `0x`**)**
- **operações:**

  `+ - * / % = == != < > =< >= & | ~ >> >>> ++ -- <<`

TIPO `int`...

- conversões possíveis:
  - promoção implicita para numéricos maiores
  - conversão explícita para tipos numéricos menores (byte, char, short)
  - conversão explícita não verifica overflow

TIPO `long`

- intervalo de valores: (sempre 64 bits)
  -9.223.372.036.854.775.808 a
  9.223.372.036.854.775.807
- representação: 64 bits, complemento de 2
- constantes:
  - decimal: -256L
  - octal: 07777L (sempre começa com zero)
  - hexadecimal: `0xA5L`, `0xa5L` (sempre começa com `0x`)

TIPO `long`...

- operações:

```
+ - * / % = == != < > =< >= & | ~  >> >>> ++ --
```

- conversões possíveis:
  - promoção implicita para numéricos maiores
  - conversão explícita para tipos numéricos menores (byte, short, int ou char)
  - conversão explícita não verifica overflow

OPERAÇÕES COM INTEIROS

- Operadores built-in de inteiros nunca indicam ocorrências de overflow ou underflow.
- Os únicos operadores aritméticos que levantam exceção são:
  - divisão inteira ("/")
  - resto de divisão inteira ("%")

  que levantam a exceção `ArithmeticException` quando denominador for zero.
- Operação inteira é de 32-bits, com resultado do tipo `int`.
- Se pelo menos um dos operandos for `long`, os demais operandos são promovidos, e a operação é executada usando precisão de 64-bits, produzindo resultado do tipo `long`.

TIPO `byte`

- **intervalo de valores: -128 a 127**
- **representação: 8 bits, complemento de 2**
- **constantes: não há, mas**
  - **pode-se usar , sem necessidade de conversão, constantes `char` ou `int`, desde que os valores caibam em 8 bits**
  - **pode-se usar, com conversão implícita, `long` e `float`, se puder convertê-las.**

TIPO `byte`...

- **operações:** toda operação aritmética é executada com 32 bits, produzindo resultados de 32 bits.
- **conversões possíveis:**
  - **promoção implicita para numéricos maiores**

```
int  x = 1;
byte b1 = 1, b2 = 2;
byte b3 = b1 + b2 ; // Errado
byte b4 = (byte) (b1 + b2); // Correto
byte b5 = (byte) (512 + 3); //  b5 recebe 3
byte b6 = x; // Errado
byte b7 = (byte) x; // Correto
```


TIPO `short`

- **intervalo de valores: -32.768 a 32.767**
- **representação: 16 bits, complemento de 2**
- **constantes: não há, mas**
  - **pode-se usar , sem necessidade de conversão, constantes `char` ou `int` desde que os valores caibam em 16 bits**
  - **pode-se usar, com conversão implícita, constantes `long` e `float`, se puder convertê-las.**
- **operações: toda operação aritmética é executada com 32 bits, produzindo resultados de 32 bits**

TIPO `short`...

- **conversões possíveis:**
  - **promoção implícita para numéricos maiores**
  - **conversão explícita para tipos numéricos menores**

```
short b1 = 1, b2 =2;
short b3 = b1 + b2 ; // Errado
short b4 = (short) (b1 + b2); // Correto
int x = 1;
short b5 = x; // Errado
short b6 = (short) x;
```

TIPO `char`

- **intervalo de valores: 0 a 65.535**
- **representação: 16 bits, sem sinal**
- **Uso das constantes;**
  - **pode-se usar, sem necessidade de conversão, constantes `int` desde que os valores caibam em 16 bits**
  - **pode-se usar, com conversão implícita, constantes `long` e `float`, se puder convertê-las.**

TIPO `char`...

- **constantes: literais de caracters que aparecem entre apóstrofes, em um dos formatos:**
  - **Caractere individual: 'A ', 'a '**
  - **Caracteres de escape:**

`'\n'`	(nova linha)	`'\b'`	(retrocesso)
`'\"'`	(aspas)	`'\r'`	(retorno de carro)
`'\t'`	(tabulação)	`'\'`	(apóstrofe)
`'\f'`	(avanço de form.)	`'\\'`	(barra)

  - **Sequência de escape octal: somente os oito bits menos significativos, isto é, `'\000'`a `'\377'`**
  - **Sequência Unicode: `'\uxxxx'`, onde `xxxx` representa 4 dígitos hexadecimais.**

TIPO `char`...

- **operações: operação aritmética para inteiro, exceto que `char` não tem sinal.**
- **conversões possíveis:**
  - **promoção implicita para numéricos maiores**
  - **conversão explícita para tipos numéricos menores**
  - **a conversão de um valor negativo para `char` resultará em um valor positivo com a mesma configuração de bits.**

```
char b1 = 'a', b2 = 'z';
char b3 = b1 + b2 ; // Errado
char  b4 = (char) (b1 + b2); // Correto
char  b4 = (char) (b1 - b2); // Errado?
```


char É NUMÉRICO

```
public class Char1{
   public static void main(String[ ] args) {
      char x = 'A'; int  y;
      System.out.print("print(x)   =  " + x);
      System.out.println(" print(y=x) = " + (y = x));
      System.out.print("print(++x) =  " + ++x);
      System.out.print(" print(y=x) = " + (y = x));
      System.out.println(" print(x)   =  " + x);
   }
}

SAÍDA:
print(x)   =  A print(y=x) = 65
print(++x) =  B print(y=x) = 66 print(x)   =  B
```

char É NUMÉRICO ...

```
public class Char2{
   public static void main(String[ ] args) {
      char x = 'A'; int  y;
      System.out.print("print(x)      =  " + x);
      System.out.println(" print(y=x)    = " + (y = x));
      System.out.print("print((++x))  =  " + (++x));
      System.out.print(" print(y=x)    = " + (y = x));
      System.out.println("print(x)      =  " + x);
   }
}
SAÍDA:
print(x)      =  A print(y=x)    = 65
print((++x))  =  B print(y=x)    = 66 print(x)      =  B
```


char É NUMÉRICO ...

```
public class Char3{
   public static void main(String[ ] args) {
      char x = 'A'; int  y;
      System.out.println("print(x)    =  " + x);
      System.out.println("print(y=x)  = " + (y = x));
      System.out.println("print(x+1)  = " + x + 1);
   }
}
SAÍDA:
print(x)    =  A
print(y=x)  = 65
print(x+1)  = A1
```


char É NUMÉRICO ...

```
public class Char4{
   public static void main(String[ ] args) {
      char x = 'A'; int  y;
      System.out.print("print(x)     =  " + x);
      System.out.println(" print(y=x)   = " + (y = x));
      System.out.print("print((x+1)) = " + (x + 1));
      System.out.print(" print(y=x)   = " + (y = x));
      System.out.println(" print(x)     =  " + x);
   }
}
SAÍDA:
print(x)     =  A print(y=x)   = 65
print((x+1)) = 66 print(y=x)   = 65 print(x)     =  A
```


TIPO `float`

- **intervalo de valores:** `-3.4E38 a 3.4E38` **aproximadamente (sempre 32 bits), com cerca de 6 ou 7 dígitos significativos**
- **representação: ponto flutuante de 32 bits (IEEE 754)**
- **constantes:** `1E1F 1e1f 2.F 2f .3f 3.14f 6.02E+23f`
- **operações:**
  `+ - * / = == != < > =< >= >> >>> ++ --`

TIPO `float`...

- **conversões possíveis:**
  - **promoção implícita para numéricos maiores**
  - **promoção implícita de numéricos menores**
  - **conversão de `double` para `float` deve ser explícita, porque pode haver perda de precisão.**

```
float repolho = 6.5f;

float alface = 6.5; // errado 6.5 e' double

float alface = (float) 6.5;
```


TIPO `double`

- **intervalo de valores:** `-1.7E308 a 1.7E308`**, com cerca de 14 ou 15 dígitos significativos**
- **representação: ponto flutuante de 64 bits (IEEE 754)**
- **constantes:** `1E1 1e1D 2. 2.d 2.3 .3d 3.14 6.02E+23d`
- **operações:**
  `+ - * / = == != < > =< >= >> >>> ++ --`
- **conversões possíveis:**
  - **promoção implicita de numéricos menores**
  - **conversão de `double` para `float` deve ser explícita, porque pode haver perda de precisão.**

```
double cereja = '\n';  // atribui 10.0d a cereja
```


DECLARAÇÕES

```
boolean b1, b2 = true;

byte e1 = -128, e2 = 127 ;
short s1 = -32768, s2 = 32767;
char c = 'A';
int x = 1000, y;
long d = 10000000000000000;

float f = 6.5f;
double d = 6.5;

Pessoa p = new Pessoa();

int [] v = new int[100];
```


CONSTANTES SIMBÓLICAS

```
class CircleConst {
  public static final double PI = 3.1416;
}
class Naipe  {
  final static int PAUS    = 1;  final static int ESPADAS = 2;
  final static int OUROS   = 3;  final static int COPAS   = 4;
}
class Uso {
  public static void main(String[] args){
     int x = Naipe.OUROS;
     .......
     area = CircleConst.PI * raio * raio;
  }
}
```

CONSTANTES SIMBÓLICAS ...

```
enum Naipe{
   PAUS, ESPADAS, OUROS, COPAS;
}

class Uso {
  public static void main(String[] args){
     Naipe x = Naipe.OUROS;  .......
}
```


EXPRESSÕES

TIPOS DE EXPRESSÕES

- **aritmética:** `(4+5)*i`
- **lógica:** `(i > 0) && (i < 4)`
- **condicional:**
  `expression1 ? expression2 : expression3`
  `(i > 0) ? true : false`
- **com operador ,:**
  `for (i=0 , j=0 ; i < 10 && j < 20;`
  `i++ , j++) { ... }`
- **com efeito colateral:**
  `i++  ++i  a = b`

OPERADORES UNÁRIOS

`- exp`	**troca de sinal**
`! exp`	**negação lógica**
`~ exp`	**complemento de 1**
`++v`	**pré-incremento**
`--v`	**pré-decremento**
`v++`	**pós-incremento**
`v--`	**pós-decremento**
`new tipo`	**criação de objetos**
`(type-name) exp`	**type casting**

OPERADORES BINÁRIOS

*	multiplicação
/	divisão
%	resto
+	adição ou concatenação
-	subtração
<	menor que
>	maior que
<=	menor ou igual
>=	maior ou igual
==	igual
!=	diferente

OPERADORES BINÁRIOS ...

<<	deslocamento para esquerda
>>	deslocamento arit. p/ direita
>>>	deslocamento logico p/ direita
&	and bit a bit ou and lógico
|	or bit a bit ou or lógico
^	or exclusivo bit a bit ou or lógico exclusivo
&&	and lógico condicional
||	or lógico condicional
instanceof	identificação do tipo dinâmico

OPERADORES DE ATRIBUIÇÃO

```
x   = e
x  += e          x = x + e
x  -= e          x = x - e
x  *= e          x = x * e
x  /= e          x = x / e
x  %= e          x = x % e
x >>= e          x = x >> e
x >>>= e         x = x >> e
x <<= e          x = x << e
x  &= e          x = x & e
x  ^= e          x = x ^ e
x  |= e          x = x | e
```


PRECEDÊNCIA E ASSOCIATIVIDADE

Pr	OPERADORES	As
1	() [] .	LR
2	! ~ ++ -- - (type) sizeof	RL
3	* / %	LR
4	+ -	LR
5	<< >> >>>	LR
6	< > <= >= instanceof	LR
7	== !=	LR
8	&	LR
9	^	LR
10	|	LR
11	&& **(curto circuito)**	LR
12	|| **(curto circuito)**	LR
13	? :	RL
14	= += -= *= /= %= &= ^= |= <<= >>>= >>=	RL
15	, (somente no for)	LR

ORDEM DE AVALIAÇÃO DE OPERANDOS

- **Operandos são avaliados da esquerda para a direita.**
- **Em uma referência a arranjo, a expressão anterior aos `[ ]` é avaliada antes de qualquer índice.**
- **Na alocação de um arranjo, as dimensões são avaliadas, uma por uma, da esquerda para a direita.**

ORDEM DE AVALIAÇÃO DE OPERANDOS ...

```
public class OrdemDeAvaliacao {
   public static void main(String[ ] args) {
       int i = -1;
       int[ ] v = new int[8];
       for (int k=0; k < v.length; k++) {
           v[k] = k;
           System.out.print("v[" + k + "]=" + v[k] + " ");
       }
       v[i+=1] = v[i+=2] - (v[i=3] - v[i+=4]);
       System.out.println("\nValor de i = " + i);
       for (int k=0; k < v.length; k++)
           System.out.print("v[" + k + "]=" + v[k] + " ");
   }
}
```


ORDEM DE AVALIAÇÃO DE OPERANDOS ...

- **O valor inicial de `i = -1`**
- **O valor inicial do vetor `v` é:**

```
v[0]=0 v[1]=1 v[2]=2 v[3]=3 v[4]=4 v[5]=5 v[6]=6 v[7]=7
```

- **Após a execução de**

```
v[i+=1] = v[i+=2] - (v[i=3] - v[i+=4]);
```

tem-se que:

- **valor de `i = 7`**
- **valor de `v`:**

```
v[0]=6 v[1]=1 v[2]=2 v[3]=3 v[4]=4 v[5]=5 v[6]=6 v[7]=7
```


AVALIAÇÃO COM CURTO-CIRCUITO

- **Os operandos de `&&`, `||`, `?:` e `,` são avaliados na medida em que são necessários.**
- **Operadores `&&` e `||` curto-circuitam a avaliação LR da expressão tão logo seja possível inferir o resultado.**

```
int a[ ] = new int[11];
     ...
if ( (i <= 10 && a[i] == 0)) || (a.length < 11) ) {
     ...
} else{ ... }
```

ARRANJOS

ARRANJOS DE JAVA

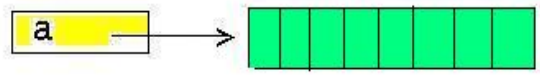
- **São objetos criados dinamicamente**
- **Zero ou mais componentes de mesmo tipo**
- **Índices de um arranjo** `v` **variam de** `0` **a** `v.length-1`
- **Indexação fora de limites gera** `IndexOutOfBoundsException`**.**
- **Tamanho de um arranjo é definido na criação e não pode ser alterado.**
- **Arranjos multidimensionais são arranjos de arranjos**
- **Declaração de arranjo especifica o número de dimensões**

ALOCAÇÃO DE ARRANJOS

- **Arranjos sempre alocados no heap via operador** `new`
- **Na criação do arranjo, especificam-se** tipo dos elementos, número de níveis de aninhamento de arranjos **e o** tamanho **de pelo menos um dos níveis (dimensão) de aninhamento.**
- **Áreas alocadas pelo** `new` **liberadas automaticamente.**
- **Elementos de arranjo inicializados conforme seu tipo.**
- `v1.length` **- retorna o número de elementos de** `v1`**.**

CRIAÇÃO DE OBJETOS ARRANJO

- `new T[tam]`
  - **aloca um vetor de tamanho** `tam` **de elementos do tipo** `T`**;**
  - **inicializa o vetor conforme** `T`**;**
  - **retorna o endereço da área alocada;**
  - **valor retornado tem tipo '**`T[ ]`**'**
- **Ex.:** `int a [ ] = new int[8];`

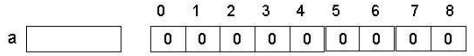


DECLARAÇÃO DE ARRANJOS

```
int [ ] a = new int[9];
int [ ] b = new int[8];
a[0] = 1;
a[1] = 2;
a[2] = 3;
b = new int[5];
```

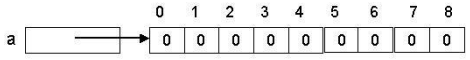
DECLARAÇÃO DE ARRANJOS ...

```
int [ ] a = new int[9];
int [ ] b = new int[8];
a[0] = 1;
a[1] = 2;
a[2] = 3;
b = new int[5];
```



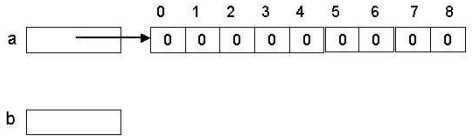
DECLARAÇÃO DE ARRANJOS ...

```
int [ ] a = new int[9];
int [ ] b = new int[8];
a[0] = 1;
a[1] = 2;
a[2] = 3;
b = new int[5];
```



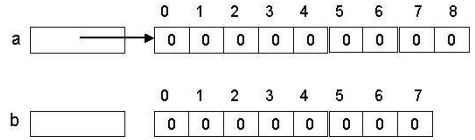
DECLARAÇÃO DE ARRANJOS ...

```
int [ ] a = new int[9];
int [ ] b = new int[8];
a[0] = 1;
a[1] = 2;
a[2] = 3;
b = new int[5];
```



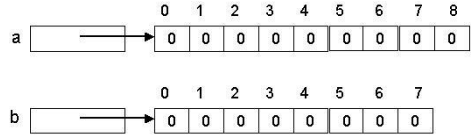
DECLARAÇÃO DE ARRANJOS ...

```
int [ ] a = new int[9];
int [ ] b = new int[8];
a[0] = 1;
a[1] = 2;
a[2] = 3;
b = new int[5];
```



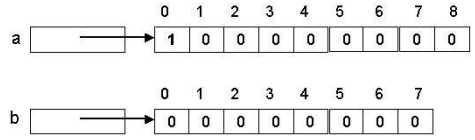
DECLARAÇÃO DE ARRANJOS ...

```
int [ ] a = new int[9];
int [ ] b = new int[8];
a[0] = 1;
a[1] = 2;
a[2] = 3;
b  = new int[5];
```



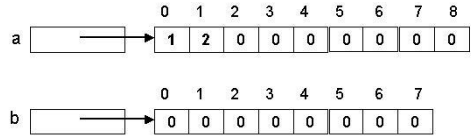
DECLARAÇÃO DE ARRANJOS ...

```
int [ ] a = new int[9];
int [ ] b = new int[8];
a[0] = 1;
a[1] = 2;
a[2] = 3;
b = new int[5];
```



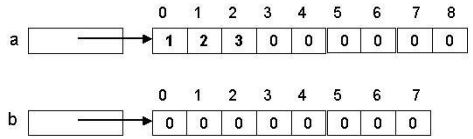
DECLARAÇÃO DE ARRANJOS ...

```
int [ ] a = new int[9];
int [ ] b = new int[8];
a[0] = 1;
a[1] = 2;
a[2] = 3;
b = new int[5];
```



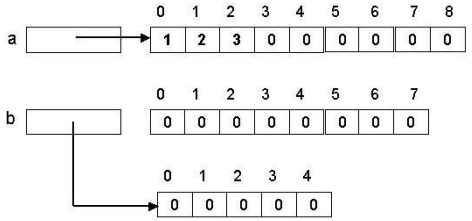
DECLARAÇÃO DE ARRANJOS ...

```
int [ ] a = new int[9];
int [ ] b = new int[8];
a[0] = 1;
a[1] = 2;
a[2] = 3;
b = new int[5];
```



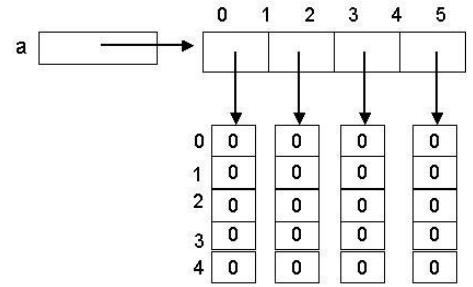
DECLARAÇÃO DE ARRANJOS

```
int [ ] a = new int[9];
int [ ] b = new int[8];
a[0] = 1;
a[1] = 2;
a[2] = 3;
b = new int[5];
```



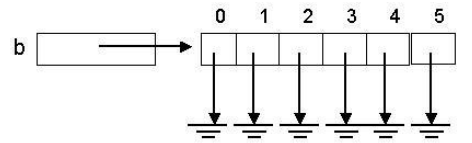
DECLARAÇÃO DE ARRANJOS MULTIDIMENSIONAIS

```
int [ ][ ] a;
a = new int[6][5];
```



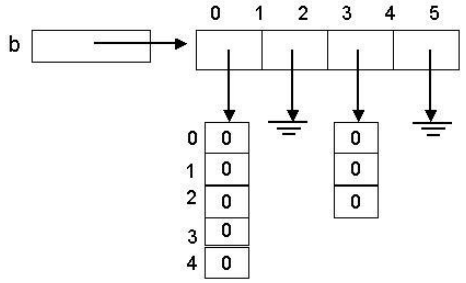
DECLARAÇÃO DE ARRANJOS MULTIDIMENSIONAIS...

```
int [ ][ ] b;
b = new int[6][ ];
b[0] = new int[5];
b[3] = new int[3];
```



DECLARAÇÃO DE ARRANJOS MULTIDIMENSIONAIS...

```
int [ ][ ] b;
b = new int[6][ ];
b[0] = new int[5];
b[3] = new int[3];
```



POSIÇÃO DOS COLCHETES

- **[ ] junto ao tipo aplica-se a todos os elementos declarados.**
- **[ ] junto a uma variável aplica-se somente a variável.**
- **Variável `j` é um arranjo de `shorts`**
- **Variável `i` é um arranjo de arranjos de `shorts`**

```
char c [ ] [ ] = new char[12][31];
char [ ] d [ ] = new char[12][31];
char [ ] [ ] e = new char[12][31];

short [ ] v, i[ ], j, k[ ][ ];
```


TAMANHO DE ARRANJO I

```
public class Array2 {
  public static void main (String [] args){
      int [][] a=new int[4][]; int [][] b= new int[4][5];
      PrintStream o = System.out;
      a[0] = new int[1];   a[1] = new int[2];
      a[2] = new int[3];   a[3] = new int[4];
      o.print("a[0]: " + a[0].length);
      o.print(", a[1]: " + a[1].length);
      o.print(", a[2]: " + a[2].length);
      o.print(", a[3]: " + a[3].length);
      o.print(", a   : " + a.length + ", b   : " + b.length);
   }
}
Saida: a[0]: 1, a[1]: 2, a[2]: 3, a[3]: 4, a: 4, b: 4
```


TAMANHO DE ARRANJO II

```
public class Array3 {
  public static void main (String [] args){
      int[ ] a ;
      for(int i=0; i < 3; i++) {
         a = new int[i];
         System.out.println("Tamanho de a : " + a.length);
      }
   }
}
Saida:
Tamanho de a   : 0
Tamanho de a   : 1
Tamanho de a   : 2
```

INDEXAÇÃO

- **Arranjos podem ser indexados por valores** `int`.
- **Índices do tipo** `byte`, `short` **e** `char` **são promovidos a** `int`.
- **Arranjos não podem ser indexados por valores** `long`.
- **Indexação fora de limites gera a exceção** `IndexOutOfBoundsException`.

UM PEQUENO EXEMPLO

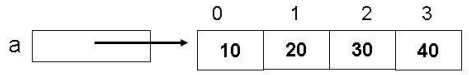
```
class Gauss {
  public static void main(String[] args) {
    int [] a = new int[101];
    for (int i=0; i < a.length; i++) a[i]= i;
    int sum = 0;
    for (int i=0; i < a.length; i++) sum += a[i];
    System.out.println(sum);
}
Saida: 5050
```


INICIAÇÃO DE ARRANJOS

- **Iniciador de arranjo pode ser especificado na declaração do arranjo, com a função de criar o arranjo e fornecer os valores iniciais dos elementos.**
- **O iniciador é uma lista de expressões, separadas por vírgula e entre chaves (**`{ e }`**).**
- **O comprimento do arranjo é o número de expressões.**
- **Cada expressão especifica o valor de um elemento do arranjo.**
- **Cada expressão deve ter tipo compatível para atribuição com o declarado para os elementos do arranjo.**
- **Pode haver iniciador dentro de iniciador (multidimensional).**

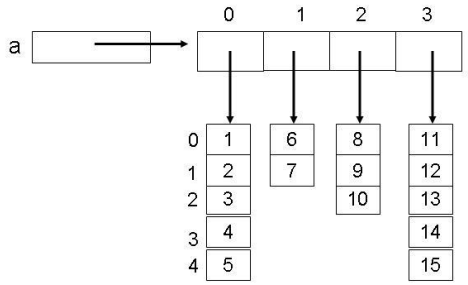
INICIAÇÃO DE ARRANJOS

```
int[ ] a =  {10, 20, 30, 40}
```



INICIAÇÃO DE ARRANJOS ...

```
int [ ][ ] a = {{1,2,3,4,5},{6,7},{8,9,10},{11,12,13,14,15}};
```



INICIAÇÃO DE ARRANJOS ...

```
public class IniciaArranjo1{
  public static void main(String args[]){
      int [] a = {1, a.length, 2, 3, 4};
      System.out.print("Elementos de a[ ] : ");
      for (int i = 0 ; i < a.length; i ++ )
            System.out.print(" " + a[i]);
  }
}
```

- **Mensagem de javac: "Variable** `a` **might not have been initialized"**

INICIAÇÃO DE ARRANJOS ...

```
public class IniciaArranjo2{
  public static void main(String args[]){
      int [ ] b;
      int [] a = {1, b.length, 2, 3, 4};
      System.out.print("Elementos de a[ ] : ");
      for (int i = 0 ; i < a.length; i ++ )
            System.out.print(" " + a[i]);
  }
}
```

- **Mensagem de javac: "Variable** `b` **might not have been initialized"**

INICIAÇÃO DE ARRANJOS ...

```
public class IniciaArranjo3{
  public static void main(String args[]){
      int [ ] b;
      if (args.length > 0) b = new int[10];
      int [ ] a = {1, b.length, 2, 3, 4};
      System.out.print("Elementos de a[ ] : ");
      for (int i = 0 ; i < a.length; i ++ )
            System.out.print(" " + a[i]);
  }
}
```

- **Mensagem de javac: "Variable** `b` **might not have been initialized"**

INICIAÇÃO DE ARRANJOS ...

```
public class IniciaArranjo4{
  public static void main(String args[]){
      int [ ] b;
      if (args.length > 0)
           b = new int[10];
      else b = new int[100];
      int [ ] a = {1, b.length, 2, 3, 4};
      for (int i = 0 ; i < a.length; i ++ )
            System.out.print(" " + a[i]);
  }
}
```

- **Saída de** `java IniciaArranjo4`**:** `1 100 2 3 4`

INICIAÇÃO DE ARRANJOS ...

```
class Teste {
   public static void main(String[] args) {
     int[ ][ ] a = { {1, 2}, null};
     for (int i =0; i < 2 ; i++){
         for (int j =0; j < 2 ; j++){
             System.out.println(a[i][j]);
         }
     }
}
Saida: 1
       2
       NullPointerException
```


COMANDOS EM JAVA

BLOCO

- `{ <lista de declarações e comandos> }`
- `{ <lista-de-comandos> }`
- **Ocultação de nomes não é permitido em escopo aninhados**
- **Exemplo:**

```
public static void main(String[] args) {
  int i = 10, j ;
  j = i*10;
  if (i == 10) {int j = 100; ... }
  j = j + k;
}
```

COMANDO if

- if (<exp_booleana>) <comando>
- if (<exp_booleana>) <comando1> else <comando2>

```
int numofdigits = 0, numofothers = 0;
String s; ........;
for (int i=0 ; i < s.length(); i++) {
    if (s.charAt(i) == '0' || s.charAt(i) == '1' ||
        s.charAt(i) == '2' || s.charAt(i) == '3' ||
        s.charAt(i) == '4' || s.charAt(i) == '5' ||
        s.charAt(i) == '6' || s.charAt(i) == '7' ||
        s.charAt(i) == '8' || s.charAt(i) == '9') {
           numofdigits++;
    } else numofothers++;
}
System.out.print(numofdigits + " " + numofothers);
```


COMANDOS switch

```
switch ( <expressao_inteira> ) {
  case <exp_constante1>:
       .....
       break;
  case <exp_constante2>:
       ....
       break;
  ..................
  default:
       ....
       break;
}
```

- **Se nenhum caso for escolhido e não houver cláusula default, a execução continua no próximo comando.**

COMANDOS switch...

```
public static void main(String[] args) {
  int x = 0, k = 2;
  switch (k) {
    case 1: x = x + 1;
            break;
    case 2: x = x + 1;
    case 3: x = x + 1;
    case 4: x = x + 1;
            break;
    case 5: x = 5;
    default: x = x + 1000;
  };
  System.out.println("X = " + x); // Imprime "X = 3"
}
```


COMANDOS switch...

```
public static void main(String[] args) {
  int x = 0, k = 2;
  switch (k) {
    default: x = 1000;
    case 1: x = x + 1;
            break;
    case 2: x = x + 1;
    case 3: x = x + 1;
    case 4: x = x + 1;
            break;
    case 5: x = 5;
  };
  System.out.println("X = " + x); // Saida: X = 3
 }
```

COMANDOS switch...

```
public static void main(String[] args) {
  int x = 0, k = 200;
  switch (k) {
    default: x = 1000;
    case 1: x = x + 1;
            break;
    case 2: x = x + 1;
    case 3: x = x + 1;
    case 4: x = x + 1;
            break;
    case 5: x = 5;
  };
  System.out.println("X = " + x);// Saida: X= 1001
 }
```


COMANDOS switch...

```
public static void main(String[] args) {
  int x = 0, k = 5;
  switch (k) {
    case 1: x = x + 1;
            break;
    case 2: x = x + 1;
    case 3: x = x + 1;
    case 4: x = x + 1;
            break;
    case 5: x = 5;
    default: x = x + 1000;
  };
  System.out.println("X = " + x); // Imprime "X = 1005"
}
```


COMANDOS switch...

```
int numofdigits = 0, numofothers = 0;
String s; ........
for (int i=0 ; i < s.length(); i++) {
    switch(s.charAt(i)) {
      case '0': case '1': case '2': case '3':
      case '4': case '5': case '6': case '7':
      case '8': case '9':
         numofdigits++;
         break;
      default : numofothers++;
    }
}
```


COMANDOS while

- `while ( <expressao_booleana> ) <comando>`
- **Exemplo:**

```
int[ ] a = {0,0,0,0,0,3,5,10,1,2,0,4};
int h = 0;

while ( a[h] == 0) h++;

while ( h < a.length ) {
   System.out.println(a[h]);
   h++;
}
```

COMANDOS do-while

- do <comando> while (<expressao_booleana>)
- **Exemplo:**

```
int[ ] a = {0,0,0,0,0,3,5,10,1,2,0,4};
int h = 0;

do {h++;} while ( a[h-1] == 0 ) ;

if ( h < a.length )
   do {
       System.out.println(a[h]);
       h++;
   } while ( h < a.length )
```


COMANDO for

- for (<expr-init>; <condição> ; expr-incr>) <comando>

Exemplos:

```
int s = 0, a [ ] = {1,2,3,4,5,6,7,8,9};

for (int i = 0; i < a.length ; i++) s += a[i];

for (int i=0, j=a.length-1 ; i<a.length ; i++,j--) {
    System.out.println("a["+i+"]+a["+j+"]= "+(a[i]+a[j]));
}
```


COMANDO for...

- **Cuidado: Programa abaixo tem vários erros!**

```
public class For1{
  public static void main( String [] args) {
    int a [ ] = {1,2,3,4,5,6,7,8,9};
    int i = 9, j = 9;
    for (int i=0, int j=a.length-1 ; i<a.length ; i++,j--)
      System.out.println("a["+i +"]+a["+j+"]= "+(a[i]+a[j]));
    System.out.println("a["+i +"]+a["+j+"]= "+(a[i]+a[j]));
    for (int i=0, int j=a.length-1 ; i<a.length ; i++,j--)
      System.out.println("a["+i +"]+a["+j+"]= "+(a[i]+a[j]));
   }
}
```


COMANDO for...

- **Agora funciona!**

```
public class For2 {
  public static void main( String [] args) {
    int a [ ] = {1,2,3,4,5,6,7,8,9};
    for (int i=0, j=a.length-1 ; i<a.length ; i++,j--)
      System.out.println("a["+i +"]+a["+j+"]= "+(a[i]+a[j]));
    for (int i=0, j=a.length-1 ; i<a.length ; i++,j--)
      System.out.println("a["+i +"]+a["+j+"]= "+(a[i]+a[j]));
  }
}
```

COMANDO `for`...

- **Agora também funciona!**

```
public class For3 {
  public static void main( String [] args) {
    int a [ ] = {1,2,3,4,5,6,7,8,9};
    for (int i=0, j=a.length-1 ; i<a.length ; i++,j--)
      System.out.println("a["+i +"]+a["+j+"]= "+(a[i]+a[j]));
    for (int i=0, j=a.length-1 ; i<a.length ; i++,j--)
      System.out.println("a["+i +"]+a["+j+"]= "+(a[i]+a[j]));
    int i = 8, j = 8;
    System.out.println("a["+i +"]+a["+j+"]= "+(a[i]+a[j]));
  }
}
```


COMANDO `for` APRIMORADO

- `for ( <tipo> <identificador> : NomeDeArranjo) <comando>`
- `for ( <tipo> <identificador> : NomeDeColeção) <comando>`
- **Exemplos:**

```
int s = 0;
int [] a = {1,2,3,4,5,6,7,8,9};
for (int x : a) s += x;
System.out.print(s);
```

- **O identificador no `for` não pode ser usado para modificar elementos do arranjo**
- **Deve ser usado apenas para acesso aos elementos do arranjo.**

COMANDO `continue`

- **Comando `continue` somente pode ocorrer dentro de comandos `for`, `while` ou `do`**
- **Dentro de `while` ou `do`, `continue` causa a expressão do loop ser novamente avaliada e testada para execução da próxima iteração**
- **Dentro de `for`, `continue` causa a expressão de incremento ser avaliada e a expressão do loop ser testada para execução da próxima iteração**
- `continue` **tem dois formatos:**
  - `continue` **que se refere ao loop envolvente mais próximo**
  - `continue <Rótulo>`, **que se refere ao loop envolvente que tem o rótulo indicado**

COMANDO `continue` ...

```
public class Continue1 {
  public static void main(String[] args) {
    int j, a = 0, i= 0;
    externo: while (i < 7) {
         j = 0; i++;
         while(j < 6) {
              if (i < 4){j++; i++; continue;}
              if (i==5) continue externo;  a++; j++;
         }
         System.out.print("(" + i + "," + a + ")" + ", "); i++;
    }
    System.out.print(a);
  }
}
```

COMANDO `continue` ...

```
public class Continue2 {
   public static void main(String[] args) {
     int a = 0;
     externo: for (int i = 0; i < 7 ; i++) {
          for (int j = 0; j < 6; j++) {
              if (i < 4 || i == 6) {continue;}
              if (i==5) {continue externo;}
              a++;
          }
          System.out.print("(" + i + "," + a + ")" + ", ");
      }
      System.out.print(a);
   }
} // Saida: (0,0), (1,0), (2,0), (3,0), (4,6), (6,6), 6
```


COMANDO `break`

- **Comando** `break` **somente pode ocorrer dentro de comandos** `switch`, `for`, `while` **ou** `do`
- **Tem dois formatos:**
  - `break`, **que termina o comando** `switch`, `for`, `while` **ou** `do` **envolvente mais próximo**
  - `break <Rótulo>`, **que termina o comando** `switch`, `for`, `while` **ou** `do` **envolvente que tem o rótulo indicado**

COMANDO `break`...

```
public class Break1 {
   public static void main(String[] args) {
      int a = 0;
      inicio: for (int j = 0; j < 6; j++) {
          switch (j) {
             case 0 : a = 20;   break;
             case 2 : a = 30;   break;
             default: a = 100;  break inicio;
          }; a++;
          System.out.print("(" + j + "," + a + ") " );
      };
      System.out.print("Valor final de a = "  + a);
   }
}    // Saida: (0,21) Valor final de a = 100
```


OUTROS COMANDOS

- **Returno de função:**
  `return <exp>;`
- **Levantamento de exceção:**
  `throw <exp>;`
- **Habilitação e tratamento de exceção:**
  `try <bloco> <catches>`
- **Sincronização de acesso:**
  `synchronized (<exp>) <bloco>`

FIM

AMBIENTES DE PROGRAMAÇÃO
CLASSES E OBJETOS

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

19 de março de 2015

SUMÁRIO

- Noção de Objetos
- Classes de Java
- Declaração de Objetos
- Criação de Objetos
- Controle de Visibilidade
- Métodos
  - Passagem de Parâmetros
  - Mensagens e Métodos
- Função Construtora
- Função Finalizadora
- Acesso a Campos de objetos
- Campos e Variáveis
- Arranjo de Objetos
- Iniciação de Objetos
  - Iniciação de Campos
  - Iniciação Complexa
- Exemplos
  - Conta em Banco

CLASSES

PRIMEIRA NOÇÃO DE OBJETOS

- Objetos são conceitos de sofware que modelam entidades da aplicação
- Objetos são abstrações de dados
- Objetos têm estado (estrutura interna)
- Objetos são manipulados só pelas operações
- Objetos correspondem a variáveis

CONCEITO DE CLASSES

- **Classe é um conceito de software que descreve a estrutura e o comportamento de um conjunto de objetos.**
- **Uma classe consiste em:**
  - **uma estrutura de dados;**
  - **um conjunto de operações para manipular esta estrutura;**
  - **uma interface bem definida.**
- **Classes servem para definir novos tipos abstratos de dados.**
- Objeto **está para sua** classe **assim como uma** variável **está para seu** tipo.
- **Definição de classes não aloca memória para campos de objetos, porque classe é apenas um molde para criar objetos.**

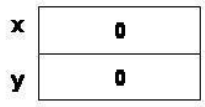
CONCEITO SUPORTADOS POR CLASSES

- Abstração
- Encapsulação
- Proteção de Dados
- Polimorfismo
- Hierarquia

DEFINIÇÃO DE CLASSES

```
class Ponto1 {
  public double x, y;
  public void clear( ) {
    this.x = 0; this.y = 0;
  }
}
```

```
class Ponto1 {
  public double x, y;
  public void clear( ) {
     x = 0; y = 0;
  }
}
```



- **Campos de classes são inicializados por default.**
- **Variáveis locais de métodos não são inicializadas automaticamente.**

CLASSES DE JAVA

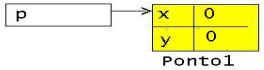
- **Classes introduzem a implementação de um novo tipo.**
- **Elementos da classe podem ser:**
  - **membros de dados ou campos**
  - **construtores**
  - **finalizadores**
  - **membros funções ou métodos**
  - **iniciadores estáticos**
- **As operações da classe são definidas por membros função.**
- **Nomes de métodos podem ser sobrecarregados.**
- **Construtores são como métodos, mas não podem ser chamados diretamente.**

DECLARAÇÃO DE OBJETOS

- **Uma declaração de classe não reserva memória apenas para o descritor do tipo definido pela classe.**
- **Declaração da forma** `ClassName x;` **cria uma** referência `x` **para objetos do tipo** `ClassName`.
- **Declaração da forma** `ClassName [] x;` **Cria uma** referência `x` **para objetos do tipo arranjo de** `ClassName`.

CRIAÇÃO DE OBJETOS DE CLASSE

- `new ClassName(args)`
  - **aloca um objeto da classe** `ClassName`;
  - **inicializa os campos do objeto conforme seus tipos;**
  - **inicializa os campos conforme expressão de inicialização definida, se for o caso;**
  - **executa a função construtora;**
  - **retorna o endereço da área alocada.**
  - **valor retornado tem tipo 'referência a** `ClassName`**'**
- **Exemplo:** `Ponto1 p = new Ponto1();`

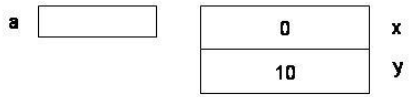


CRIAÇÃO E MANIPULAÇÃO DE OBJETOS

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
           main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```

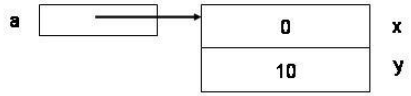
CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
           main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



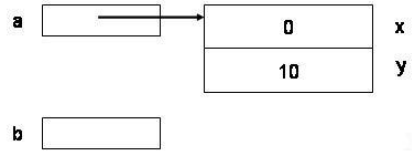
CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
           main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



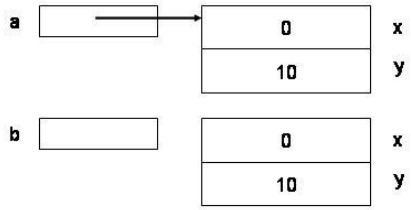
CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
           main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



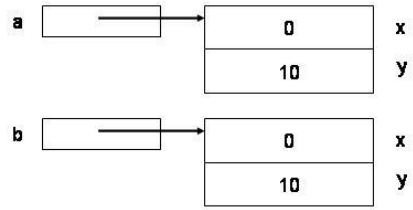
CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
           main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



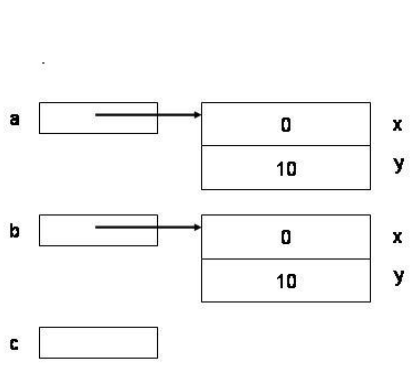
CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
           main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



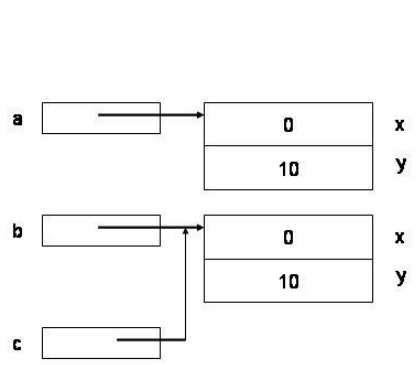
CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
           main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



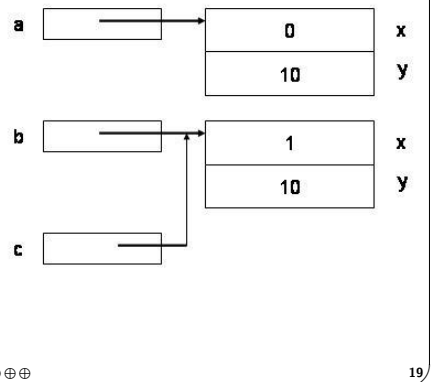
CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
           main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



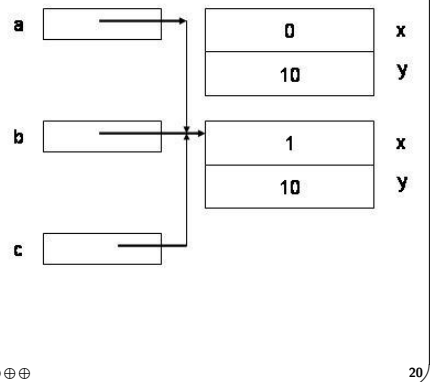
CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
           main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



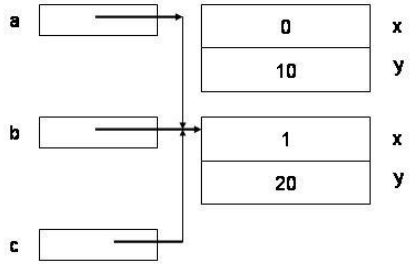
CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
           main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



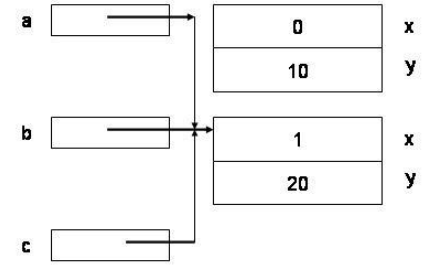
CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
            main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



CRIAÇÃO E MANIPULAÇÃO DE OBJETOS ...

```
class M {
  public int x;
  public int y = 10;
}
public class Teste {
  public static void
            main(String[] args){
    M a = new M(), b = new M();
    M c;
    c = b; c.x = b.x + 1;
    a = b; a.y = b.y + c.y;
  }
}
```



CONTROLE DE VISIBILIDADE

Campos, métodos e construtores têm o formato:

`<modificadores de acesso> tipo nome ...;`

Modificadores de controle de acesso a membros são:

- `public`: **membros declarados como** `public` **são acessíveis de qualquer lugar onde a classe for acessível, inclusive nas respectivas subclasses.**
- `private`: **membros declarados como** `private` **somente são acessíveis a membros funcionais da própria classe.**
- `protected`: **membros protegidos são membros privados accessíveis nas subclasses e dentro do mesmo pacote.**
- **vazio: membros declarados sem modificador de acesso são acessíveis somente no mesmo pacote.**

CONTROLE DE VISIBILIDADE ...

```
class Ponto1 {
  public double x, y;
  public void clear( ) {
    this.x = 0; this.y = 0;
  }
}
```

```
class Ponto2 {
    private double x, y;
    public void clear( ) {
       x = 0; y = 0;
    }
}
```

```
class A {
   public static void main(String[ ] args) {
      Ponto1 a = new Ponto1( );
      Ponto2 b = new Ponto2( );
      a.x = a.y + 1;
      b.x = b.y + 1; // ERRADO
   }
}
```

MÉTODOS

PASSAGEM DE PARÂMETROS DE MÉTODOS

- **Parâmetros de métodos são sempre passados** por valor.
- **Note que :**
  - **O valor de um objeto é sua referência**
  - **Arranjos são referências a objetos**
- **Não há passagem por referência em Java, mas referências podem ser passadas por valor.**
- **Objetos referenciados por parâmetros podem ser modificados.**
- **Referências passadas como parâmetros não são modificadas**

PASSAGEM DE PARÂMETROS DE MÉTODOS ...

```
class Passagem1 {
  public static void main (String [ ] args) {
     double x = 1.0 ;
     System.out.println("A: x  = " + x);
     divida(x);
     System.out.println("D: x = " + x);
  }
  public static void divida (double arg) {
     arg /= 2.0;
     System.out.println("M: arg =" + arg);
  }
}
SAIDA: A: x  = 1   M: arg = 0.5   D: x = 1
```


PASSAGEM DE PARÂMETROS ...

```
class Ponto3 {
    private double x, y;

    public void clear( ) {
       x = 0; y = 0;
    }
    public void set(double a, double b) {
       x = a; y = b;
    }
    public void show( ) {
       System.out.print("mostra ponto (" + x + "," + y + ")");
    }
}
```

PASSAGEM DE PARÂMETROS ...

```
class Passagem2 {
  public static void altera1(Ponto3 p) { p.clear();  }
  public static void altera2(int[] t) { t[2]= 4;  }
  public static void main (String [ ] args) {
     Ponto3 x = new Ponto3();
     int [ ] y = {1, 2, 3};
     x.set(10.0, 10.0); x.show();
     altera1(x); x.show();
     altera2(y);
     for(int i=0;i<y.length;i++) System.out.print(" " + y[i]);
  }
}
SAIDA: mostra ponto (10,10)    mostra ponto (0,0)  1 2 4
```


MENSAGENS E MÉTODOS

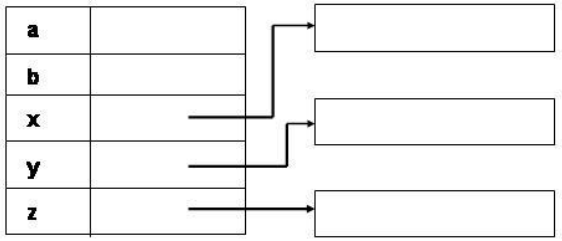
- **As funções membros de uma classe também são chamadas de métodos.**
- **A operação de chamar uma das funções de uma classe para um dado objeto, como por exemplo,** `z.clear()`**, é interpretada como** enviar uma mensagem ao objeto receptor.
- Objeto receptor **é aquele recebe a mensagem e trata mensagem. É o** `z`**, no caso acima.**
- **Executar a mensagem é executar a função membro do objeto que corresponde à mensagem recebida.**

FUNÇÃO CONSTRUTORA

```
class ClassName {
   Tipo ed;
   public ClassName () { ... }
   public ClassName (Tipo1 args1) { ... }
   public ClassName (Tipo2 args2, Tipo1 arg3)
        { ... }
}
```


FUNÇÃO CONSTRUTORA...

```
classe C {
    Tipo1 a; Tipo2 b;
    ClassName x = new ClassName();
    ClassName y = new ClassName(a);
    ClassName z = new ClassName(b,a);
}
```

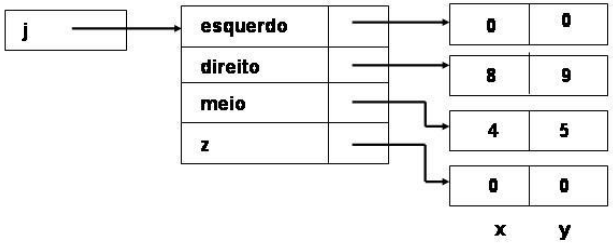


FUNÇÃO CONSTRUTORA...

```
class Ponto4 {
  private double x, y;
  Ponto4( ) { x = 0; y = 0;  }
  Ponto4(double a, double b) {
    x = a; y = b;
  }

  public void set(double a, double b) { x = a; y = b; }
  public void hide( ) { ... }
  public void show( ) { ... }
}
```


FUNÇÃO CONSTRUTORA...

```
class Janela2{
    Ponto4 esquerdo  = new Ponto4(), direito = new Ponto4(8,9);
    Ponto4 meio = new Ponto4(4,5), z = new Ponto4();
    ...
}
... Janela2 j = new Janela2();
```



REGRAS DA FUNÇÃO CONSTRUTORA

- **Tem o mesmo nome da classe e não tem tipo de retorno.**
- **É opcional.**
- **Pode ser** `public`, `protected` **ou** `private`.
- **Se função construtora for** `private`, **objetos da classe nunca poderão ser criados fora da classe.**
- **Se função construtora for** `protected`, **somente herdeiros poderão criar objetos da classe.**

REGRAS DA FUNÇÃO CONSTRUTORA ...

- **Se não houver, na classe, definição alguma de construtoras, por default é criada uma sem parâmetros.**
- **Construtora default somente é criada no caso acima.**
- **Possível mais de uma função construtora por classe, via overloading**
- **Chamada à construtora ocorre na criação dos objetos:**
  - `ClassName x = new ClassName(args);`

FUNÇÃO FINALIZADORA

- Executada automaticamente quando área do objeto for recolhida pelo coletor de lixo.
- Permite executar ações antes que o objeto seja efetivamente destruído.
  Por exemplo, pode-se fechar arquivos, etc.

FUNÇÃO FINALIZADORA ...

```
public class ProcessFile {
  private Stream file;
  public ProcessFile(String path){file = new Stream(path); }
  public void close() {
   if (file != null) { file.close(); file.null(); }
  }
  protected void finalize() throws Throwable {
      super.finalize();
      close();
   }
}
```


RESSURREIÇÃO DE OBJETOS

- Um método de finalização pode ressuscitar um objeto, criando uma referência para ele,
  por exemplo, via uma referência estática.
- Ressurreição não é uma boa prática de programação.
- Java invoca `finalize` no máximo uma única vez para cada objeto.
- Objetos ressuscitados são recolhidos pelo coletor de lixo sem chamada à função `finalize`.
- Isto é, objetos ressuscitados perdem o direito a finalização.
- A boa prática recomenda que se faça um clone do objeto em vez de ressuscitá-lo.

EXECUÇÃO DO COLETOR DE LIXO I

```
public class G1 {
  public static void main(String[ ] args) {
      A a = new A(); a = new A ( );
      System.gc ();
      for (int i=1; i <1000000; i++);
      System.out.print("Acabou!");
  }
}
class A {
  protected void finalize( ) throws Throwable {
      super.finalize();
      System.out.print("Finalize foi chamada. ");
  }
} Saída: Finalize foi chamada. Acabou!
```

EXECUÇÃO DO COLETOR DE LIXO II

```
public class G2 {
  public static void main(String[ ] args) {
     A a = new A(); a = new A ( );
     //System.gc ();
     for (int i=1; i <1000000; i++);
     System.out.print("Acabou!");
  }
}
class A {
  protected void finalize( ) throws Throwable {
     super.finalize();
     System.out.print("Finalize foi chamada. ");
  }
} Saída: Acabou!
```


EXECUÇÃO DO COLETOR DE LIXO III

```
public class G3 {
  public static void main(String[ ] args) {
     A a = new A(); a = new A ( );
     System.gc ();
     //for (int i=1; i <1000000; i++);
     System.out.print("Acabou!");
  }
}
class A {
  protected void finalize( ) throws Throwable {
     super.finalize();
     System.out.print("Finalize foi chamada. ");
  }
} Saída: Acabou!Finalize foi chamada.
```


EXECUÇÃO DO COLETOR DE LIXO IV

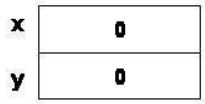
```
public class G4 {
  public static void main(String[ ] args) {
     A a = new A(); a = new A ( );
     System.gc ();
     //for (int i=1; i <1000000; i++);
     //System.out.print("Acabou!");
  }
}
class A {
  protected void finalize( ) throws Throwable {
     super.finalize();
     System.out.print("Finalize foi chamada. ");
  }
} Saída: Finalize foi chamada.
```


ACESSO A CAMPOS DE OBJETOS

CONSIDERE A DEFINIÇÃO DE `Ponto`

```
class Ponto {
  public double x, y;
  public void clear( ) {
    this.x = 0; this.y = 0;
  }
}
```

```
class Ponto {
    public double x, y;
    public void clear( ) {
       x = 0; y = 0;
    }
}
```



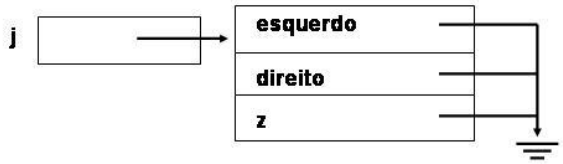
USO DE `this`

```
class Janela {
   Ponto esquerdo  = new Ponto( );
   Ponto direito   = new Ponto( );
   Ponto z;
   public void set( ){
     this.esquerdo.x  = 15;
     this.z = this.direito;
     this.direito.x  = 10;
     this.direito.y  = 20;
     this.z.clear();
   };
}
....
Janela j = new Janela( ); j.set( );
```


EXECUÇÃO DE `Janela j= new Janela()`

```
class Janela {
   Ponto esquerdo  = new Ponto( );
   Ponto direito = new Ponto( );
   Ponto z;
   ...
}
... Janela j = new Janela( ); j.set( );
```

Primeira fase do `new` (alocação e inicialização por default):



EXECUÇÃO DE `Janela j= new Janela()`...

```
class Janela
   Ponto esquerdo  = new Ponto( );
   Ponto direito = new Ponto( );
   Ponto z;
   ...

... Janela j = new Janela( ); j.set( );
```

Segunda fase do `new` (inicialização)

EXECUÇÃO DE `j.set()`

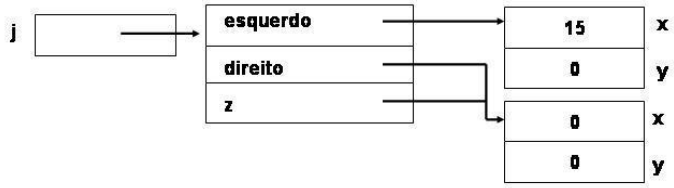
```
class Janela {   ...
   public void set(){
     this.esquerdo.x  = 15;     this.z = this.direito;
     this.direito.x = 10;     this.direito.y  = 20;
     this.z.clear();
    };
}
... Janela j = new Janela( ); j.set(); ...
```

EXECUÇÃO DE `j.set()`...

```
class Janela {   ...
   public void set(){
     this.esquerdo.x  = 15;     this.z = this.direito;
     this.direito.x = 10;     this.direito.y  = 20;
     this.z.clear();
    };
}
... Janela j = new Janela( ); j.set(); ...
```

EXECUÇÃO DE `j.set()`...

```
class Janela {   ...
   public void set(){
     this.esquerdo.x  = 15;     this.z = this.direito;
     this.direito.x = 10;     this.direito.y  = 20;
     this.z.clear();
    };
}
... Janela j = new Janela( ); j.set(); ...
```

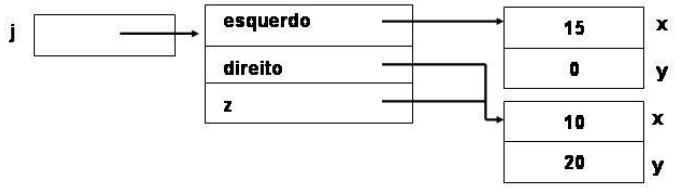


EXECUÇÃO DE `j.set()`...

```
class Janela {   ...
   public void set(){
     this.esquerdo.x  = 15;     this.z = this.direito;
     this.direito.x = 10;     this.direito.y  = 20;
     this.z.clear();
    };
}
... Janela j = new Janela( ); j.set(); ...
```

EXECUÇÃO DE `j.set()`...

```
class Janela {   ...
   public void set(){
     this.esquerdo.x  = 15;     this.z = this.direito;
     this.direito.x = 10;     this.direito.y  = 20;
     this.z.clear();
    };
}
... Janela j = new Janela( ); j.set(); ...
```

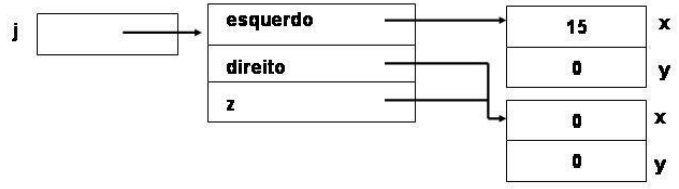


EXECUÇÃO DE `j.set()`...

```
class Janela {   ...
   public void set(){
     this.esquerdo.x  = 15;     this.z = this.direito;
     this.direito.x = 10;     this.direito.y  = 20;
     this.z.clear();
    };
}
... Janela j = new Janela( ); j.set(); ...
```

EXECUÇÃO DE `j.set()`...

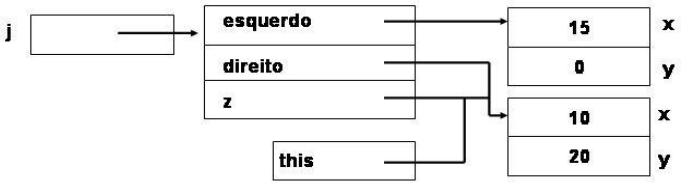
```
class Janela {   ...
   public void set(){
     this.esquerdo.x  = 15;     this.z = this.direito;
     this.direito.x = 10;     this.direito.y  = 20;
     this.z.clear();
    };
}
... Janela j = new Janela( ); j.set(); ...
```



VALOR DE REFERÊNCIA `this`

```
class Ponto {
  public double x, y;
  public void clear( ) {
    this.x = 0; this.y = 0;
  }
}
```

```
class Janela {
    ... z.clear(); ...
}
```

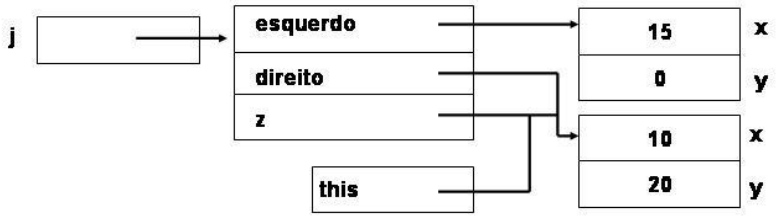


COMPILAÇÃO DE this

```
class Ponto {
  public double x, y;
  public void clear() {
          // public void clear(Ponto this){
    this.x = a; this.y = b;
  }
}

class Janela1 {
     ... z.clear(); ...      // clear(z);
}
```

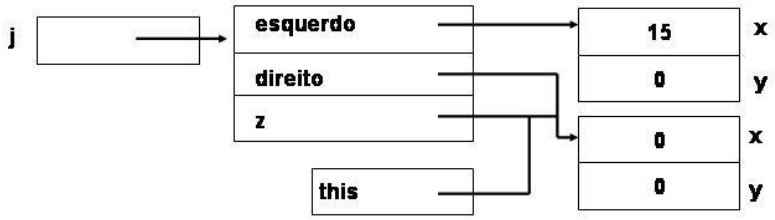

COMPILAÇÃO DE this...

```
class Janela {
     ... z.clear(); ...      // clear(z);
}
```



COMPILAÇÃO DE this...

```
class Janela {
     ... z.clear(); ...      // clear(z);
}
```

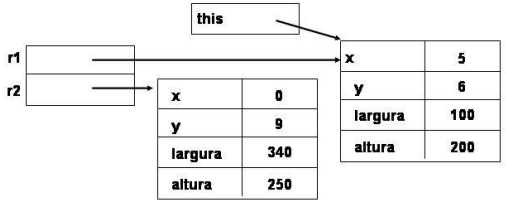


REFERÊNCIA this

```
class Rectangle {
  private int x; private int y;
  private int largura;  private int altura;
  public Rectangle(int x, int y, int w, int h){
     this.x = x;
     this.y = y;
     this.largura = w; altura = h;
  }
  public void draw(){
     System.out.println("Rectangle: " +
       x + ","+ y + "," + largura + "," + altura);
  }
  public static void main(String args[ ]){ ...}
}
```

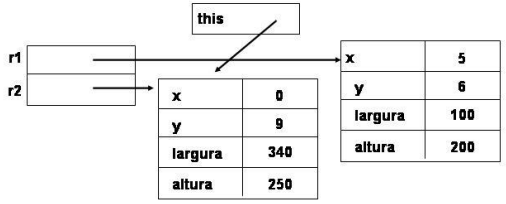
REFERÊNCIA this...

```
  public static void main(String args[ ]) {
    Rectangle r1 = new Rectangle(5,5,100,200);
    Rectangle r2 = new Rectangle(0,9,340,250);
    r1.draw(); r2.draw();
  }
Saida: Rectangle 5, 5, 100, 200
       Rectangle 0, 9, 340, 250
```



REFERÊNCIA this...

```
  public static void main(String args[ ]){
    Rectangle r1 = new Rectangle(5,5,100,200);
    Rectangle r2 = new Rectangle(0,9,340,250);
    r1.draw(); r2.draw();
  }
Saida: Rectangle 5, 5, 100, 200
       Rectangle 0, 9, 340, 250
```

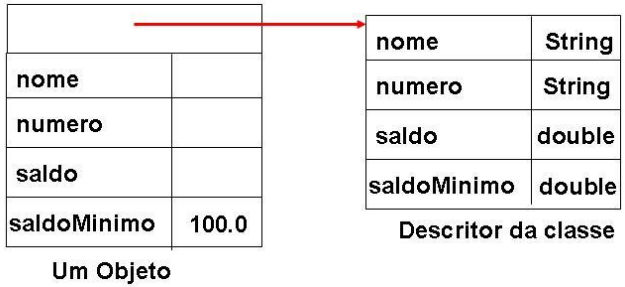


CAMPOS E VARIÁVEIS

CAMPOS DE OBJETOS

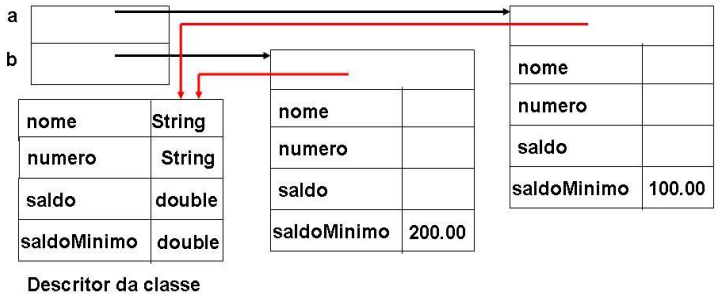
```
class Conta1 {
  String nome;  String numero;
  double saldo; double saldoMinimo = 100.00;
}
```

Leiaute de objetos Conta1:



CAMPOS DE OBJETOS ...

```
class Banco1 {
  Conta1 a = new Conta1();
  Conta1 b = new Conta1(); b.saldoMinimo = 200.00;
} ... new Banco1();...
```

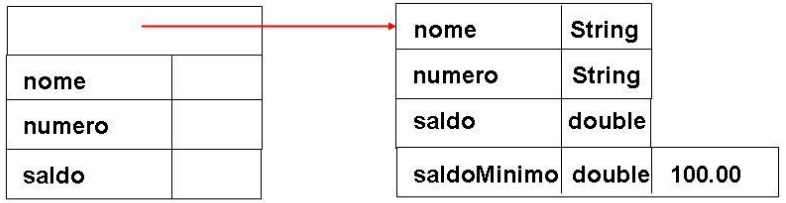


Descritor da classe

VARIÁVEIS DE CLASSE

```
class Conta2{
  String nome;   String numero;
  double saldo;  static double saldoMinimo = 100.00;
}
```

Leiaute de objetos Conta2:

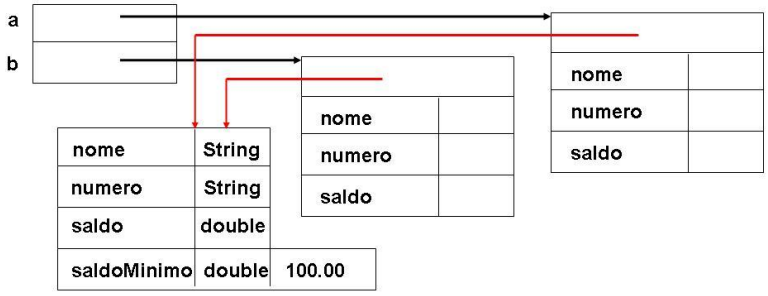


Um Objeto

Descritor da classe

VARIÁVEIS DE CLASSE ...

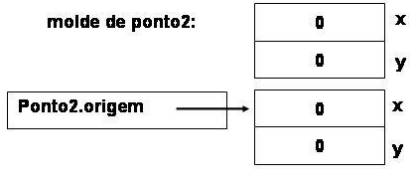
```
class Banco2{
  Conta2 a = new Conta2(); Conta2 b = new Conta2();...
}
.... new Banco2(); ... Conta2.saldoMinimo = 1000;
```



Descritor da classe

ALOCAÇÃO DE VARIÁVEIS DE CLASSE

```
class Ponto2 {
  public double x, y;
  public static Ponto2 Origem = new Ponto2();
  public void clear( ) {x = 0; y = 0;}
}
```



- **Campos estáticos são inicializados na primeira vez em que a classe que os contém for encontrada durante a execução, i.e., um de métodos ou campos forem acessados.**

ALOCAÇÃO DE VARIÁVEIS DE CLASSE ...

```
class Refs {
  public static void main(String [] args){
     Ponto2 esquerdo = Ponto2.Origem;
     Ponto2 direito = new Ponto2();  Ponto2 z = direito;
     esquerdo.x  = 15; direito.x = 10.0; direito.y = 20.0;
     z.clear(); esquerdo.clear();
  }
}
```

ALOCAÇÃO DE VARIÁVEIS DE CLASSE ...

```
class Refs {
  public static void main(String [] args){
     Ponto2 esquerdo = Ponto2.Origem;
     Ponto2 direito = new Ponto2(); Ponto2 z = direito;
     esquerdo.x  = 15; direito.x = 10.0; direito.y = 20.0;
     z.clear(); esquerdo.clear();
  }
}
```

ALOCAÇÃO DE VARIÁVEIS DE CLASSE ...

```
class Refs {
  public static void main(String [] args){
     Ponto2 esquerdo = Ponto2.Origem;
     Ponto2 direito = new Ponto2(); Ponto2 z = direito;
     esquerdo.x  = 15; direito.x = 10.0; direito.y = 20.0;
     z.clear(); esquerdo.clear();
  }
}
```

VARIÁVEIS DE CLASSE X `this`

```
class A{
  static int x = 100;
  static int f() {
     return this.x;
  }
}
public class Teste {
  public static void main(String[] args){
     System.out.println("f = " + A.f());
  }
}

Mensagem: "Undefined Variable this"
```

VARIÁVEIS DE CLASSE X this...

```
class B{
  static int x = 100;
  static int f() {
    return x;
  }
}
public class Teste {
  public static void main(String[] args){
     System.out.println("f = " + B.f());
  }
}
Saida: f = 100
```


VARIÁVEIS DE CLASSE X this ...

```
public class C{
  public static void main(String[] args){
    A a = new A();
    System.out.println("A.x = " + a.x);
    System.out.println("f   = " + a.f());
   }
}
class A{
    static int x = 100;
    public int f() { return this.x; }
}
Saida: A.x = 100
       f   = 100
```

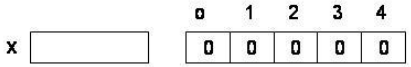

ARRANJOS DE OBJETOS

CRIAÇÃO E MANIPULAÇÃO DE ARRANJO

```
class A {
   int r, s;
   A(int d, int e){r=d;s=e;}
}
class B {
 public ... main(String[] a) {
   int[ ] x = new int[5];
   x[2] = x[1] + 1;
   A [ ]  y = new A[5];
   y[2] = new A(1,2);
   y[5] = new A(4,6);
 }
}
```

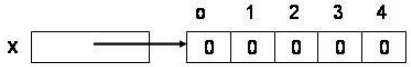
CRIAÇÃO E MANIPULAÇÃO DE ARRANJO ...

```
class A {
   int r, s;
   A(int d, int e){r=d;s=e;}
}
class B {
 public ... main(String[] a) {
   int[ ] x = new int[5];
   x[2] = x[1] + 1;
   A [ ]  y = new A[5];
   y[2] = new A(1,2);
   y[4] = new A(4,6);
 }
}
```



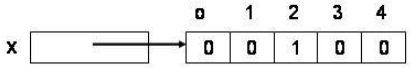
CRIAÇÃO E MANIPULAÇÃO DE ARRANJO ...

```
class A {
   int r, s;
   A(int d, int e){r=d;s=e;}
}
class B {
 public ... main(String[] a) {
   int[ ] x = new int[5];
   x[2] = x[1] + 1;
   A [ ]  y = new A[5];
   y[2] = new A(1,2);
   y[4] = new A(4,6);
 }
}
```



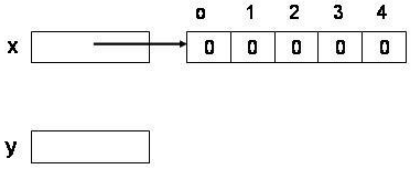
CRIAÇÃO E MANIPULAÇÃO DE ARRANJO ...

```
class A {
   int r, s;
   A(int d, int e){r=d;s=e;}
}
class B {
 public ... main(String[] a) {
   int[ ] x = new int[5];
   x[2] = x[1] + 1;
   A [ ]  y = new A[5];
   y[2] = new A(1,2);
   y[4] = new A(4,6);
 }
}
```



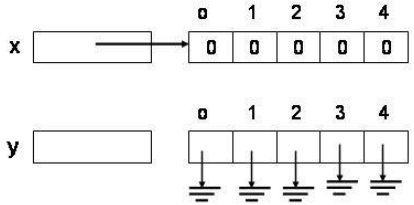
CRIAÇÃO E MANIPULAÇÃO DE ARRANJO ...

```
class A {
   int r, s;
   A(int d, int e){r=d;s=e;}
}
class B {
 public ... main(String[] a) {
   int[ ] x = new int[5];
   x[2] = x[1] + 1;
   A [ ]  y = new A[5];
   y[2] = new A(1,2);
   y[4] = new A(4,6);
 }
}
```



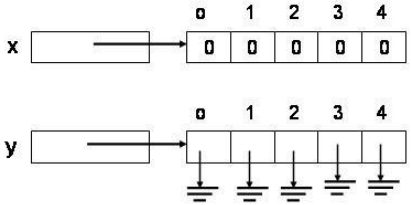
CRIAÇÃO E MANIPULAÇÃO DE ARRANJO ...

```
class A {
   int r, s;
   A(int d, int e){r=d;s=e;}
}
class B {
 public ... main(String[] a) {
   int[ ] x = new int[5];
   x[2] = x[1] + 1;
   A [ ]  y = new A[5];
   y[2] = new A(1,2);
   y[4] = new A(4,6);
 }
}
```



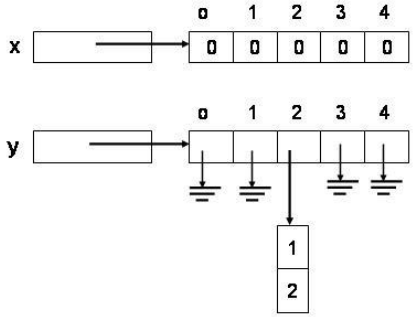
CRIAÇÃO E MANIPULAÇÃO DE ARRANJO ...

```
class A {
   int r, s;
   A(int d, int e){r=d;s=e;}
}
class B {
 public ... main(String[] a) {
   int[ ] x = new int[5];
   x[2] = x[1] + 1;
   A [ ]  y = new A[5];
   y[2] = new A(1,2);
   y[4] = new A(4,6);
 }
}
```



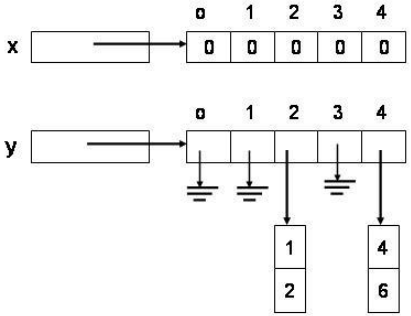
CRIAÇÃO E MANIPULAÇÃO DE ARRANJO ...

```
class A {
   int r, s;
   A(int d, int e){r=d;s=e;}
}
class B {
 public ... main(String[] a) {
   int[ ] x = new int[5];
   x[2] = x[1] + 1;
   A [ ]  y = new A[5];
   y[2] = new A(1,2);
   y[4] = new A(4,6);
 }
}
```



CRIAÇÃO E MANIPULAÇÃO DE ARRANJO ...

```
class A {
   int r, s;
   A(int d, int e){r=d;s=e;}
}
class B {
 public ... main(String[] a) {
   int[ ] x = new int[5];
   x[2] = x[1] + 1;
   A [ ]  y = new A[5];
   y[2] = new A(1,2);
   y[4] = new A(4,6);
 }
}
```



INICIAÇÃO DE OBJETOS

INICIAÇÃO DE CAMPOS E VARIÁVEIS

INICIAÇÃO DE CAMPOS E VARIÁVEIS

- Variáveis de classe, de instância e componentes de arranjos são iniciados com valores default no momento em que são criados.
- Parâmetros são iniciados com o argumento correspondente.
- Variáveis locais não são iniciadas automaticamente.
- Variáveis locais devem ser explicitamente iniciadas pelo programador de uma forma verificável pelo compilador (4.5.4).

INICIAÇÃO DE CAMPOS

- Campos de objetos podem ser iniciados de cinco maneiras:
  - By default após criação do objeto
  - Iniciadores na declaração
  - Blocos de Inicialização de Estáticos (variáveis de classe)
  - Blocos de Inicialização de Não-Estáticos (campos do objeto)
  - Funções Construtoras
- Campos estáticos são inicializados quando a classe que os contém for usada.
- Campos não-estáticos são inicializados quando o objeto correspondente for criado.
- Campos de classes são inicializados com valores default.

INICIAÇÃO COMPLEXA DE CAMPOS

- A inicialização complexa de campos pode ser feita via blocos de inicialização e funções construtoras.
- Os blocos de inicialização podem ser de estáticos e de não-estáticos.
- Blocos de inicialização de estáticos são executados quando a classe for carregada e logo após a execução as inicializações indicadas nas declaração de estáticos.
- Se a classe tiver mais de um bloco de inicialização, todos serão executados em ordem de ocorrência, dentro de sua categoria.
- Blocos de inicialização de não-estáticos são executados antes de executar o corpo da construtora, mas depois das inicializações contidas nas declarações.

INICIALIZADORES NÃO-ESTÁTICOS

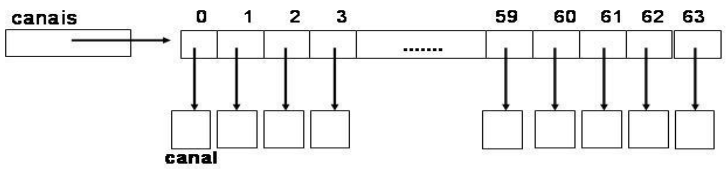
```
public class A {
  int x = 10;
  {x += 100; System.out.print(" x1 = " + x);}
  public A() {
    x += 1000; System.out.print(" x2 = " + x);
  }
  public A(int n) {
     x += 10000; System.out.print(" x4 = " + x);
  }
  public static void main(String[ ] args) {
     A a = new A( ) ; A b = new A(1) ;
  }
}
Saída: x1 = 110 x2 = 1110 x1 = 110 x4 = 10110
```


INICIALIZADORES ESTÁTICOS

```
class Canal{
   public Canal(int ch) { .. };  ...
}
class ConexaoTelefonica {
   int t;
   static final int MAXCANAIS = 64;
   static Canal canais[ ] = new Canal[MAXCANAIS];
   "Inicialização de canais";

   static final int MAXRESERVAS = 10;
   static Canal canaisReserva[ ] = new Canal[MAXRESERVAS];
   "Inicialização de canaisReserva";
}
```


INICIALIZACAO DE canais

```
static {
  for (int i = 0; i < MAXCANAIS; i ++) {
      canais[i] = new Canal(i);
  }
}
```



INICIALIZAÇÃO DE canaisReserva

```
static {
  for (int i = 0; i < MAXRESERVAS; i ++){
      canaisReserva[i] = new Canal(i);
  }
}
```


MOMENTO DA INICIALIZAÇÃO

```
class A {
   public static int r = X.f('A'), v = X.f('A');
   static {r = 100; System.out.print(" r = " + r);}
   {r = 1000; System.out.print(" r = " + r);}
   public A() { r = 10000; System.out.print(" r = " + r);}
}

class B {
   public static int s = X.f('B'),  w = X.f('B');
   static {s = 100; System.out.print(" s = " + s);}
   {s = 1000; System.out.print(" s = " + s);}
   public B(){s = 10000; System.out.print(" s = " + s);}
}
```


MOMENTO DA INICIALIZAÇÃO ...

```
class C {
   public static int t = X.f('C'), z = X.f('C');
   static {t = 100; System.out.print(" t = " + t);}
   {t = 1000; System.out.print(" t = " + t);}
   public C(){t = 10000; System.out.print(" t = " + t);}
}

class D {
   public static int u = X.f('D'), k = X.f('D');
   static {u = 100; System.out.print(" u = " + u);}
   {u = 1000; System.out.print(" u = " + u);}
   public D(){u = 10000; System.out.print(" u = " + u);}
}
```


MOMENTO DA INICIALIZAÇÃO ...

```
class X {
  public static int f(char m){
     System.out.print(" " + m);
     return 1;
  }
}
```

MOMENTO DA INICIALIZAÇÃO ...

```
public class LoadClass {
   public static void main(String[] args){
     PrintStream o = System.out;
     o.print(" P1"); A a = new A(); o.print(" P2"); B b;
     o.print(" P3"); A c = new A(); o.print(" P4"); B d;
     o.print(" P5"); int x = B.s;   o.print(" P6"); int y = B.s;
     o.print(" P7"); B e = new B(); o.print(" P8");
     if (args.length==0) {C g = new C();} else {D g = new D();};
     o.print(" Acabou");
   }
} // Para uma execução java LoadClass xx, têm-se:
P1 A A  r = 100 r = 1000 r = 10000 P2 P3 r = 1000 r = 10000
P4 P5 B B  s = 100  P6 P7 s = 1000 s = 10000
P8 D D u =100 u = 1000 u = 10000 Acabou
```


EXEMPLO

CONTA EM BANCO

```
public class Conta {
  static private double taxa_anual;
  private double saldo_corrente;
  public static double taxa(){return taxa_anual;}
  public static void def_taxa(double valor){
     if (valor > 0.0 && valor < 12.0) {
        taxa_anual = valor;
     }
  }
  public void creditar(double q){...}
  public void debitar(double q){...}
  public double saldo(){...}
}
```


CONTA EM BANCO ...

```
public void creditar(double q){
   if (q > 0.0) {
      saldo_corrente += q;
   }
}
public void debitar(double q){
   if (q > 0.0 && q <= saldo_corrente){
      saldo_corrente -= q;
   }
}
public double saldo(){
   return saldo_corrente;
}
```

CONTA EM BANCO ...

```
public class Aplicacao {
  public static void main(String args[]){
    double deposito = (double)Integer.parseInt(args[0]);
    double taxa_anual = (double)Integer.parseInt(args[1]);
    Conta c = new Conta();
    double juros;
    Conta.def_taxa(taxa_anual);
    System.out.println(
       "Depositando " + deposito +
       "por mes a taxa de " + Conta.taxa());
    "CALCULA EVOLUÇÃO DO SALDO";
}
```


CALCULA EVOLUÇÃO DO SALDO

```
System.out.println("Veja a Evolução:");
for (int ano = 0 ; ano <10; ano++){
    System.out.print(ano + " - ");
    for (int mes = 0; mes < 12; mes++){
        juros = c.saldo()*Conta.taxa()/1200.0;
        c.creditar(juros);
        c.creditar(deposito);
        System.out.print(c.saldo() + ", ");
    }
    System.out.println();
}
```


FIM

AMBIENTES DE PROGRAMAÇÃO
ALGUMAS CLASSES BÁSICAS

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

19 de março de 2015

SUMÁRIO

- Classe Math
- Classe String
- Classes Invólucro
- Entrada e Saída com Mylo
- Entrada e Saída do Swing
- Entrada e Saida de Fluxo
- Implementação de Mylo

CLASSE `MATH`

CLASSE `Math`

- Os argumentos em radianos e todos os valores são `double`.

Função	Valor
`Math.PI`	π
`Math.E`	e neperiano
`Math.sin(a)`	seno de `a`
`Math.cos(a)`	co-seno de `a`
`Math.tan(a)`	tangente de `a`
`Math.asin(v)`	arco-seno de `v` $\in [-1.0, 1.0]$
`Math.acos(v)`	arcoco-seno de `v` $\in [-1.0, 1.0]$
`Math.atan(v)`	arcotang(`v`) (ret. em $[-\pi/2, \pi/2]$)
`Math.atan2(x,y)`	arcotang(`x/y`) (ret. em $[-\pi, \pi]$)

CLASSE `Math`...

- **Os argumentos em radianos e todos os valores são** `double`**.**

Função	Valor
`Math.exp(x)`	e^x
`Math.pow(y,x)`	y^x
`Math.log(x)`	$\ln x$
`Math.sqrt(x)`	$\sqrt{x}$
`Math.ceil(x)`	**menor inteiro** $\geq$ `x`
`Math.floor(x)`	**maior inteiro** $\leq$ `x`
`Math.rint(x)`	**valor truncado de** `x`
`Math.round(x)`	`(int)floor(x+0.5)`
`Math.abs(x)`	**valor absoluto de** `x`
`Math.max(x,y)`	**máximo de** `x` **e** `y`
`Math.min(x,y)`	**mínimo de** `x` **e** `y`

CLASSE STRING

CONSTANTES `String`

- **Constantes do tipo String são seqüências de caracteres UNICODE entre** `"`**:**

```
""   " "   "\n"   "\r"   "\""
"um string"
"uma parte" + "outra parte"
```

- **O tipo de uma constante string é o de uma referência a objetos do tipo** `String`**.**
- **O operador** `+` **concatena** `String`**s.**

CONSTANTES STRINGS

- **Quando um dos operandos do** `+` **é um** `String` **o outro operando pode ser um objeto que tenha a operação** `toString()`**:**

```
A x = new A();
String s = "Seu objeto e\' " + x;
```

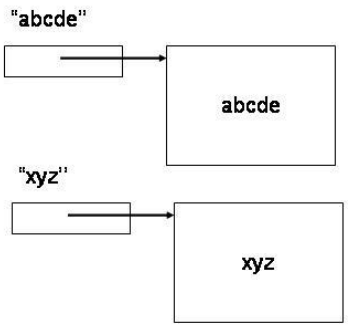
é o mesmo que

```
String s = "Seu objeto e\' " + x.toString();
```

- **Expressões de constantes strings são avaliadas em tempo de compilação e valores idênticos produzem strings de mesma referência, isto é, mesmo objeto.**
- **Strings construídos** durante a execução **são sempre novos e distintos de qualquer outro string, mesmo quando representam o mesmo valor.**

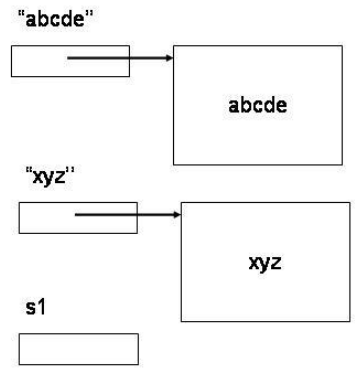
CONSTANTE STRING

```
public class Teste {
  public static void
           main(String[] args){
    String s1;
    s1 = "xyz";
    System.out.print("abcde" +
                     s1 + "xyz");
  }
}
```



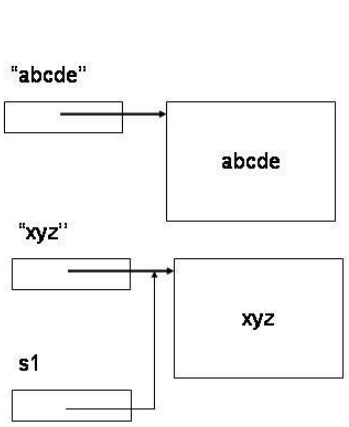
CONSTANTE STRING ...

```
public class Teste {
  public static void
           main(String[] args){
    String s1;
    s1 = "xyz";
    System.out.print("abcde" +
                     s1 + "xyz");
  }
}
```



CONSTANTE STRING ...

```
public class Teste {
  public static void
           main(String[] args){
    String s1;
    s1 = "xyz";
    System.out.print("abcde" +
                     s1 + "xyz");
  }
}
```



Constante String É UMA REFERÊNCIA

```
public class C {
  public static void main(String[ ] args) {
      String x = null;
      if( "abc".equals("abc") )
           System.out.println("Funcionou");
      else System.out.println("Não Funcionou");
      if( x.equals("abc") )
           System.out.print("Iguais ");
      else System.out.print("Diferentes");
  }
}
Funcionou
Exception in thread "main" java.lang.NullPointerException
at C.main(C.java:7)
```

Constante String É UMA REFERÊNCIA ...

```
public class C {
  public static void main(String[ ] args) {
      String x = null;
      if( "abc".equals("abc") )
           System.out.println("Funcionou");
      else System.out.println("Não Funcionou");
      if( "abc".equals(x) )
           System.out.print("Iguais ");
      else System.out.print("Diferentes");
  }
}
Saída: Funcionou
       Diferentes
```


INTRODUÇÃO À CLASSE `String`

- **Classe** `String` **não possui métodos para modificar o estado do objeto corrente.**
- **Objetos da classe** `String` **são read-only, i.e., constantes**
- **Objetos da classe** `StringBuffer` **são mutáveis.**
- **Constantes** `String`**s com mesmo conteúdo denotam mesmo objeto.**
- **Constantes strings de mesmo conteúdo têm a mesma referência.**
- **Classe** `String` **é** `final`**.**

OPERAÇÕES BÁSICAS COM STRINGS

- `public String( )`:
  contrói novo string vazio.
  ```
  String s = new String();
  ```
- `public String(String value)`:
  constrói um novo string cópia de `value`**.**
  ```
  String s = new String("abcde");
  String s = "abcde";
  ```
- `public String(char value[])`:
  constrói novo string a partir de um vetor de `char`**.**
- `public int length()`:
  retorna o número de caracteres string this**.**
  ```
  String s = "abcde";    int n = s.length();
  ```


OPERAÇÕES BÁSICAS COM STRINGS ...

- `public char charAt(int index)`:
  retorna o caractere na posição `index`**.**
  ```
  int i= 10, k = s.charAt(i);
  ```

```
String str = new String(...);
char c;
...
int [] counts = new int[256];
for (int i=0; i < str.length(); i++) {
     c = str.charAt(i);
     if (c<256) counts[c]++;
}
```

LOCALIZAÇÃO DE CHAR DENTRO DO STRINGS

- `public int indexOf(char ch)`:
  retorna a posição (≥ 0) da primeira ocorrência do caractere `ch` **no string** this.
  Retorna `-1` **se não achar.**

```
s= "01234a678901234a678901234a6789a";
int k = s.indexOf('a');
```

- `public int indexOf(char ch, int start)`:
  retorna a posição $\geq$ `start` **da primeira ocorrência do caractere** `ch` **no string** this.

```
int k = s.indexOf('a',10);
```


LOCALIZAÇÃO DE CHAR DENTRO DO STRINGS

- `public int lastIndexOf(char ch)`:
  retorna a posição da última ocorrência de `ch` **no string** this.

```
s= "01234a678901234a678901234a6789a";
int k = s. lastIndexOf('a');
```

- `public int lastIndexOf(char ch,int start)`:
  retorna a posição $\leq$ `start` **da última ocorrência de** `ch` **no string** this.

```
int k = s.lastIndexOf('a',10);
```

```
static int NumCharsEntre(String str; char ch) {
  int inicio = str.indexOf(ch);
  if (inicio < 0) return -1;
  int fim = str.lastIndex(ch);
  return fim - inicio - 1;
}
```


LOCALIZAÇÃO DE STRING DENTRO DO STRINGS...

- `public int indexOf(String str)`:
  retorna posição da primeira ocorrência do string `str` **no string** this.

```
String s1 = "abcxxxghxxxjklm", s2 = "xxx";
int k = s1.indexOf(s2); // k = 3
```

- `public int indexOf(String str, int start)`:
  retorna posição $\geq$ `start` **da primeira ocorrência de** `str` **em** this.
- `public int lastIndexOf(String str)`:
  retorna posição da última ocorrência do string `str` **no string** this.
- `public int lastIndexOf(String str, int start)`:
  retorna posição $\leq$ `start` **da última ocorrência de** `str` **em** this.

DIVISÃO DE STRING

- `public String[ ] split(String separador)`:
  divide o string separados por elementos da expressão regular dada.
  Exemplos de expressão regular: `"[ ]"`, `"[abc;]"`, `"[a-z0-9]"`

```
public static void main(String[ ] args) {
    String a = "abc cde,efg hij";
    String s[];
    s = a.split("[ ,]");
    for (int i = 0; i < s.length; i++ )
         System.out.print(s[i] + "-");
}
```

produz : `abc-cde-efg-hij`

COMPARAÇÃO DE STRINGS

- `public boolean equals(Object x)`:
  retorna `true` **se o string** this **tiver o mesmo conteúdo que string** `x`.
  `boolean b = s1.equals(s2);`
- `public boolean equalsIgnoreCase(Object x)`:
  retorna `true` **se o string** this **tiver o mesmo conteúdo que string** `x`, **considerando maiúsculo igual a minúsculo.**
- `public int compareTo(String s)`:
  retorna inteiro menor que zero, zero ou maior que zero conforme o string this **seja menor, igual ou maior que o string** `s`, **respectivamente.**
  `int k = s1.compareTo(s2);`
- `==`:
  retorna `true` **se os operandos contêm a mesma referência.**

PESQUISA EM TABELA DE STRINGS

```
class Tabela {
   private final static int MAX = 100;
   private String[] tabela = new String[MAX];
   private int tamanho = 0;
   public int inclui(String chave) {
      for (int i=0; i < tamanho; i++)
          if (tabela[i].equals(chave)) return i;
      if (tamanho == MAX ) throw new Estouro();
      tabela[tamanho++] = chave;
      return tamanho - 1;
   }
 ...
}
```


POOL DE String

- **Classe** `String` **contém um área interna de objetos** `String`, **onde string iguais são sempre o mesmo objeto. No** *pool* **não há** `String`**s duplicados.**
- **A função** `String intern()` **retorna a referência de uma cópia de objeto** `this` **na área de** *pool*. **Se cópia não existia, a cópia é feita.**
- **Note que:**
  `String s1, s2;` **...**
  `s1 == s2`: `true` **se referências iguais**
  `s1.intern() == s2.intern()`: `true` **se conteúdos iguais**

PESQUISA EM TABELA DE STRINGS

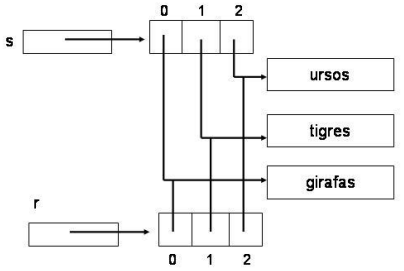
```
class Tabela {
   private final static int MAX = 100;
   private String[] tabela = new String[MAX];
   private int tamanho = 0;
   public int inclui(String chave){
      String x = chave.intern();
      for (int i=0; i < tamanho; i++)
          if (tabela[i] == x) return i;
      if (tamanho == MAX ) throw new Estouro();
      tabela[tamanho++] = x;
      return tamanho - 1;
   }
 ...
}
```

PESQUISA BINÁRIA EM TABELA DE STRINGS

```
class Y {
   private String[] tabela;
   public int pesquisa(String chave) {
      int inf = 0; int sup = tabela.length -1;
      while (inf <= sup) {
         int meio = inf + (sup - inf)/2;
         int comp = chave.compareTo(tabela[meio]);
         if (comp == 0) return meio;
         else if (comp < 0) sup = meio-1; else inf = meio+1;
      }
      return -1;
   }
   ...
}
```

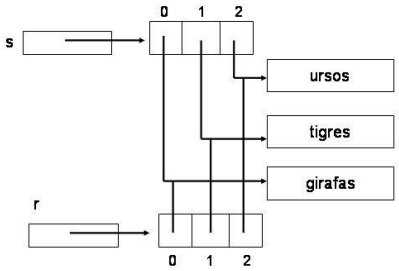

ARRANJOS DE STRINGS

```
String[ ] s = {"girafas", "tigres", "ursos" };
String[ ] r = new String[3];
r[0] = "girafas"; r[1] = "tigres"; r[2] = "ursos";
```



ARRANJOS DE STRINGS ...

```
String[ ] s = new String[ ]{"girafas", "tigres", "ursos"};
String[ ] r = new String[3];
r[0] = "girafas"; r[1] = "tigres"; r[2] = "ursos";
```



CLASSES INVÓLUCRO

CLASSES DE TIPOS BÁSICOS

- **Todo tipo básico tem uma classe correspondente:**

boolean	**Boolean**
char	**Character**
byte	**Byte**
short	**Short**
int	**Integer**
long	**Long**
float	**Float**
double	**Double**

- **Valores de tipo básicos podem ser** empacotados
- **Objeto empacotado (wrapped) não pode ser modificado**

CLASSES DE TIPOS BÁSICOS...

```
int i = 50;
int j;
Integer x;
float f = 100.0;
float y;
Float z;
x = new Integer(i);
j = x.intValue();
z = new Float(f);
y = z.FloatValue();
```


CLASSE `java.lang.Boolean`

- **Objetos do tipo** `Boolean` **representam valores do tipo** `boolean`**.**

```
public final class Boolean {
   public static final Boolean TRUE  = new Boolean(true);
   public static final Boolean FALSE = new Boolean(false);
   public Boolean(boolean value);
   public Boolean(String s);
   public String toString();
   public boolean equals(Object obj);
   public int hashCode();
   public boolean booleanValue();
   public static Boolean valueOf(String s);
   public static boolean getBoolean(String name);
}
```


CLASSE `java.lang.Character`

- **Objetos do tipo** `Character` **representam valores primitivos do tipo** `char`**.**

```
public final class Character {
   public Character(char value);
   public static boolean isLowerCase(char ch);
   public static boolean isUpperCase(char ch);
   public static boolean isTitleCase(char ch);
   public static boolean isDigit(char ch);
   public static boolean isLetter(char ch);
   public static boolean isLetterOrDigit(char ch);
   public static boolean isJavaLetter(char ch);
   public static boolean isJavaLetterOrDigit(char ch);
   public static boolean isSpace(char ch);
```

CLASSE java.lang.Character...

```
   public String toString();
   public boolean equals(Object obj);
   public char charValue();
   public static char toLowerCase(char ch);
   public static char toUpperCase(char ch);
   public static char toTitleCase(char ch);

   public static int digit(char ch,int radix);
   public static char forDigit(int digit, int radix);
}
```


CLASSE java.lang.Number

- **A classe abstrata** `Number` **tem subclasses** `Integer`, `Long`, `Float` **e** `Double`.
- **As subclasses empacotam os tipos primitivos correspondentes.**
- **Cada subclasse define o método para converter o valor numérico representado pelo tipo correspondente, isto é,** `int`, `long`, `float` **ou** `double`.

CLASSE java.lang.Number...

```
public abstract class Number {
  public abstract int intValue();
  public abstract long longValue();
  public abstract float floatValue();
  public abstract double doubleValue();
}
```


CLASSE java.lang.Integer

```
public final class Integer extends Number {
 public static final int MIN_VALUE = 0x80000000;
 public static final int MAX_VALUE = 0x7fffffff;
 public Integer(int value);
 public Integer(String s) throws NumberFormatException;
 public String toString();
 public boolean equals(Object obj);
 public int hashCode();
 public int intValue();
 public long longValue();
 public float floatValue();
 public double doubleValue();
```

CLASSE java.lang.Integer...

```
public static String toString(int i);
public static String toString(int i, int radix);
public static String toHexString(long i);
public static String toOctalString(long i);
public static String toBinaryString(long i);
```


CLASSE java.lang.Integer...

```
public static int parseInt(String s)
       throws NumberFormatException;
public static int parseInt(String s,
       int radix) throws NumberFormatException;
public static Integer valueOf(String s)
       throws NumberFormatException;
public static Integer valueOf(String s,
       int radix) throws NumberFormatException;
public static Integer getInteger(String nm);
public static Integer getInteger(String nm, int val);
public static Integer getInteger(String nm,Integer val);
}
```


CLASSE java.lang.Long

```
public final class Long extends Number {
 public static final long MIN_VALUE = 0x8000000000000000L;
 public static final long MAX_VALUE = 0x7fffffffffffffffL;
 public Long(long value);
 public Long(String s) throws NumberFormatException;

 public boolean equals(Object obj);
 public int hashCode();
 public int intValue();
 public long longValue();
 public float floatValue();
 public double doubleValue();
```


CLASSE java.lang.Long

```
public String toString();
public static String toString(long i);
public static String toString(long i, int radix);
public static String toHexString(long i);
public static String toOctalString(long i);
public static String toBinaryString(long i);
```

CLASSE java.lang.Long...

```
 public static long parseLong(String s)
                throws NumberFormatException;
 public static long parseLong(String s,
     int radix) throws NumberFormatException;
 public static Long valueOf(String s)
                throws NumberFormatException;
 public static Long valueOf(String s,
     int radix) throws NumberFormatException;
 public static Long getLong(String nm);
 public static Long getLong(String nm, long val);
 public static Long getLong(String nm, Long val);
}
```


CLASSE java.lang.Float

```
public final class Float extends Number {
 public static final float MIN_VALUE = 1.4e-45f;
 public static final float MAX_VALUE = 3.4028235e+38f;
 public static final float NEGATIVE_INFINITY = -1.0f/0.0f;
 public static final float POSITIVE_INFINITY = 1.0f/0.0f;
 public static final float NaN = 0.0f/0.0f;
 public Float(float value);
 public Float(double value);
 public Float(String s) throws NumberFormatException;
```


CLASSE java.lang.Float

```
 public String toString();
 public boolean equals(Object obj);
 public int hashCode();
 public int intValue();
 public long longValue();
 public float floatValue();
 public double doubleValue();
 public static String toString(float f);
```


CLASSE java.lang.Float...

```
 public static float parseFloat(String s)
        throws NumberFormatException;
 public static Float valueOf(String s)
        throws NullPointerException, NumberFormatException;
 public boolean isNaN();
 public static boolean isNaN(float v);
 public boolean isInfinite();
 public static boolean isInfinite(float v);
 public static int floatToIntBits(float value);
 public static float intBitsToFloat(int bits);
}
```

CLASSE java.lang.Double

```
public final class Double extends Number {
 public static final double MIN_VALUE = 5e-324;
 public static final double MAX_VALUE
                          = 1.7976931348623157e+308;
 public static final double NEGATIVE_INFINITY = -1.0/0.0;
 public static final double POSITIVE_INFINITY = 1.0/0.0;
 public static final double NaN = 0.0/0.0;
 public Double(double value);
 public Double(String s)throws NumberFormatException;
```


CLASSE java.lang.Double

```
public String toString();
public boolean equals(Object obj);
public int hashCode();
public int intValue();
public long longValue();
public float floatValue();
public double doubleValue();
public static String toString(double d);
```


CLASSE java.lang.Double...

```
 public static double parseDouble(String s)
        throws NumberFormatException;
 public static Double valueOf(String s)
        throws NullPointerException, NumberFormatException;
 public boolean isNaN();
 public static boolean isNaN(double v);
 public boolean isInfinite();
 public static boolean isInfinite(double v);
 public static long doubleToLongBits(double value);
 public static double longBitsToDouble(long bits);
}
```


ENTRADA E SAÍDA COM MYIO

CLASSES DO PACOTE `MyIo`

- `class Kbd`
- `class InText`
- `class Screen`
- `class OutText`

ENTRADA DE TECLADO `MyIo`

```
public class Kbd {
 final static public boolean readBoolean() throws IOException;
 final static public int readInt() throws IOException;
 final static public long readLong() throws IOException;
 final static public float readFloat() throws IOException;
 final static public double readDouble() throws IOException;
 final public double readString() throws IOException;
}
```


ENTRADA DE TECLADO `MyIo`...

```
import java.io.IOException;
import MyIo.Kbd;

public class A {
  public static void main(String[ ] args) {

     int x = Kbd.readInt();

     float f = Kbd.readFloat();

     ...
  }
}
```


ENTRADA DE ARQUIVO DE CHAR `MyIo`...

```
public class InText {
   public InText(String fileName) throws IOException;
   final public boolean readBoolean() throws IOException;
   final public int readInt() throws IOException;
   final public long readLong() throws IOException;
   final public float readFloat() throws IOException;
   final public double readDouble() throws IOException;
   final public double readString() throws IOException;
}
```

- **Separador dos dados é um dos caracteres em** `" \t\n\r\f"`

ENTRADA DE ARQUIVO DE CHAR MyIo...

```
import java.io.IOException;
import MyIo.InText;

public class A {
   public static void main(String[ ] args) {

      InText in = new InText("arquivo");

      int i = in.readInt();
      double d = in.readDouble();
      ...
   }
}
```


SAÍDA PARA TELA MyIo

```
public class Screen {
   final static public void println();
   final static public void print(String s);
   final static public void print(boolean b);
   final static public void print(byte b);
   final static public void print(char c);
   final static public void print(int i);
   final static public void print(long n);
   final static public void print(float f);
   final static public void print(double d);
   final static public void println(T d);
}
```


SAÍDA PARA TELA MyIo ...

```
import MyIo.Screen;
public class A {
   public static void main(String[ ] args) {
      int i = 10;

      Screen.println(i);

      Screen.print("Um texto");
   }
}
```


SAÍDA PARA ARQUIVO CHAR MyIo

```
public class OutText {
  public OutText(String fileName) throws IOException;
  final public void println();
  final public void print(String s);
  final public void print(boolean b);
  final public void print(byte i);
  final public void print(short i);
  final public void print(int i);
  final public void print(long n);
  final public void print(float f);
  final public void print(double d);
  final public void println(T d);
}
```

SAÍDA PARA ARQUIVO CHAR MyIo ...

```
import java.io.IOException;
import MyIo.OutText;
public class A {
   public static void main(String[ ] args) {
      int i = 10, k = 11;

      OutText o = new OutText("arquivo");

      o.println();
      o.println(i);
      o.print(k);
   }
}
```


ENTRADA E SAÍDA GRÁFICAS ELEMENTARES DO SWING

Classe JOptionPane

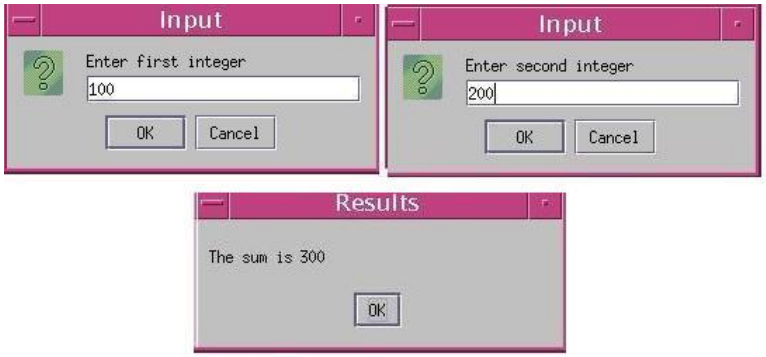
```
import javax.swing.JOptionPane;
public class Dialogo1 {
   public static void main( String args[] ) {
      JOptionPane.showMessageDialog(
         null, "Welcome\nto\nJava\nProgramming!");
      System.exit( 0 );
   }
}
```

DIÁLOGO DE E/S

```
import javax.swing.JOptionPane;
public class Addition {
  public static void main( String[ ] args) {
    String s1, s2; int sum;
    s1=JOptionPane.showInputDialog("Enter first integer");
    s2=JOptionPane.showInputDialog("Enter second integer");
    sum = Integer.parseInt(s1) + Integer.parseInt(s2);
    JOptionPane.showMessageDialog(
       null, "The sum is " + sum, "Results",
       JOptionPane.PLAIN_MESSAGE);
    System.exit( 0 );
  }
}
```

EXECUÇÃO DE `Addition`

```
>java Addition
```



MÉTODO `JOptionPane.showMessageDialog`

`JOptionPane.showMessageDialog(local,texto,titulo,icone)`:

- `local` : **Define o componente no qual a caixa de diálogo será colocada. Se `null`, a caixa de diálogo é apresentada usualmente no centro da tela.**
- `texto`: **Texto a ser exibido**
- `titulo`: **Texto da Barra de Titulo**
- `icone`: **icone da Mensagem:**
  - `JOptionPane.ERROR_MESSAGE`
  - `JOptionPane.INFORMATION_MESSAGE`
  - `JOptionPane.WARNING_MESSAGE`
  - `JOptionPane.QUESTION_MESSAGE`
  - `JOptionPane.PLAIN_MESSAGE`

LEITURA DE STRINGS SEPARADOS POR ,

```
import java.io.*; import java.util.*;
public class Io {
  public static void main (String[ ] args) throws Exception {
    String [] a;
    DataInputStream in;
    in = new DataInputStream(new FileInputStream("str.txt"));
    String s = in.readLine();
    a = s.split("[,]"); // divide strings separados por ,
    for(int i=0; i<a.length;i++)System.out.print(a[i] + "--");
  }
}
Entrada: ab1c cd2ef,ghi jk3mnop,qrs4tuvz !@#$5%^&*() ZZZ
Saida  : ab1c cd2ef--ghi jk3mnop--qrs4tuvz !@#$%^&*() ZZZ--
```


LEITURA DE STRINGS SEPARADO POR BRANCO

```
import java.io.*; import java.util.*;
public class Io {
  public static void main (String[ ] args) throws Exception {
    String [] a ;
    DataInputStream infile;
    in = new DataInputStream(new FileInputStream("str.txt"));
    String s = in.readLine();
    a = s.split("[ ]"); // divide strings separados por branco
    for(int i=0; i<a.length;i++) System.out.print(a[i] + "--");
  }
}
Entrada: ab1c cd2ef,ghi jk3mnop,qrs4tuvz !@#$5%^&*() ZZZ
Saida  : ab1c--cd2ef,ghi--jk3mnop,qrs4tuvz--!@#$5%^&*()--ZZZ--
```

LEITURA DE STRINGS SEPARADO POR BRANCO ou ,

```
import java.io.*; import java.util.*;
public class Io {
  public static void main (String[ ] args) throws Exception {
    String [] a;
    DataInputStream infile;
    in = new DataInputStream(new FileInputStream("str.txt"));
    String s = in.readLine();
    a = s.split("[ ,]"); // divide str separados por br ou ,
    for(int i=0; i<a.length;i++) System.out.print(a[i] + "--");
  }
}
Entrada: ab1c cd2ef,ghi jk3mnop,qrs4tuvz !@#$5%^&*() ZZZ
Saida  : ab1c--cd2ef--ghi--jk3mnop--qrs4tuvz--!@#$5%^&*()--ZZZ--
```


LEITURA DE STRINGS SEPARADO POR DIGITOS

```
import java.io.*; import java.util.*;
public class Io {
  public static void main (String[ ] args) throws Exception {
    String [] a;
    DataInputStream infile;
    in = new DataInputStream(new FileInputStream("str.txt"));
    String s = in.readLine();
    a = s.split("[0-9]"); // divide str separados por dígitos
    for(int i=0; i<a.length;i++) System.out.print(a[i] + "--");
  }
}
Entrada: ab1c cd2ef,ghi jk3mnop,qrs4tuvz !@#$5%^&*() ZZZ
Saida  : ab--c cd--ef,ghi jk--mnop,qrs--tuvz !@#$--%^&*() ZZZ--
```


ENTRADA E SAÍDA DE FLUXO DE BYTES

ENTRADA E SAÍDA DE FLUXO DE BYTES

```
public abstract class InputStream {
  public abstract int read( );
  public int read(byte [ ] b){ ...};
  public void close ( ){ ...};
  ...
}

public abstract class OutputStream {
  public abstrat void write(int);
  public void write(byte [ ] b){ ...};
  public void close (){ ...};
  ...
}
```

SAÍDA COM CONVERSÃO DE DADOS

```
public class FilterOutputStream extends OutputStream {...}

public class PrintStream extends FilterOutputStream {
  public void print(char    buf){ ... };
  public void print(int     buf){ ... };
  public void print(float   buf){ ... };
  public void print(boolean buf){ ... };
  public void print(String  buf){ ... };
  public void println(... buf){ ... }; ....
}
```


ENTRADA E SAÍDA PADRÃO

```
public class System {
  public static InputStream in;
  public static PrintStream out;
  public static PrintStream err;
}
```


ENTRADA COM CONVERSÃO DE DADOS

```
public class FileInputStream extends InputStream {
   FileInputStream(String nome){... }
   ...
}

public class FilterInputStream extends InputStream { ... }

public class DataInputStream extends FilterInputStream {
  public DataInputStream(InputStream in){ ... }
  public int readInt( ){ ...}
  public double readDouble( ){ ...}
  ....
}
```


USO DE `DataInputStream`

```
FileInputStream f = new FileInputStream("arquivo.dat");
DataInputStream g = new DataInputStream(f);
...
int k    = g.readInt();
double s = g.readDouble( );
```

ENTRADA DE TEXTO

```
public class Reader { ... }

public class BufferedReader extends Reader {
    public BufferedReader(Reader f){ ... }
    public String readLine( ){ ... }
    public char read( ) { ...}
    ....
}

public class InputStreamReader extends Reader {
   public InputStreamReader(InputStream in) { ... }
   ...
}
```


USE DE `BufferedReader`

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
public class ConsoleInputTest {
   public static void main(String[] args) throws IOException {
      Purse myPurse = new Purse();
      BufferedReader console = new BufferedReader(
         new InputStreamReader(System.in));
      "Interação para obter número de moedas"
      double totalValue = myPurse.getTotal( );
      System.out.println("The total is " + totalValue);
      System.exit(0);
   }
}
```


INTERAÇÃO PARA OBTER NÚMERO DE MOEDAS

```
System.out.println("Quantas moedas de 5 centavos?");
String input = console.readLine();
int count = Integer.parseInt(input);
myPurse.addNickels(count);

System.out.println("Quantas moedas de 10 centavos?");
input = console.readLine();
count = Integer.parseInt(input);
myPurse.addDimes(count);

System.out.println("Quantas moedas de 25 centavos?");
input = console.readLine();
count = Integer.parseInt(input);
myPurse.addQuarters(count);
```


IMPLEMENTAÇÃO DE `MYIO`

IMPLEMENTAÇÃO DE Kbd.java

```
package MyIo; import java.io.*; import java.util.*;
public class Kbd {
  private static DataInputStream infile =
               new DataInputStream(System.in);
  private static StringTokenizer st;

  final static public int readInt() throws IOException {
    if (st == null || !st.hasMoreTokens())
        st = new StringTokenizer(infile.readLine());
    return Integer.parseInt(st.nextToken());
  }
  ...
}
```


IMPLEMENTAÇÃO DE Screen.java

```
package MyIo;
public class Screen {
  final static public void print(String s){
     System.out.print(s);
  }
  final static public void print(char c){
     System.out.print(c);
  }
  final static public void print(int i){
     System.out.print(i);
  }
  ...
}
```


IMPLEMENTAÇÃO DE InText.java

```
package MyIo; import java.io.*; import java.util.*;
public class InText {
  private DataInputStream infile;
  private StringTokenizer st = null;
  public InText(String fileName) throws IOException {
    InputStream fin = new FileInputStream(fileName);
    infile = new DataInputStream(fin);
  }
  final public int readInt() throws IOException {
    if (st == null || !st.hasMoreTokens())
       st = new StringTokenizer(infile.readLine());
    return Integer.parseInt(st.nextToken());
  }  ...
}
```


IMPLEMENTAÇÃO DE OutText.java

```
package MyIo;
import java.io.*;
public class OutText {
  private PrintStream out;
  public OutText(String fileName) throws IOException {
    OutputStream fout = new FileOutputStream(fileName);
    out = new PrintStream(fout);
  }
  final public void print(String s){out.print(s);  }
  final public void println(String s){out.println(s);  }
  final public void print(byte i){out.print(String.valueOf(i));}
  final public void print(int i){out.print(String.valueOf(i));}
  ...
}
```

FIM

AMBIENTES DE PROGRAMAÇÃO
INTERFACE DE CLASSES

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

AMBIENTES DE PROGRAMAÇÃO
TIPOS ABSTRATOS DE DADOS

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

19 de março de 2015

SUMÁRIO

- Tipos Abstratos de Dados
- TAD BigInt

TIPOS ABSTRATOS DE DADOS

CONCEITO DE TAD

- Definição:
  TAD é uma estrutura de programa que encapsula uma determinada estrutura de dados e a torna conhecida somente pelas operações que se podem realizar sobre seus elementos, sem que se indiquem como a estrutura de dados e as operações são codificadas.

- Observações:
  - Constantes são consideradas operações sem parâmetros
  - o TAD totalmente é caracterizado pelas operações
  - o termo abstrato enfatiza que a estrutura dados do TAD não deve ser acessível pelo usuário do tipo.

TIPO ABSTRATO DE DADOS

- é o tipo de um objeto
- pode ser implementado por meio de classes
- descreve uma coleção de objetos
- é um molde para fazer objetos
- é caracterizado por suas operações
- a representação dos objetos do tipo é inacessível ao usuário do tipo
- as constantes de um TAD são consideradas operações sem parâmetros do tipo

CLASSE X TIPOS ABSTRATOS DE DADOS

- Classe é um mecanismo para implementar tipos abstratos de dados.
- Encapsulação é o agrupamento de estruturas de dados e funções dentro de uma classe e o respectivo controle de acesso aos membros da classe.
- Em todo TAD, a estrutura de dados deve ser formada por campos privados e somente as operações (métodos) devem ser públicas.
- Uma classe implementa um TAD somente se todos os seus atributos forem privados e somente seus métodos forem públicos.
- Toda classe cria um novo tipo, não necessariamente um TAD.

INTERFACE X TIPOS ABSTRATOS DE DADOS

- Interface é um mecanismo para especificar os tipos das operações de um TAD.
- A interface indica o que deve ser público no TAD, sem impor qualquer compromisso com como será implementado por uma ou mais classes.
- A interface define o contrato do TAD.

TIPO ABSTRATO DE DADOS `Ponto`

```
public interface Ponto{
  void clear( );
  void set(double a, double b);
  void hide( );
  void show( );
}
public class MeuPonto implements Ponto{
  private double x, y;
  public void clear( ) { x = 0; y = 0; }
  public void set(double a, double b) { x = a; y = b; }
  public void hide( ) { ... }
  public void show( ) { ... }
}
```


TAD GRANDE INTEIRO

CLASSE BigInt

- **Implementar um novo tipo de inteiros, chamado BigInt, capaz de operar com valores de até 50 algarismos.**

```
public classe BigInt {
   private static final int Ndigits = 50;
   private byte [] valor = new byte[Ndigits];

   public BigInt(long d){ ... }
   public BigInt(String d){ ... }
   public BigInt soma (BigInt x){   ... }
   public BigInt sub (BigInt x){  ... }
   public void print(){ ...}
   public void println(){ ...}
   ... Outras Operacoes
}
```


USO DE BigInt

```
public class Usuaria{
   BigInt a = new BigInt(1000000);
   BigInt b = new BigInt("5.000.000.000.000.000");
   BigInt c, d;
   d = b.soma(a);  // d = b + a
   c = a.soma(d);  // c = a + d
   c.println();
}

Saída: 5000000002000000
```


CONSTRUTORAS DE BigInt

```
public BigInt(long d){
    for (int i=Ndigits-1; i>=0; i--) {
      valor[i] = (byte)(d % 10);  d = d / 10;
    }
}
public BigInt(String d){
   char a; int k = Ndigits-1;
   for (int i=d.length()-1; i>=0; i--) {
     a = d.charAt(i);
     if (Character.isDigit(a))
             valor[k--] = (byte)(Character.digit(a,10));
   }
   for (int i = k; i>=0; i--) valor[i] = 0;
}
```


OPERAÇÕES EM CLASSES I ...

```
 public BigInt soma (BigInt x){
    byte vaium = 0, d;
    BigInt total = new BigInt(0);
    for (int i=Ndigits-1; i>=0; i--) {
      d = (byte)(valor[i]+x.valor[i]+vaium);
      if (d > 9) {
         total.valor[i] = (byte)(d-10); vaium = 1;
      }
      else { total.valor[i] = (byte)d; vaium = 0; }
    }
    return total;
 }
```

OPERAÇÕES EM CLASSES I ...

```
public void print(){
    char a; byte b; int k = 0;
    while ((k < Ndigits - 1) && (valor[k] == 0)) k++;
    for (int i = k ; i < Ndigits; i++) {
                System.out.print(valor[i]);
    }
}

public void println(){
   print();
   System.out.println();
}
```


OPERAÇÕES EM CLASSES II

```
public classe BigInt {
  Estruturas de Dados ...
  public BigInt(long d){ ... }
  public BigInt(String d){ ... }
  public void soma (BigInt x){  ... }
  public void sub (BigInt x){  ... }
  public void print() {...}
  public void println() {...}
  ...
  Outras Operações
}
```


USO DE `BigInt`

```
classe Usuaria{
   BigInt a = new BigInt(1000000);
   BigInt b = new BigInt("5.000.000.000.000.000");
   b.soma(a);  // b = b + a
   a.soma(b);  // a = a + b
   a.println();
}

Saída: 5000000002000000
```


OPERAÇÕES EM CLASSE II ...

```
  public void soma (BigInt x){
     byte vaium = 0, d;
     for (int i=Ndigits-1; i>=0; i--) {
        d =(byte)(valor[i]+x.valor[i]+vaium);
        if d > 9 {
           valor[i] = d - 10; vaium = 1;
        }
        else { valor[i] = d; vaium = 0; }
     }
 }
 public void sub (BigInt x){  ... }
 ...
}
```

OPERAÇÕES EM CLASSES III

```
public classe BigInt {
 private static final int Ndigits = 50;
 byte [] valor = new byte[Ndigits];
 public BigInt(long d){ ... }
 public BigInt(String d){ ... }
 public void soma (BigInt x){ ... }
 public void sub (BigInt x){  ... }
 ...
 public BigInt Copia() {...}
 Outras Operações
}
```


OPERAÇÕES EM CLASSES III

```
classe Usuaria{
   BigInt a = new BigInt(1000000);
   BigInt b = new BigInt("5.000.000.000.000.000");
   BigInt c, e;
   e = b.copia();
   e.soma(a);   // e = b + a
   c = a.copia();
   c.soma(e);  // c = a + (b + a)
   c.println();
}

Saída: 5000000002000000
```


FIM

AMBIENTES DE PROGRAMAÇÃO

HIERARQUIA

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

AMBIENTES DE PROGRAMAÇÃO

TRATAMENTO DE EXCEÇÕES

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

19 de março de 2015

CONCEITO DE EXCEÇÕES

DETECÇÃO E TRATAMENTO DE ERROS

PROBLEMA:

- **Métodos de bibliotecas (métodos-servidor) têm meios para detectar situações de erro de execução, MAS geralmente não sabem o que fazer nestas situações.**
- **Métodos-cliente sabem o que fazer nestes casos, MAS não têm meios para detectar a situação de erro ocorrida no método-servidor.**

DETECÇÃO E TRATAMENTO DE ERROS ...

SOLUÇÃO:

- **Tratamento de Exceptions**
- **Com exceções, métodos-servidor têm mais de uma opção de retorno:**
  - **retorno normal, após o ponto de chamada**
  - **retorno (via exceção) em outros endereços**
- **Métodos-cliente podem então perceber se houve ou não falha na execução do método-servidor e processar cada tipo de retorno de forma apropriada.**

EXCEÇÕES EM JAVA

LEVANTAMENTO E CAPTURA DE FALHA

```
class E extends Exception {
   public int a; ... public E(int x){...}
}
class A {
   void g(...) throws E {... throw new E(k); ... }
   ...
}
class B {  ...
   void f(...) {
      A a;  ...
      try { ... a.g(...); ...}
      catch (E e){...  e.a ...};
   }
}
```

FORMATO GERAL DAS EXCEÇÕES

```
try {... bloco de comandos contendo throw ...}
catch (tipo_excecao1 identificador) {
    bloco de comandos
}
catch (tipo_excecao2 identificador) {
    bloco de comandos
}
...
catch (tipo_excecaoN identificador) {
    bloco de comandos
}
finally {bloco de comandos sempre executado}
```


CLÁUSULA `finally`

- `finally` **somente não é executado quando o bloco** `try` **executa** `System.exit(0)`**, o qual finaliza imediatamente a execução**
- **No demais casos,** `finally` **é sempre executado, até mesmo quando o bloco do** `try` **ou** `catch` **executam** `return`
- **Valor de retorno calculado antes de executar o** `finally`
- **Parâmetro de** `catch` **deve ser da família** `Throwable`

CLÁUSULA `finally` e `System.exit`

```
public class Finally {
   static public void main(String[ ] args){
     Final a = new Final( ); a.f(); System.out.print("Acabou");
   }
}
class Final {
   public void f( ) {
     int x = 10; System.out.println("x = " + x);
     try { x = 11; System.exit(0); }
     catch(Exception e){x = 12; System.out.println("x = " + x);}
     finally {x = 11; System.out.print("x = " + x);}
   }
} // Saida: x = 10
```


CLÁUSULA `finally` e `return`

```
public class Finally {
   static public void main(String[ ] args) {
     Final a = new Final( ); a.f(); System.out.print("Acabou");
   }
}
class Final {
   public void f( ) {
      int x = 10; System.out.print("x = " + x);
      try {x = 11; System.out.print("x = " + x); return; }
      catch(Exception e){x = 12; System.out.print("x = " + x);}
      finally {x = 13; System.out.print("x = " + x);}
      System.out.print("Aqui não passa");
   }
} //Saida: x = 10  x = 11  x = 13  Acabou
```

CLÁUSULA finally e catch com throw

```
public class Finally {
   static public void main(String[ ] args) {
     Final a = new Final( ); a.f(); System.out.print("Acabou");
  }
}
class Final {
   public void f( ) {
     int x = 10; System.out.print("x = " + x);
     try{x=11;System.out.print("x = "+x);throw new Exception();}
     catch(Exception e){x =12;System.out.print("x = " + x); }
     finally {x = 13; System.out.print("x = " + x); }
   }
}
//Saida: x = 10 x = 11 x = 12 x = 13 Acabou
```


CLÁUSULA finally com return

```
public class Finally {
   static public void main(String[ ] args) {
     Final a = new Final(); int x = a.f();
     System.out.print("x = " + x);
  }
}
class Final {
   public int f( ) {
     int x = 10;
     try {return x; } catch(Exception e) {  x =100; }
     finally {x = 1000; return 2*x;}
   }
}
//Saida: x = 2000
```


CLÁUSULA finally e catch com return

```
public class Finally {
   static public void main(String[ ] args){
     Final a = new Final( );int x = a.f();
     System.out.println("x = " + x);
  }
}
class Final {
   public int f( ) {
     int x = 0;;
     try { x = 1; throw new Exception( ); }
     catch(Exception e) { x = x + 10; return x; }
     finally {x = x + 100; return 2*x;}
   }
} //Saida: x = 222
```


CLÁUSULA finally e catch com return

```
public class Finally {
   static public void main(String[ ] args) {
     Final a = new Final( );
     int x = a.f(); System.out.println("x = " + x);
   }
}
class Final {
   public int f( ) {
     int x = 0;;
     try { x = 1; throw new Exception( ); }
     catch(Exception e) { x = 100; return x+7; }
     finally {x = x + 1000; }
   }
} //Saida: x = 107
```

CLÁUSULA `finally` e Valor de retorno

```
public class Finally {
   static public void main(String[ ] args){
     Final a = new Final( ); int x = a.f();
     System.out.println("x = " + x + ", a.y = " + a.y);
  }
}
class Final {
   public int y = 0;
   public int f( ) {
     int x = 0; try { x = 1; throw new Exception( ); }
                catch(Exception e) { x = 100; return x+7; }
                finally {x = x + 1000; y = x; }
   }
} //Saida: x = 107, a.y = 1100
```


EXEMPLOS DE EXCEÇÕES

EXEMPLO DE EXCEÇÕES I

```
public class Range extends Exception { };
class Vector {
   private int [ ] p; private int sz;
   public void Vector(int n){
      p = new int[n]; sz = n;
   }
   ...
   public int get(int i) throws Range {
      if ( 0 <= i && i < sz ) return p[i] ;
      throw new Range();
   }
}
```


EXEMPLO DE EXCEÇÕES I ...

```
class User {
   .......
   void f( ){
      Vector  x; ...
      try { ...; g(x); ... }
      catch(Range x){ ... tratador da excecao ... };
      ...
   }
   void g(Vector v) throws Range {
       ... k = v.get(j); ...
   }
   public static void main(String[] args){... f(); ...}
}
```

EXEMPLO DE EXCEÇÕES II

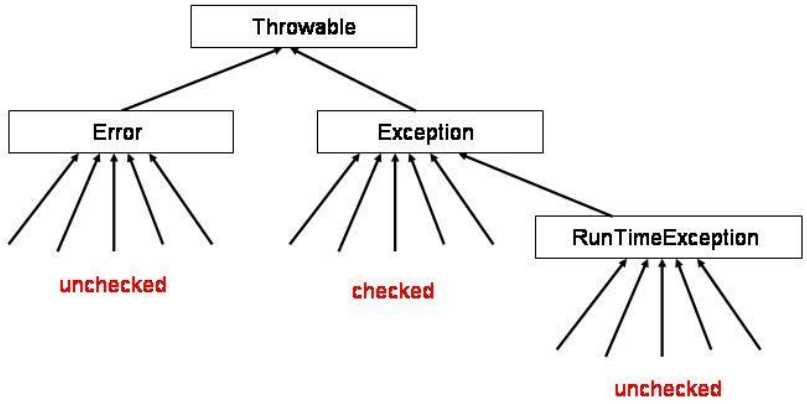
```
public class Range extends Exception { };
public class Size  extends Exception { };
class Vector {
   private int [ ] p; private int max;
   public Vector(int n) throws Size {
       if (n  < 0 ) { throw new Size(); }
       p = new int[n]; max = n;
   }
   public int get(int i) throws Range {
      if ( 0 <= i && i < max ) return p[i] ;
      throw new Range();
   }
   ...
}
```


EXEMPLO DE EXCEÇÕES II ...

```
class User {
   int n,k;
   void f( ) throws Range, Size {
      Vector x = new Vector(k); ...
      try { ... z= new Vector(n);...; g(z); ...   }
      catch(Range e) { ... tratador da excecao ...}
      catch(Size e)  { ... tratador da excecao ...}
      g(x); ...
   }
   void g(Vector v) throws Range {... ; k = v.get(j); ...}
   public static void main(String[] args) throws Range {
      try { ... f( ); ...} catch(Size e) { ... }
   }
}
```


DECLARAÇÕES DE EXCEÇÃO

HIERARQUIA DE EXCEÇÕES



DECLARAÇÃO DE EXCEÇÕES

- **Exceção em Java é um objeto.**
- **O tipo de uma exceção é sempre uma classe.**
- **A classe definida para ser exceção deve estender** `Exception` **ou** `Throwable`**.**
- **A exceções do tipo** `Error` **e** `RunTimeException` **são chamadas** unchecked exceptions
- **Toda extensão de** `Error` **ou de** `RunTimeException` **declarada pelo usuário são** unchecked exception
- **Toda extensão de** `Exception` **declarada pelo usuário são** checked exceptions

DECLARAÇÃO DE `checked exceptions`

- **Todo método é obrigado a anunciar em seu cabeçalho as exceções** checked **que ele levanta e que não são por ele tratadas.**
- **Se um método declara que pode lançar uma excecão** `A`**, ele pode também lançar qualquer descendente de** `A`**.**
- **Um método sem lista de** `throws` **não pode lançar exceções** checked **não tratadas por ele.**

MÉTODOS QUE FALHAM

- **Métodos que falham devem anunciar em seus cabeçalhos que tipo de exceções** checked **eles podem levantar e que não são por eles tratadas.**
- **Cabeçalho de métodos que podem falhar:**

```
<modificador> T f(lista de parametros)
   throws <lista de excecoes>
```

- **Exceções do tipo** unchecked **não precisam ser anunciadas nos cabeçalhos de métodos.**

MÉTODOS QUE FALHAM ...

- **Um método indica que falhou levantando uma exceção via o comando:**

```
   throw new X(...);
```

  onde `X` **é o nome da classe que identifica a exceção**
- **A exceção lançada é o objeto criado no** `throw`
- **O objeto lançado é como um argumento passado remotamente à função** `catch`**, que tratará a exceção.**

O LANÇAMENTO DE EXCEÇÕES

- Comando `try` delimita região do programa onde falhas são detectadas.
- Comando `throw new X(...)` cria um objeto do tipo X e o lança.
- O tipo do objeto lançado identifica o `catch` que deve tratar as excecões.
- O objeto lançado pode ser capturado no mesmo método ou em qualquer um dos métodos localizados no `caminho inverso das chamadas`.

LANÇAMENTO DE EXCEÇÕES ...

- Iniciadores campos não devem fazer chamadas a métodos que falham, porque não têm como capturar exceções.
- Blocos de iniciação de variáveis estáticas não podem lançar exceções, mas podem capturá-las.
- Construtores tanto podem lançar como capturar exceções.

CAPTURA DE EXCEÇÕES

- Lançamento de exceções ocorridos no escopo do `try` são capturados pelos `catches` correspondentes.
- `catch` correspondente é aquele cujo parâmetro tem o tipo do objeto lançado e que está localizado no caminho das chamadas ativas.
- Comando `throw` não-tratado por um `catch` do método causa procura no `caminho inverso das chamadas`, do primeiro `catch` que trata a exceção.
- Neste caso, a pilha de execução é podada no nível do `catch` que faz a captura.

CAPTURA DE EXCEÇÕES ...

- Se um método lança uma exceção que nunca é capturada, o programa é cancelado com uma mensagem padrão.
- Após execução do `catch`, o processamento continua na claúsula `finally` e depois após o `try`.

TRANSMISSÃO DE INFORMAÇÃO SOBRE A FALHA

CLASSES DE EXCEÇÃO

```
class Máximo extends Exception {
    public byte id;
    public Máximo(byte id){ this.id = id; }
}

class Mínimo extends Exception {
    public byte id;
    public Mínimo(byte id){ this.id = id; }
}

class Tamanho extends Exception {
    public byte id;
    public Tamanho(byte id){ this.id = id; }
}
```


TIPO PILHA DE INTEIROS

```
class IntStack {
   private static final int Bottom = -1;
   private int last; private int[] item;
   private int top = Bottom;
   public Stack(int n) throws Tamanho {
      if (n < 0) throw new Tamanho(1);
      this.last = n -1 ;
      item = new int[n];
   }
   public boolean empty() {... }
   public void push(int v) throws Máximo { ... }
   public int pop() throws Mínimo { ... }
}
```


TIPO PILHA DE INTEIROS ...

```
public boolean empty() {
   return (top == Bottom);
}

public void push(int v) throws Máximo {
   if (top == last) throw new Máximo(1);
   item[++top]= v;
}

public int pop() throws Mínimo {
   if(empty()) throw new Mínimo(1);
   return item[top--];
}
```

TIPO FILA DE INTEIROS

```
class IntQueue {
   private int front; private int back;
   private int tamanho;  private int[] item;
   public Queue(int n) throws Tamanho {
      if (n < 0) throw new Tamanho(2);
      item = new int[n]; tamanho = n;
   }
   public boolean empty() { ... }
   public void insert(int v) throws Máximo { ... }
   public int remove() throws Mínimo { ... }
}
```


TIPO FILA DE INTEIROS ...

```
public boolean empty() {
   return (front == back);
}
public void insert(int v) throws Máximo {
   int nextpos = (++back) % tamanho;
   if (nextpos == front) throw new Máximo(2);
   back = nextpos;
   item[back]= v;
}
public int remove() throws Mínimo {
   if(empty()) throw new Mínimo(2);
   front = (++front % tamanho);
   return item[front];
}
```


USO DOS TIPOS `IntStack` e `IntQueue`

```
class User {
  public static void main(String args) throws Tamanho {
    IntStack s = new IntStack(5); IntQueue q = new IntQueue(5);
    for (int i=0; i<= 10; i++) {
      try {... s.push(i);...; k = s.pop();...; q.insert(i);...;}
      catch(Máximo e) {
        if (e.id == 1)
             System.out.println("Foi na Pilha");
        else System.out.println("Foi na Fila");
      }
      catch (Mínimo e) { ... }
    }
  }
}
```


AGRUPAMENTO DE EXCEÇÕES

AGRUPAMENTO DE EXCEÇÕES

- **Pode-se criar uma hierarquia de classes de exceção para melhor estruturar seu tratamento.**

```
class MathError extends Exception { }
class Overflow extends MathError { }
class Underflow extends MathError { }
class ZeroDivide extends MathError { }
class E extends Exception { }
```


AGRUPAMENTO DE EXCEÇÕES ...

```
class User {
  ...
  try {
     ... throw new Overflow(); ...
     ... throw new ZeroDivide; ...
     ... throw new E(); ...
  }
  catch(Overflow e)  {trata Overflow e descendentes }
  catch(Underflow e) {trata Underflow e descendentes}
  catch(MathError e) {trata MathError e descendentes}
  catch(Exception e) {trata todas que chegarem aqui}
  finally { sempre passa por aqui }
  ...
}
```


ORDEM DE `catches`

- **A ordem dos `catch` não é livre.**
- **O compilador não aceita a ordenação de `catches` usada no método `noGood` abaixo.**

```
class A extends Exception { }
class B extends A { }
class BadCatch {
  public void noGood () {
    try {... throw new B(); ...}
    catch (Exception e ) { ... }
    catch (A s) { ... }
    catch (B s) {....}
  }
}
```


ORDEM DE `catches` ...

- **A ordem abaixo está bem.**

```
class A extends Exception { }
class B extends A { }
class GoodCatch {
  public void goodTry () {
    try { ... throw new B(); ...}
    catch (B s) {....}
    catch (A s) { ... }
    catch (Exception e) { ...}
  }
}
```

INFORMAÇÕES AGREGADAS A EXCEÇÕES

DESCRIÇÃO DA EXCEÇÃO

- **Toda classe que estende** `Throwable` **ou** `Exception` **tem um string que descreve a condição de exceção.**

```
public class E1 {
  public static void main(String[] args) {
    try { throw new ArrayStoreException();}
    catch (ArrayStoreException e) {
        System.out.println(e);    }
  }
}
Saida: java.lang.ArrayStoreException
```


DESCRIÇÃO DA EXCEÇÃO...

```
class Ex extends ArrayStoreException { }
public class E2 {
  public static void main(String[] args) {
    String s;
    try { throw new Ex();}
    catch (ArrayStoreException e) {
        s = e.toString();
        System.out.println(e + " " + s);
    }
  }
}
Saida: Ex Ex
```


DESCRIÇÃO DA EXCEÇÃO ...

- **Informações podem ser acrescentadas à descrição da exceção via chamada à construtora da superclass**

```
class Ex extends ArrayStoreException {
    Ex(){ super("New String"); }
}
public class E3 {
  public static void main(String[] args) {
    try { throw new Ex();}
    catch (ArrayStoreException e) {
        System.out.println(e);
    }
  }
}
Saida: Ex: New String
```

DESCRIÇÃO DA EXCEÇÃO ...

```
class Ex extends ArrayStoreException {
    Ex(String s){ super(s); }
}
public class E4 {
  public static void main(String[] args) {
    try { throw new Ex("New String");}
    catch (Ex e) {System.out.println(e);}
    catch (ArrayStoreException e) {System.out.println(e);}
  }
}
Saida: Ex: New String
```


FIM

AMBIENTES DE PROGRAMAÇÃO

AGRUPAMENTO DE CLASSES

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

AMBIENTES DE PROGRAMAÇÃO

GENERICIDADE

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

19 de março de 2015

SUMÁRIO

- Classes Parametrizadas
- Tad Importantes
  - Interface List
  - Classe ArrayList
  - Classe LinkedList
  - Classe Vector

CLASSES PARAMETRIZADAS

Classe Genérica

```
class A<T>{
    T x ;
    public A() { x = (T) new Object(); }
    public void s(T y){x = y;}
    public T g( ) { return x;}
}
public class G {
    public static void main(String[ ] a) {
        A<Integer> m = new A<Integer>();
        int k,i = 100;
        m.s(i);  k = m.g();
        System.out.print("X = "+ k);
    }
} //Saída: X = 100
```


RESTRIÇÃO NA CRIAÇÃO DE OBJETO GENÉRICO

```
class A<T> {
    T x ;
    public A() { x = new T( ); }
    public void s(T y){x = y;}
    public T g( ) { return x;}
}
```

Mensagem do compilador java:

```
unexpected type; Required class
public A() { x = new T( ); }
                    ^
```


ALOCAÇÃO DE OBJETOS T

```
class A<T> {
    T x ;
    public A() { x = (T) new Object(); }
    public void s(T y){x = y;}
    public T g( ) { return x;}
}
public class G {
    public static void main(String[ ] a) {
        A<Integer> m = new A<Integer>(); int k,i = 10;
        A<Double>  q = new A<Double>(); double w, t = 100.0;
        m.s(i); k = m.g(); q.s(t); w = q.g();
        System.out.print("X = "+ k + ", W = "+ w);
    }
}  // Saída: X = 10, W = 100.0
```


HERANÇA COM GENÉRICOS INSTANCIADO

```
class A<T> {
  T x ;
  public void s(T y){x = y;}
  public T g( ) { return x;}
}
class B extends A<Double>{ }
public class G {
   public static void main(String[ ] a) {
      A<Integer> m = new A<Integer>(); int k,i = 10;
      m.s(i); k = m.g();
      B q = new B();  double w, t = 100.0; q.s(t); w = q.g();
      System.out.print("X = "+ k + ", W = "+ w);
   }
} Saída: X = 10, W = 100.0
```

USO ERRADO DE GENÉRICOS

```
class A<T> {
  T x ;
  public void s(T y){x = y;}
  public T g( ) { return x;}
}
class B extends A<Double>{ }
public class G {
   public static void main(String[ ] a) {
      A<Integer> m = new A<Integer>(); int k,i = 10;
      m.s(i); k = m.g();
      B q = new B();  int w, t = 100; q.s(t); w = q.g();
      System.out.print("X = "+ k + ", W = "+ w);
   }
} mensagem: incompatible type em q.s(t) e w.q.g()
```


HERANÇA DE GENÉRICOS (ERRADO)

```
class A<T> {
  T x ;
  public A() {x = (T) new Object();}
  public void s(T y){x = y;}
  public T g( ) {return x;}
}

class B extends A<T>{ }
```

Mensagem do compilador: cannot find symbol T

HERANÇA DE GENÉRICOS COM PARÂMETROS DE TIPO

```
class A<T> {
  T x ;
  public A() { x = (T) new Object(); }
  public void s(T y){x = y;}
  public T g( ) { return x;}
}
class B<T> extends A<T>{ }
public class G {
  public static void main(String[ ] a) {
    B<Integer> q = new B<Integer>(); int w,t = 10;
    q.s(t); w = q.g();
    System.out.print("W = "+ w);
  }
}  // Saída: W = 10
```


HERANÇA COM MAIS DE UM PARÂMETRO DE TIPO

```
class A<T> {
  T x ;
  public A() { this.x = (T) new Object(); }
  public void s(T y){x = y;}
  public T g( ) { return x;}
}

class B<T,R> extends A<T>{
     public R h;
     public B(R v) { super(); h = v;}
}
```

HERANÇA COM MAIS DE UM PARÂMETRO DE TIPO ...

```
public class G {
  public static void main(String[ ] args) {
    A<Integer> m = new A<Integer>();
    int k,i = 10; m.s(i); k = m.g();
    B<Integer,Double> q = new B<Integer,Double>(1000.0);
    int a,b = 100;
    double w;
    q.s(b);
    a = q.g();
    w = q.h;
    System.out.print("k = "+ k + ", a = "+ a + ", w = " + w);
  }
} // Saída: k = 10, a = 100, w = 1000.0
```


HERANÇA DE PARÂMETRO DEFAULT

```
class A<T> {
  T x ;
  public A() { x = (T) new Object(); }
  public void s(T y){x = y;}
  public T g( ) { return x;}
}
class B extends A{ } // mesmo que class B extends A<Object>{ }
public class G {
  public static void main(String[ ] a) {
    B q = new B(); int w,t = 10;
    q.s(t); w = (Integer)q.g(); // g: void -> Object
    System.out.print("W = "+ w);
  }
}  // Saída: W = 10
```


INTERFACES GENÉRICAS

```
public interface I<T> {
  void s(T y);
  T g ( );
}
class A<T> implements I<T>{
  T x ;
  public A( ) { }
  public void s(T y){
     x = y;
  }
  public T g( ) {
     return x;
  }
}
```

```
class B extends A<Integer>{ }
public class G {
  public static
       void main(String[ ] a){
    B q = new B();
    int w,t = 10;
    q.s(t);
    w = (Integer)q.g();
    System.out.print("W = "+ w);
  }
}
// Saida: W = 10
```


INTERFACES GENÉRICAS ...

```
public interface I<T> {
  void s(T y);
  T g ( );
}
class A implements I<Integer>{
  Integer x ;
  public A( ) { }
  public void s(Integer y){
    x = y;
  }
  public Integer g( ) {
    return x;
  }
}
```

```
class B extends A{ }
public class G {
  public static
       void main(String[ ] a){
    B q = new B();
    int w,t = 10;
    q.s(t);
    w = (Integer)q.g();
    System.out.print("W = "+ w);
  }
}
// Saida: W = 10
```

PARÂMETRO DE TIPO COM LIMITE SUPERIOR

```
class A<T extends R> {
    int x;
    public void s(T y){ x = y.f();}
    public int  g( ) { return x;}
}
class R {
    public int f( ){return 1;}
}
class R1 extends R{
    public int g( ){return 100;}
}
class R2 extends R{
    public int f( ){return 1000;}
}
```


PARÂMETRO DE TIPO COM LIMITE SUPERIOR ...

```
public class G {
  public static void main(String[ ] args) {
      A<R1> m = new A<R1>( );
      A<R2> r = new A<R2>( );
      R1 a = new R1();
      R2 b = new R2();
      int j,k;
      m.s(a); r.s(b);
      k = m.g(); j = r.g();
      System.out.print("k = "+ k + ", j = "+ j);
  }
} // Saída: k = 100, j = 1000
```


PARÂMETRO DE TIPO COM LIMITE SUPERIOR(ERRADO)

```
class A<T extends R> {
  int x;
  public A( ) {  }
  public void s(T y){ x = y.f();}
  public int g( ) { return x;}
}
class R {public int f( ){return 1;}}
public class G{
  public static void main(String[ ] args) {
      A<Integer> m = new A<Integer>( );
      int j,k = 1;  m.s(k); j = r.g();
      System.out.print("k = "+ k + ", j = "+ j);
  }
} // ERRO DE COMPILAÇÃO
```


CURINGAS EM PARÂMETROS DE MÉTODOS

```
class R {
    public int f( ){return 1;}
}
class R1 extends R{
    public int g( ){
       return 100;
   }
}
class R2 extends R{
    public int f( ){
       return 1000;
   }
}
```

```
class C {
   void h(A<R> a) { a.f() ; }
}
...
A<R1> a = new A<R1>;
C c = new C(); D d = new D();
c.h(a); // A<R1> não é subtipo
d.h(a); // OK
class D {
   void h(A<? extends R> a) {
        a.f() ;
    }
}
```

INTERFACE DE ITERADORES

```
interface Iterator{
    boolean hasnext( );

    Object next( );
}
```


LISTAS HETEROGÊNEAS

```
List stringList  = new LinkedList();
List integerList = new LinkedList();
integerList.add(new Integer(1));
integerList.add(new Integer(2));
stringList.add(new String("Eu sou um String"));
stringList.add(new Integer(1));
Iterator listIterator = integerList.iterator();
while(listIterator.hasNext()) {
   String item = (String)listIterator.next(); //OK?
}
listIterator = stringList.iterator();
while (listIterator.hasNext()) {
  String item = (String)listIterator.next(); //OK?
}
```


LISTAS HOMOGÊNEAS

```
import java.util.LinkedList;
import java.util.Collections;
import java.util.Iterator;
public class GenericsExample {
  static public void main(String[] args) {

    LinkedList<String>  stringList = new LinkedList<String>();
    LinkedList<Integer> integerList= new LinkedList<Integer>();

    "Uso das Listas de Integers e de Strings"

  }
}
```


USO DAS LISTAS DE INTEIROS E DE STRINGS

```
integerList.add(new Integer(1));
integerList.add(new Integer(2));
stringList.add(new String("Eu sou um String"));
stringList.add(new Integer(1)); // compilation error

Iterator<Integer> listIterator = integerList.iterator();
String item;
while(listIterator.hasNext()) {
    item = listIterator.next(); // compilation error
}
listIterator = stringList.iterator(); // compilation error
while (listIterator.hasNext()) {
    item = listIterator.next();
}
```

TAD IMPORTANTES IMPLEMENTADOS EM JAVA

- `interface List`:
- `class ArrayList implements List`:
  - **provê acesso eficiente a seus elementos**
  - **eficiência na inserção e remoção no fim da lista**
  - **inserção ou remoção no meio causa movimentação de elementos**
- `class LinkedList implements List`:
  - **provê acesso menos eficiente a elementos aleatórios**
  - **eficiência na inserção e remoção em qualquer posição da lista**
  - **inserção ou remoção no meio causa movimentação de elementos**
- `class Vector implements List`:
  - **provê um arranjo de tamanho flexível**

CLASSE VECTOR

ALGUMAS OPERAÇÕES DA CLASSE `Vector`

- `public Vector(int initialCapacity, int capacityIncrement)`
- `public Vector(int initialCapacity)`
- `public Vector()`
- `public final void addElement(Object elem)`
- `public final void insertElementAt(Object elem, int index)`
- `public final void setElementAt(Object elem, int index)`
- `public final void removeElementAt(int index)`
- `public final void ElementAt(int index)`
- `public final Enumeration elements( )`
- `public int capacity( )`
- `public int capacityIncrement( )`

Interface `Enumeration`

```
interface Enumeration{
     boolean hasMoreElements( );
     object nextElement( ) throws NoSucElementException;
 }
```

FIM

AMBIENTES DE PROGRAMAÇÃO
INTERFACES GRÁFICAS DO USUÁRIO

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

23 de março de 2015

SUMÁRIO

- Introdução
- Classe Graphics
- Classes Polygon, Font e Color
- Componentes Básicos
- Gerência de Layout
- Modelo de Eventos
- Botões
  - Botões de Comando
  - Seleção de Opções
  - Seleção Exclusiva
  - Seleção em Listas
- Painéis
- Tratamento de Eventos
  - Eventos de Mouse
  - Eventos de Teclado
- Componentes Avançados
  - JTextArea
  - JSlider
  - Menu de Opções
  - Janelas Múltiplas
- Aparência e Comportamento

INTRODUÇÃO

GLOSSÁRIO

GUI - *Grafical User Interface*: **é a interface gráfica de um programa com a qual o usuário pode interagir.**

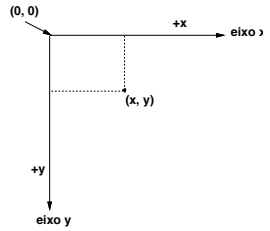
Componente : **um objeto gráfico com o qual o usuário interage via mouse ou teclado.**

AWT - *Abstract Windowing Toolkit*: **um pacote de classes Java que disponibiliza ao programador uma série de classes relacionadas com as capacidades de *GUI* da plataforma local.**

Swing : **um pacote gráfico Java, mais moderno que o AWT, cujos componentes são escritos, manipulados e exibidos completamente em Java. Em geral o comportamento de componentes AWT depende da plataforma, ao passo que o comportamento dos componentes Swing não depende dela.**

SISTEMAS DE COORDENADAS

- **As unidades são medidas em *Pixels*.**
- ***Pixel*** - **A menor resolução do monitor.**
- **(0, 0)** - **Canto superior esquerdo de uma janela.**



CLASSES IMPORTANTES PARA DESENHOS

- **A classe JFrame permite a construção de janelas.**
- **A classe Graphics é usada para desenhar formas sobre componentes gráficos.**
- **A classe `Polygon` é usada para criar polígonos**
- **A classe Font controla a aparência das fontes disponíveis.**
- **A classe Color define a cor dos elementos gráficos.**

JANELAS, QUADROS OU FRAMES

- **Uma *Frame* é uma janela que possui decorações como bordas, barra de título e botões para fechar e iconificar.**
- **Em Java, frames podem ser objetos da classe JFrame.**
- **Aplicativos que possuem uma interface gráfica usam frames para definir janelas.**
- **Classes gráficas baseadas no pacote javax.swing geralmente herdam as propriedades da classe JFrame.**
- **O estilo de uma janela, que é um objeto do tipo JFrame, depende da plataforma (Ex.: Windows, UNIX, etc).**
- **Pode-se desenhar sobre um objeto JFrame e adicionar-lhe componentes gráficos, como botões, rótulos, caixas de texto.**

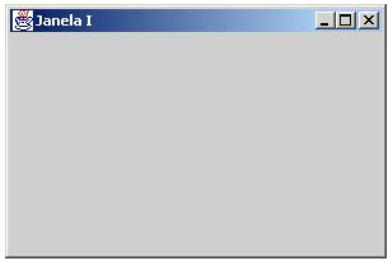
HIERARQUIA DE JFRAME

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
           |      |
           |      +--java.awt.Window
           |               |
           |               +--java.awt.Frame
           |                      |
           |                      +--javax.swing.JFrame
           +--javax.swing.JComponent
                |
                +--- javax.swing.JPanel
```


MÉTODOS DA CLASSE JFRAME I

- `setTitle(String t)`:
  define título da janela
- `setSize(int largura, int altura)` **ou** `setSize(Dimension d)`:
  define dimensões da janela
- `show( )`:
  coloca janela visível e por cima das demais
- `setVisible(boolean b)`:
  controla visibilidade da janela
- `setLocation(int x, int y)` **ou** `setLocation(Point p)`:
  posiciona a janela na tela

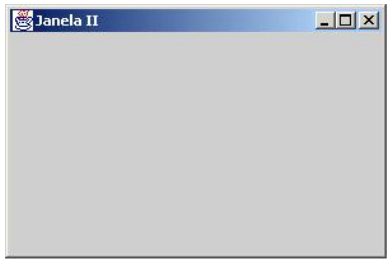
CRIAÇÃO DE PRIMEIRA JANELA



CRIAÇÃO DA PRIMEIRA JANELA ...

```
import javax.swing.*;

class Janela extends JFrame {
   public Janela ( ) {
      setTitle("Janela I");
      setSize(300,200);
  }
}
public class UsaPrimeiraJanela {
   public static void main(String [ ] args) {
      JFrame janela = new Janela( );
      janela.setVisible(true);
   }
}
```

CRIAÇÃO DE JANELA QUE FECHA



CRIAÇÃO DE JANELA QUE FECHA ...

```
import javax.swing.*; import java.awt.event.*;
class Janela extends JFrame {
   public Janela (){setTitle("Janela II"); setSize(300,200);}
}
class Ouvinte extends WindowAdapter{
   public void windowClosing (WindowEvent e){System.exit(0);}
}
public class UsaSegundaJanela {
   public static void main(String [ ] args) {
      JFrame janela = new Janela( );
      Ouvinte a = new Ouvinte( ); janela.addWindowListener(a);
      janela.setVisible(true);
   }
}
```

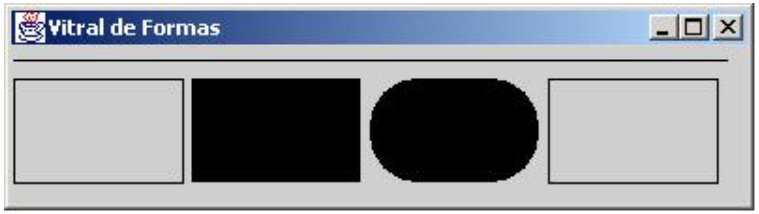

OUTRA FORMA DE ADICIONAR O OUVINTE

```
import javax.swing.*; import java.awt.event.*;
class Janela extends JFrame {
   public Janela (){setTitle("Janela II"); setSize(300,200);}
}
public class UsaSegundaJanela {
   public static void main(String [ ] args) {
      JFrame janela = new Janela( ); janela.setVisible(true);
      janela.addWindowListener(
         new WindowAdapter{
             public void windowClosing (WindowEvent e)
                 {System.exit(0);}
         }
      );
} }
```


MÉTODOS DA CLASSE JFRAME II

- `paint(Graphics g)`:
  - **Desenha informações gráficas na janela.**
  - **É chamado automaticamente, por exemplo, em resposta a um** *evento*, **como** *click* **de mouse ou abertura de janela.**
  - **Deve ser redefinido pelo programa do usuário**
  - **Não deve ser chamado diretamente**
  - `paint(Graphics g)` **é da classe** `Container`
- `repaint()`:
  - **Limpa o fundo do componente gráfico corrente de qualquer desenho anterior e chama o método** `paint` **para redesenhar as informações gráficas sobre tal componente.**
  - **Método** `repaint()` **pode ser chamado, quando necessário.**

CRIAÇÃO DE OUTRA JANELA



CRIAÇÃO DE OUTRA JANELA ...

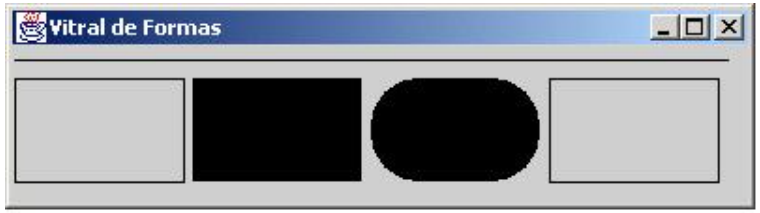
```
import java.awt.*; import javax.swing.*;
public class Vitral extends JFrame {
    public Vitral( ) {
        super ("Vitral de Formas"); setSize (400, 110);
        show();  // setVisible(true);
    }
    public void paint (Graphics g) {
        g.drawLine (5, 30, 380, 30);
        g.drawRect (5, 40, 90, 55);
        g.fillRect (100, 40, 90, 55);
        g.fillRoundRect (195, 40, 90, 55, 50, 50);
        g.draw3DRect (290, 40, 90, 55, true);
    }
}
```


CRIAÇÃO DE OUTRA JANELA ...

```
import java.awt.*; import javax.swing.*;
import java.awt.event.*;
public class UsaVitral {
    public static void main (String args[]) {
        Vitral j = new Vitral( );
        j.addWindowListener(
            new WindowAdapter () {
                public void windowClosing (WindowEvent e) {
                    System.exit(0);
                }
            }
        );
    }
}
```


SAÍDA DE `Vitral`

```
>java UsaVitral
```



SEQÜÊNCIA DE AÇÕES DE `UsaVitral`

- **Construtora de `JFrame` cria a estrutura que representa a janela**
- **Coloca-se título na janela Vitral**
- **Define-se o tamanho da janela**
- **Torna a janela visível, causando chamada ao `paint()`**
- **Associa-se objeto ouvinte ao objeto Vitral**

MÉTODOS DA CLASSE JFRAME III

- `dispose ( )`:
  fecha janela e recupera os recursos do sistema utilizados
- `setIconImage(Image imagem)`:
  define a imagem da janela iconizada ou minimizada
- `setBounds(intx, int y, int largura, int altura)`:
  posiciona janela na tela e define suas dimensões
- `Point getLocation( )`:
  devolve coordenadas do vértice esquerdo superior da janela

CLASSE `GRAPHICS`

CLASSE `Graphics`

- **A classe `Graphics` é abstrata.**
- **Cada ambiente implementa uma classe derivada de `Graphics` com todos os recursos de desenho.**
- **Objetos `Graphics` controlam como os desenhos são feitos.**
- **Objetos `Graphics` permitem:**
  - **desenhar formas geométricas;**
  - **manipular estilos, tamanhos e tipos de fontes;**
  - **manipular cores;**
- **Objeto do tipo `Graphics` criado pela GUI e passado automaticamente ao `paint`**

DESENHO DE FIGURAS

- `public void drawLine (int x1, int y1, int x2, int y2)`
  - `g.drawLine(x1,x2,y1,y2)` **desenha no objeto** `g` **do tipo** `Graphics` **uma linha entre os pontos** (x1, y1) **e** (x2, y2)
- `public void drawRect (int x, int y, int width, int height)`
  - **Desenha um retângulo com a largura (**width**) e a altura (**height**) especificadas.**
- `public void fillRect(int x, int y, int width, int height)`
  - **Desenha um retângulo sólido com a largura e altura especificadas, com canto superior esquerdo no ponto** (x, y)**.**

DESENHO DE FIGURAS ...

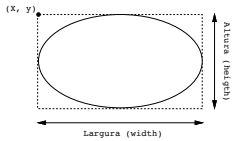
- `public void clearRect (int x, int y, int width, int height)`
  - **Desenha um retângulo transparente, ou seja, com a cor da borda e a cor de fundo igual a cor do componente sobre o qual será desenhado. Usado para separar componentes.**
- `public drawRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)`
  - **Desenha um retângulo com cantos arredondados.**
- `public void fillRoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight)`
  - **Desenha um retângulo sólido, preenchido com a cor atual e com cantos arredondados.**

DESENHO DE FIGURAS ...

- `public void draw3DRect (int x, int y, int width, int height, boolean b)`
  - **Desenha um retângulo tridimensional em algo relevo quando** b **é** true **e em baixo relevo quando** b **é** false**.**
- `public void fill3DRect (int x, int y, int width, int height, boolean b)`
  - **Desenha um retângulo tridimensional preenchido com a cor atual em alto ou em baixo relevo de acordo com o valor do parâmetro** b**.**

DESENHO DE FIGURAS ...

- `public void drawOval (int x, int y, int width, int height)`
  - **Desenha uma oval cujos cantos tocam a parte central das bordas de um retângulo de canto superior esquerdo em** (x, y) **tendo largura e altura especificadas por** width **e** height **respectivamente.**
- `public void fillOval (int x, int y, int width, int height)`
  - **Desenha uma oval com o fundo preenchido na cor atual.**



CONTROLE DE CORES

- `public Color getColor( )`
  devolve um objeto Color que representa a cor de textos e linhas gráficos
- `public void setColor (Color c)`
  Configura a cor atual para desenho de textos e linhas gráficos.

DESENHO DE ARCOS

- `public void drawArc (int x, int y, int width, int height, int startAngle, int arcAngle)`
  - **Desenha um arco em relação às coordenadas do canto superior esquerdo do retângulo delimitador** `(x, y)` **com a largura (**`width`**) e a altura (**`height`**) especificadas.**
  - **O segmento de arco é desenhado iniciando em** `startAngle` **e varrendo** `arcAngle` **graus.**
- `public void fillArc (int x, int y, int width, int height, int startAngle, int arcAngle)`
  - **Desenha um arco sólido (isto é, um setor) em relação às coordenadas do canto superior esquerdo do retângulo delimitador** (x, y).

DESENHO DE POLÍGONOS

- `public void drawPolyline(int xPoints[], int yPoints[], int npts)`
  - **Desenha uma série de linhas conectadas.**
  - **Se o último ponto definido por um par** `(x, y)`**,** $(\mathbf{x} \in \mathrm{xValues}$ **e** $\mathbf{y} \in \mathrm{yValues})$ **for diferente do primeiro ponto, a polilinha não será fechada.**
  - `npts` **dá o número de pontos**

DESENHO DE POLÍGONOS ...

- `public void drawPolygon (Polygon p)`
  - **Desenha o poligono fechado especificado .**
- `public void fillPolygon (Polygon p)`
  - **Desenha o polígono especificado preenchido com a cor corrente do contexto gráfico.**
- `public void fillPolygon(int xPoints[],int yPoints[],int npts)`
  - **Desenha um polígono sólido cujas coordenadas são especificadas pelos** *arrays* $\mathrm{xPoints}$ **e** $\mathrm{yPoints}$.
  - **O último argumento especifica o número de pontos.**
  - **Esse método desenha um polígono fechado – mesmo se o último ponto for diferente do primeiro ponto.**

DESENHO DE POLÍGONOS ...

- `public void drawPolygon(int xPoints[],int yPoints[],int npts)`
  - **Desenha um polígono.**
  - **A abscissa** `x` **de cada ponto é especificada pelo array** `xPoints` **e a ordenada** `y` **de cada ponto é especificada no array** `yPoints`**.**
  - **O último argumento especifica o número de pontos, ou seja, o número de vértices do polígono.**
  - **Este método desenha um polígono fechado, mesmo que o último ponto seja diferente do primeiro ponto.**

CONTROLE DE FONTES

- public Font getFont( )
  - **Retorna uma referência de objeto** Font **representando a fonte atual.**
- public void setFont(Font f)
  - **Configura a fonte atual com a fonte, o estilo e o tamanho especificados pela referência ao objeto** Font f**.**

CLASSE `POLYGON`

MÉTODOS PARA CRIAR POLÍGONOS

- `public Polygon()`:
  - **Constrói um novo objeto do tipo polígono. O polígono recém contruído não contém nenhum ponto.**
- `public Polygon(int xValues[], int yValues[], int npts)`:
  - **Constrói um novo objeto do tipo polígono com** `npts` **vértices, sendo cada vértice é formado por uma coordenada** `x` **do** *array* `xValues` **e uma coordenada** `y` **do** *array* `yValues`**.**
- `addPoint(int x, int y)`:
  Adiciona um novo ponto ao polígono

```
Polygon p = new Polygon ( );
p.addPoint(10,10); p.addPoint(10,30); p.addPoint(20,30);
g.drawPolygon(p);
```

CLASSE FONT

MÉTODOS E CONSTANTES DE Font

- public final static int PLAIN
  - **Uma constante representando um estilo de fonte simples.**
- public static int BOLD
  - **Uma constante representando um estilo de fonte em negrito.**
- public static int ITALIC
  - **Uma constante representando um estilo de fonte em itálico.**

MÉTODOS E CONSTANTES DE Font ...

- public Font (String name, int style, int size)
  - **Cria um objeto Font com a fonte, o estilo e o tamanho especificados.**
  - name **pode ser um dos nomes lógicos:** "Monospaced", "SansSerif", "Serif", "Dialog", "DialogInput"
    Fontes acima são mapeadas para as existentes na máquina. Por exemplo, no Windows "SansSerif" **é mapeado para Arial**
  - style **pode ser** Font.PLAIN, Font.ITALIC, Font.BOLD
  - style **pode combinações:** Font.ITALIC + Font.BOLD
  - size **em pontos (1 ponto = 1/72 da polegada)**

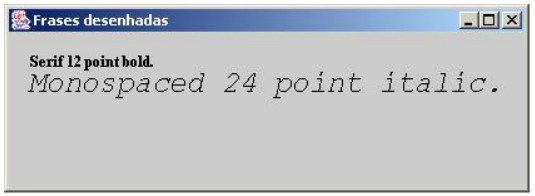
MÉTODOS E CONSTANTES DE Font ...

- public int getStyle()
  - **Retorna um valor inteiro indicando o estilo da fonte.**
- public int getSize()
  - **Retorna o tamanho da fonte.**
- public String FontName()
  - **Retorna o nome da fonte, por exemplo** "Helvetica Bold"
- public String getName()
  - **Retorna o nome lógico da fonte, por exemplo** "SansSerif"
- public String getFamily()
  - **Retorna o nome da família da fonte, por exemplo,**"Helvetica"

MÉTODOS E CONSTANTES DE Font ...

- public boolean isPlain ()
  - **Testa uma fonte quanto a um estilo de fonte simples. Retorna true se o estilo da fonte é simples (PLAIN).**
- public boolean isBold
  - **Testa uma fonte quanto a um estilo de fonte em negrito. Retorna true se a fonte estiver em negrito.**
- public boolean isItalic()
  - **Testa uma fonte quanto a um estilo de fonte em itálico. Retorna true se a fonte estiver em itálico.**

DEMONSTRANDO USO DE Font

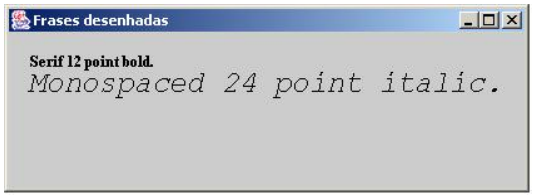


DEMONSTRANDO USO DE Font ...

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

public class DesenhaFrase extends JFrame {
    public DesenhaFrase() {
        super ("Frases desenhadas");
        setSize (420, 150);
        setVisible(true);
    }
    public void paint (Graphics g) { ...}
}
```


DEMONSTRANDO USO DE Font...

```
public void paint (Graphics g) {
    g.setFont (new Font ("Serif", Font.BOLD, 12));
    g.drawString ("Serif 12 point bold.", 20, 50);

    g.setFont (new Font ("Monospaced", Font.ITALIC, 24));
    g.drawString ("Monospaced 24 point italic.", 20, 70);
}
```



DEMONSTRANDO USO DE `Font` ...

```
import java.awt.*; import javax.swing.*;
import java.awt.event.*;
public class UsaDesenhaFrase {
   public static void main (String args[]) {
      DesenhaFrase j = new DesenhaFrase();
      j.addWindowListener(
          new WindowAdapter () {
              public void windowClosing (WindowEvent e) {
                   System.exit(0);
               }
            }
        );
   }
}
```


CLASSE `COLOR`

MÉTODOS DE `Color`

- public Color (int r, int g, int b)
  - **Cria uma cor com base no conteúdo de vermelho, verde e azul, expressos como inteiros entre 0 e 255.**
- public Color (float r, float g, float b)
  - **Cria uma cor com base no conteúdo de vermelho, verde e azul, expressos como valores de ponto flutuante entre 0.0 e 1.0**

CONSTANTES DE COR

Constante Color	Cor	Valor RGB
public final static Color orange	**laranja**	**255,200,0**
public final static Color pink	**cor-de-rosa**	**255,175,175**
public final static Color cyan	**ciano**	**0,255,255**
public final static Color magenta	**magenta**	**255,0,255**
public final static Color yellow	**amarelo**	**255,255,0**
public final static Color black	**preto**	**0,0,0**
public final static Color white	**branco**	**255,255,255**
public final static Color gray	**cinza**	**128,128,128**
public final static Color lightGray	**cinza-claro**	**192,192,192**
public final static Color darkGray	**cinza-escuro**	**64,64,64**
public final static Color red	**vermelho**	**255,0,0**
public final static Color green	**verde**	**0,255,0**
public final static Color blue	**azul**	**0,0,255**

OUTROS MÉTODOS DE `Color`

- public int getRed()
  - **Retorna um valor entre 0 e 255 representando o conteúdo de vermelho**
- public int getGreen()
  - **Retorna um valor entre 0 e 255 representando o conteúdo de verde**
- public int getBlue()
  - **Retorna um valor entre 0 e 255 representando o conteúdo de azul**

USO DE `Color`

```
import java.awt.*; import javax.swing.*;
public class RectColorido extends JFrame {
    public RectColorido() {
        super ("Retângulos coloridos");
        setSize (150, 80);
        setVisible(true);
    }
  public void paint (Graphics g) {
       g.setColor (new Color (255, 0, 0)); // vermelho
       g.fillRect (25, 25, 100, 20);
       g.setColor (new Color (0.0f, 1.0f, 0.0f )); // verde
       g.fillRect (25, 50, 100, 20);
  }
}
```


DEMONSTRANDO USO DE `Color`

```
import java.awt.*; import javax.swing.*;
import java.awt.event.*;
public class UsaRectColorido {
    public static void main (String args[]) {
        RectColorido j = new RectColorido();
        j.addWindowListener(
           new WindowAdapter () {
               public void windowClosing (WindowEvent e) {
                    System.exit(0);
                }
           }
        );
    }
}
```


DEMONSTRANDO USO DE `Color`

```
>java UsaRectColorido
```

COMPONENTES BÁSICOS DE INTERFACE

COMPONENTES BÁSICOS

- **Vimos como utilizar objetos da classe** `Graphics` **para desenhar formas geométricas e textos coloridos.**
- **Os programas vistos até então não dispunham de muitas formas de interagir com o usuário.**
- **A seguir serão estudados métodos e subclasses da classe** `Component`**, que possibilitam ao usuário interagir com programas gráficos em Java.**
- **Componentes GUI são objetos com os quais o usuário interage via mouse, teclado, voz, etc.**

COMPONENTE `JLabel`

- `JLabel` **(rótulo):**
  - **Uma área em que podem ser exibidos textos não-editáveis ou ícones.**
  - *Labels* **ou** *rótulos* **são utilizados para mostrar um pequeno texto que não pode ser alterado.**
  - **Rótulos comuns são vistos nomeando botões e caixas de texto.**
- **Métodos de** `JLabel`**:**
  - `JLabel( )`**: rótulo sem identificação**
  - `JLabel(String t)`**: cria rótulo com identificação** `t`
  - `void setText(String t)`**: define a identificação do rótulo**
  - `String getText( )`**: devolve a identificação do rótulo**

COMPONENTE `JButton`

- `JButton` **(botão):**
  Uma área que aciona um evento quando recebe um *click* **de mouse.**
- **Métodos de** `JButton`**:**
  - `JButton( )`**: cria botão sem identificação**
  - `JButton(Icon icon)`**: cria botão com o ícone**
  - `JButton(String text)`**: cria botão com o texto**

COMPONENTE `JTextField`

- `JTextField` **(caixa de texto):**
  - **Uma área em que o usuário insere dados via teclado. A área também pode exibir informações.**
  - *Caixas de Texto* **geralmente são uma parte de uma janela onde texto pode ser escrito.**
  - **Caixas de texto são um recurso utilizado nas interfaces gráficas para receber texto do usuário ou para transmitir texto para ele.**
- **Métodos de** `JTextField`**:**
  - `JTextField( )`
  - `JTextField(int colunas)`
  - `JTextField(String texto)`

COMPONENTES `JComboBox`, `JList`, `JPanel`

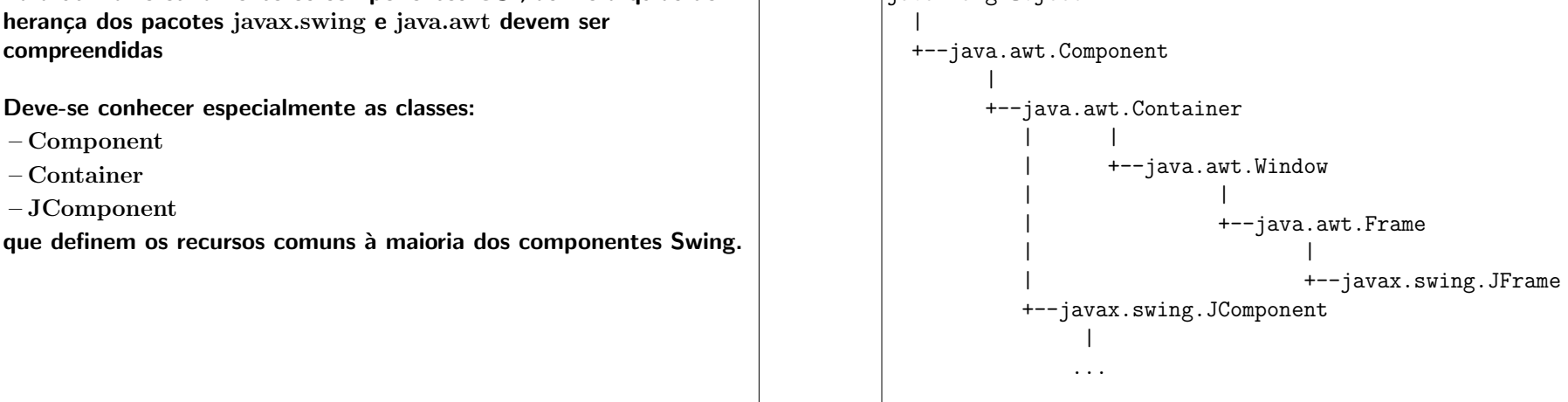
- `JComboBox` **(caixa de combinação):**
  Uma lista *drop-down* **de itens a partir da qual o usuário pode fazer uma seleção clicando em um item na lista ou digitando na caixa, se permitido.**
- `JList` **(lista):**
  Uma área em que uma lista de itens é exibida a partir da qual o usuário pode fazer uma seleção clicando uma vez em qualquer elemento na lista.
  Um clique duplo em um elemento na lista gera um evento de ação.
  Múltiplos elementos podem ser selecionados.
- `JPanel` **(painel):**
  Um contêiner em que os componentes podem ser colocados.

IMPORTANTES DE COMPONENTES SWING

- **Para utilizar efetivamente os componentes GUI, as hierarquias de herança dos pacotes** javax.swing **e** java.awt **devem ser compreendidas**
- **Deve-se conhecer especialmente as classes:**
  - Component
  - Container
  - JComponent

  que definem os recursos comuns à maioria dos componentes Swing.

HIERARQUIA DE JComponent



HIERARQUIA DE JComponent ...

```
javax.swing.JComponent
|  | ... |  |   |   |
|  |     |  |   |   +--- JPanel
|  |     |  |   +-- JComboBox
|  |     |  +--- Jlist
|  |     +--- JLabel
|  |
|  +--- AbstractButton
|           |
|           +--- JButton, JToggleButton, JMenuItem
+-- JTextComponent            |
        |                     + JRadioButton, JCheckBox
        +--- JTextField, JTextArea
```


CLASSE `Container`

- **Um** container **é uma coleção de componentes relacionados.**
- **Em aplicativos com** JFrames **e** *applets*, **anexamos os componentes ao painel de conteúdo – um** Container.
- Containers **possuem um leiaute que define como componentes gráficos serão posicionados em seu espaço.**

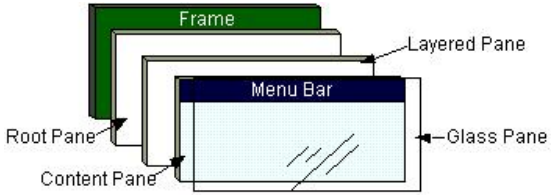
MÉTODOS DE `Container` I

- `add(Component c)`:
  adiciona o componente `c` **no fim do contêiner**
- `add(Component c, int posicao)`:
  adiciona o componente `c` **na posição indicada do contêiner**
- `setLayout(LayoutManager leiaute)`:
  define o leiaute a ser usado para posicionamento de componentes no contêiner.
  Leiaute definido por objeto do tipo:
  - `FlowLayout`
  - `BorderLayout`
  - `GridLayout`
- `setFont(Font f)`:
  define o fonte do contêiner

GERÊNCIA DE LAYOUT

ESTRUTURA DE UMA JANELA (JFrame)

- Uma janela é uma estrutura com várias camadas:



- ContentPane é usado para adicionar novos componentes à janela:

```
JFrame janela = new MinhaJanela();
Container c = janela.getContentPane();
c.setLayout (new FlowLayout( ));
c.add(componente);
```


ORGANIZAÇÃO DE COMPONENTES EM UMA GUI

- Posicionamento Absoluto:
  Sob o total controle pelo programador
- Gerente de Layout:
  Posicionamento automático
- Programação Visual com IDE:
  Caixa de ferramenta + arrastar e colar

POSICIONAMENTO ABSOLUTO

- É obtido configurando o leiaute do contêiner como null.
- Permite maior nível de controle da aparência do GUI.
- Permite especificar a posição absoluta de cada componente em relação ao canto superior esquerdo.
- É necessário que se especifique o tamanho de cada componente.

```
...
JFrame f = new MinhaFrame();
Component meuComponente = new MeuComponente( );
Container c = f.getContentPane();
c.setLayout(null);
c.add(meuComponente);
meuComponente.setBounds(x,y,largura,altura);
...
```


GERÊNCIA DE LEIAUTES

- *Gerentes de Leiaute* se prestam a organizar componentes GUI em um contêiner para propósito de apresentação.
- Retira-se do programador a tarefa de posicionar os itens em um contêiner fazendo o posicionamento automaticamente.
- Muitos ambientes de programação Java fornecem ferramentas GUI que ajudam o desenvolvedor a controlar a aparência de suas aplicações.
- Existem várias classes para gerenciar automática do leiaute: FlowLayout, BorderLayout, GridLayout, etc

CLASSE FlowLayout

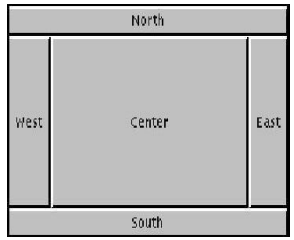
- **Dispõe os componentes seqüencialmente (da esquerda para a direita) na ordem que foram adicionados.**
- **Também é possível especificar a ordem dos componentes utilizando o método add de Container que aceita um Component e uma posição de índice inteiro como argumento.**

```
...
JFrame f = new MinhaFrame();
FlowLayout layout = new FlowLayout();
Container c = f.getContentPane();
c.setLayout(layout);
c.add(meuComponente)
...
```


CLASSE BorderLayout

- **Organiza os componentes em cinco áreas no painel: Norte, Sul, Leste, Oeste e Centro.**
- **Padrão para os painéis de JFrame.**

```
// 5 pixels de espacamento horizontal e
// 6 de espacamento vertical
// entre componentes.
JFrame f = new MinhaFrame();
BorderLayout b =  new BorderLayout(5,6);
Container c = f.getContentPane();
c.setLayout(b);
c.add(meuComponente,BorderLayout.SOUTH);
```



CLASSE GridLayout

- **Divide um painel em linhas e colunas.**
- **Componentes são colocados no painel de modo a preencher cada posição da esquerda para direira, de cima para baixo.**

```
private Container c = getContentPane();
private GridLayout grid;
// uma grade com 2 X 3, com espacamento 5 pixels X 5 pixels
// entre componentes
grid = new GridLayout (2, 3, 5, 5);
c.setLayout (grid);
for (int i = 0; i < NUM_BUT; i++) {
    b[i] = new JButton (...); b[i].addActionListener (...);
    c.add(b[i]);
}
```


MODELO DE EVENTOS

TRATAMENTO DE EVENTOS

- Quando um usuário interage com um componente da interface, um evento pode ser gerado.
- Eventos comuns:
  - mouse em movimento;
  - botão do mouse pressionado;
  - digitação de um campo de texto (ENTER);
  - abertura de uma janela;
  - seleção de um item de menu.
- Para processar um evento são necessárias duas tarefas:
  - o registro de um ouvinte de eventos;
  - a implementação de um tratador (handler) de eventos.

INTERFACE `ActionListener`

- Objetos ouvintes de eventos do tipo `ActionEvent` devem ser de classe que implementa `ActionListener`

```
interface ActionListener {
  public void actionPerformed (ActionEvent e)
}
```

- Métodos da classe `ActionEvent`:
  - `getSource( )`:
    retorna a referência do objeto originador do evento
  - `String getActionCommand( )` :
    retorna o string do comando associado com a ação, por exemplo, o rótulo de um botão

VISÃO GERAL DA MANIPULAÇÃO DE EVENTOS

- Um objeto ouvinte é instância de uma classe que implementa uma interface `EventListener`
- Uma origem de eventos é um objeto que pode registrar objetos ouvintes e a eles envia seus eventos
- A origem envia objetos eventos para todos os ouvintes dos eventos
- Os objetos ouvintes usam a informação do objeto evento recebido para determinar sua reação ao evento

EVENTOS E OUVINTES I

Interface	Métodos	Parâmetro/ Acessadores	Origem

`ActionListener`	`actionPerformed`	`ActionEvent`
		`getSource`
		`getActionCommand`
			`Button`
			`List`
			`MenuItem`
			`TextField`

Evento do tipo `ActionEvent` gerado por objeto:

- `Button` que recebe clique
- `List` que tem um item selecionado
- `MenuItem` que tem um item selecionado
- `TextField` que recebe um clique de ENTER

REGISTRO DO OUVINTE DE EVENTOS

- Cada objeto `x`, originador de eventos, do tipo JComponent faz referência a uma lista de tratadores de eventos a ele associados.
- O comando `x.addEventListener (eventListener)` associa um ouvinte (um tratador de eventos) chamado eventListener ao componente `x`.
  - Event acima pode ser: Action, Window, Mouse, MouseMotion, Key, etc
- Cada EventListener é uma interface (são 11)
- Quando um evento é disparado sobre algum objeto do tipo JComponent, sua lista de eventos é pesquisada em busca de um (ouvinte) tratador adequado.

JLabel e JTextField

- Este programa mostra dois componentes Swing e um tratador de eventos.

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;
```


USO DE Jlabel e JTextField

```
>java UsaEntradaDeTexto
```

USO DE JLabel e JTextField

```
public class EntradaDeTexto extends JFrame {
   private class TratadorDeEventosDeTexto
                        implements ActionListener{...}
   private JTextField text = new JTextField (30);
   private JLabel label = new JLabel ("Qual o seu nome?");
   EntradaDeTexto() {
      super ("labels e campos de texto");
      Container c = getContentPane();
      c.setLayout (new FlowLayout() );
      c.add (label); c.add (text);
      text.addActionListener(new TratadorDeEventosDeTexto());
      setSize (325, 100);  setVisible(true);
   }
}
```

USO DE JLabel e JTextField ...

- **classe interna para tratamento de evento implementada como uma classe aninhada da classe EntradaDeTexto**

```
private class TratadorDeEventosDeTexto implements
                    ActionListener {
    public void actionPerformed (ActionEvent e) {
        String s = "";
        if (e.getSource() == text)
            s = "Ola " + e.getActionCommand() + "!";
        JOptionPane.showMessageDialog (null, s);
    }
}
```


USO DE JLabel e JTextField ...

```
public class UsaEntradaDeTexto {
   public static void main (String args[]) {
      EntradaDeTexto j = new EntradaDeTexto();
      j.addWindowListener (
          new WindowAdapter () {
              public void windowClosing (WindowEvent e) {
                  System.exit(0);
              }
          }
      );
   }
}
```


EVENTOS E OUVINTES II

Interface	Métodos	Parâmetro/ Acessadores	Origem
WindowListener	windowClosing windowOpened windowIconified windowDeiconified windowClosed windowActivated windowDeactivated	WindowEvent getWindow	Window

Eventos de Janela ocorrem quando a janela:

- **é fechada com clique no X ou aberta;**
- **torna-se ativa com clique na janela ou torna-se inativa**
- **é minimizada (iconizada) com clique no □**
- **é restaurada (desiconizada) com clique no ícone**

CLASSES ADAPTADORAS

- **Constituem uma alternativa às interfaces ouvintes de eventos.**
- **Permitem ao programador implementar apenas os métodos necessários.**
- **A abordagem de interface obriga o programador a implementar todos os métodos da interface.**
- **Classes adaptadoras implementam interfaces deixando os métodos com o corpo vazio.**
- **O programador estende uma classe adaptadora e redefine apenas os métodos que precisar.**

CLASSE ADAPTADORA `WindowAdapter`

```
class WindowAdapter implements WindowListener {
    public void windowClosing(WindowEvent e) { }
    public void windowOpened(WindowEvent e) { }
    public void windowIconified(WindowEvent e) { }
    public void windowDeiconified(WindowEvent e) { }
    public void windowClosed (WindowEvent e) { }
    public void windowActivated (WindowEvent e) { }
    public void windowDeactivated(WindowEvent e) { }
}
```


CLASSES ADAPTADORAS DE EVENTOS

Classe Adaptadora	Interface Implementada
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

BOTÕES

BOTÕES

- **Um** *botão* **é um componente que o usuário precisa clicar para acionar uma ação específica.**
- **Existem vários tipos de botões:**
  - **botões de comando:**
    - `JButton`
  - **botões de estado:**
    - `JCheckBox`
    - `JRadioButton`

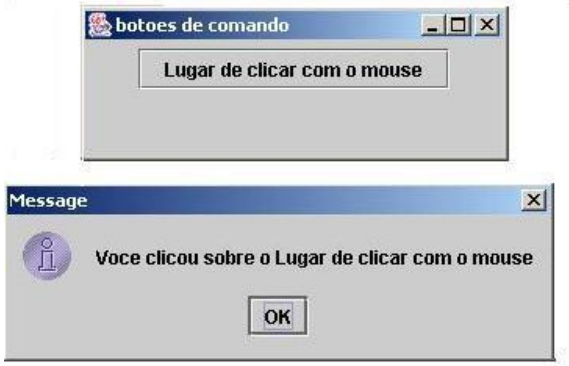
BOTÕES DE COMANDO

BOTÕES DE COMANDOS

- Botões de comando constituem um dos principais modos de interação entre o usuário e a aplicação.
- Um botão de comando é um componente que pode ser programado para desencadear um evento quando clicado pelo ponteiro de mouse ou acionado pelo usuário de alguma forma.
- Um botão de comando gera um `ActionEvent` quando pressionado com o mouse.
- Os botões de comando podem ter palavras como rótulos ou figuras, as quais permitem a implementação de interfaces de uso bastante simples e intuitivo.

USO DE BOTÕES DE COMANDO

```
>java UsaExemploBotao
```



BOTÕES DE COMANDO

```
import java.awt.*;import java.awt.event.*;import javax.swing.*;
public class ExemploBotao extends JFrame {
    private class TratadorDeEventosDeBotao
                    implements ActionListener{ ... }
    private JButton botao =
        new JButton ("Lugar de clicar com o mouse");
    public ExemploBotao() {
        super ("botoes de comando");
        Container c = getContentPane();
        c.setLayout (new FlowLayout() );  c.add (botao);
        botao.addActionListener(new TratadorDeEventosDeBotao());
        setSize (275, 100);  setVisible(true);
    }
}
```

BOTÕES DE COMANDO ...

```
private class TratadorDeEventosDeBotao
              implements ActionListener {
    public void actionPerformed (ActionEvent e){
        JOptionPane.showMessageDialog (null,
           "Voce clicou sobre o " + e.getActionCommand() );
    }
}
```


BOTÕES DE COMANDO ...

```
import java.awt.*; import java.awt.event.*;
import javax.swing.*;
public class UsaExemploBotao {
    public static void main (String args[]){
        ExemploBotao j = new ExemploBotao();
        j.addWindowListener (
            new WindowAdapter () {
                public void windowClosing (WindowEvent e) {
                    System.exit(0);
                }
            }
        );
    }
}
```


BOTÕES DE ESTADO

- **Botões de estado são caracterizados por valores de estado: ativado/desativado, verdadeiro/falso, etc.**
- **O pacote** Swing **disponibiliza dois tipos de botões de estado:**
  - JCheckBox
  - JRadioButton
- **Botões de estado geralmente são utilizados para selecionar uma opção (** JRadioButton**) ou várias opções (**JCheckBox**).**
- **As classes** JRadioButton **e** JCheckBox **são subclasses de** JToggleButton**.**

EVENTOS E OUVINTES III

Interface	Método	Parâmetro/ Acessadores	Origem
ItemListener	itemState-Changed	ItemEvent getItem getItemSelectable getStateChange	CheckBox CheckBoxMenuItem Choice List

- `ItemEvent` **ocorre quando o usuário faz uma seleção num conjunto de caixas de seleção ou itens de uma lista**
- `ItemEvent.SELECTED`**: indica que item foi selecionado**
- `ItemEvent.DESELECTED`**: indica que item foi desselecionado**
- `int getStateChange()`**: retorna o tipo de mudança de estado (**`SELECTED` **ou** `DESELECTED`**)**

SELEÇÃO DE OPÇÕES

USO DE JCheckBox

```
>java UsaDiversasOpcoes
```

JCheckBox

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class DiversasOpcoes extends JFrame {
  private class ControladorDeOpcoes implements ItemListener{...}
  private JCheckBox bold = new JCheckBox("Bold");
  private JCheckBox italic = new JCheckBox ("Italic");
  private JTextField t =
     new JTextField ("Olha como que as letras mudam", 20);
  public DiversasOpcoes () { ... }
}
```


JCheckBox ...

```
public DiversasOpcoes () {
      Container c = getContentPane();
      ControladorDeOpcoes cdo = new ControladorDeOpcoes();
      c.setLayout (new FlowLayout());
      t.setFont(new Font ("TimesRoman", Font.PLAIN, 14));
      c.add(t);   c.add (bold);   c.add(italic);
      bold.addItemListener (cdo);
      italic.addItemListener (cdo);
      setSize (275, 100);
      setVisible(true);
}
```

JCheckBox ...

```
private class ControladorDeOpcoes implements ItemListener {
  private int valBold = Font.PLAIN, valItalic = Font.PLAIN;
  public void itemStateChanged (ItemEvent e){
    if (e.getSource() == bold)
       if (e.getStateChange() == ItemEvent.SELECTED)
            valBold = Font.BOLD;
       else valBold = Font.PLAIN;
    if (e.getSource() == italic)
       if (e.getStateChange()==ItemEvent.SELECTED)
            valItalic = Font.ITALIC;
       else valItalic = Font.PLAIN;
    t.setFont(new Font("TimesRoman",valBold + valItalic, 14));
    t.repaint();
}}
```


JCheckBox ...

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class UsaDiversasOpcoes {
  public static void main (String args[]) {
     DiversasOpcoes j = new DiversasOpcoes();
     j.addWindowListener (
         new WindowAdapter () {
             public void windowClosing (WindowEvent e) {
                    System.exit(0);
             }
         }
     ); }
}
```

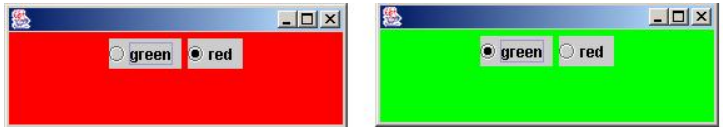

SELEÇÃO EXCLUSIVA

SELEÇÃO DE OPÇÕES MUTUAMENTE EXCLUSIVAS

- **JRadioButtons são usados quando deseja-se oferecer ao usuário um conjunto de opções dentre as quais somente uma pode ser escolhida**
- **Um conjunto de JRadioButtons mutuamente exclusivos deve ser agrupado por meio da classe ButtonGroup.**
- **Cada um dos botões que mutuamente se excluem deve ser adicionado a uma instância da classe ButtonGroup.**

USO DE JRadioButton

```
>java UsaUmaOpcao
```



JRadioButton

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

public class UmaOpcao extends JFrame {
  private class TratadorDeRBut implements ItemListener {...}
  private TratadorDeRBut controlador = new TratadorDeRBut();
  private JRadioButton green = new JRadioButton("green",false);
  private JRadioButton red   = new JRadioButton("red",true);
  private JButtonGroup radioGroup = new ButtonGroup();
  private Container c = getContentPane();

  public UmaOpcao() {...}
}
```


JRadioButton...

```
public UmaOpcao() {
        c.setBackground (Color.red);
        c.add (green);
        c.add (red);

        red.addItemListener (controlador);
        green.addItemListener (controlador);

        radioGroup.add(red);
        radioGroup.add(green);

        setSize (275, 100);
        setVisible(true);
}
```


JRadioButton ...

```
private class TratadorDeRBut implements ItemListener {
    public void itemStateChanged (ItemEvent e) {
        if (e.getSource() == red)
            c.setBackground (Color.red);
        else if (e.getSource() == green)
            c.setBackground (Color.green);
    }
}
```

JRadioButton ...

```
import java.awt.*; import java.awt.event.*;
import javax.swing.*;
public class UsaUmaOpcao {
   public static void main (String args[]) {
      UmaOpcao j = new UmaOpcao();
      j.addWindowListener (
          new WindowAdapter () {
              public void windowClosing (WindowEvent e) {
                  System.exit(0);
              }
          }
      );
  }
}
```


SELEÇÃO EM LISTAS

SELEÇÃO DE ITENS DE LISTA

- **Estes componentes fornecem listas de itens para o usuário fazer uma escolha.**
- JComboBox:
  - **permite ao usuário escolher 1 item de uma lista**
  - **geram eventos percebidos por** itemListener.
- JList:
  - **pode ser usada para implementar listas de múltiplas seleções.**
  - **geram eventos percebidos por** listSelectionListener.

CAIXA DE COMBINAÇÃO (JComboBox)

- **Itens da lista são identificados por índices, sendo 0 (zero) o primeiro**
- **Primeiro item adicionado à lista aparece como item selecionado quando JComboBox for exibida**
- **Para cada seleção são gerados dois eventos: um para o item desselecionado e outro para o selecionado**

```
private String names[] = {"desenho1", "desenho2, "desenho3"};
Container c = getContentPane();

private JComboBox images = new JComboBox (names);
images.setMaximumRowCount(12);
c.add (images);
```

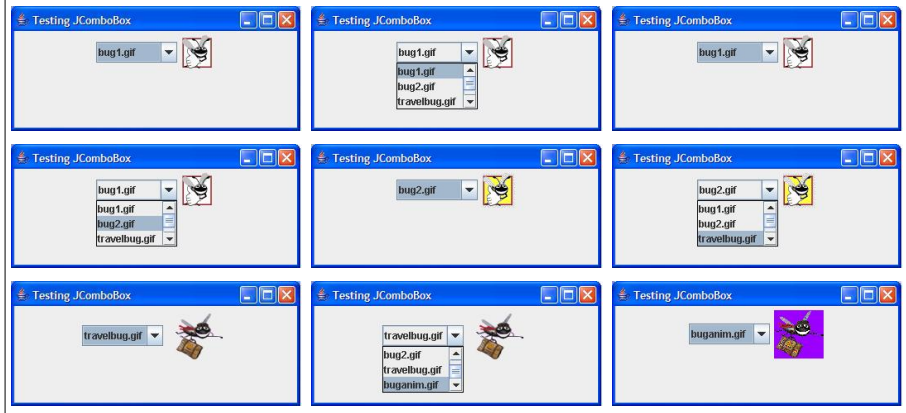
OPERAÇÕES DE JComboBox

- `JComboBox(String[ ] names ):`
- `setMaximumRowCount( int n ):`
  define número de linhas visíveis da caixa de combinação. Se necessário barras de rolagem serão automaticamente incluídas
- `int getSelectedIndex( ):`
  índice dos itens inicia-se com 0

JComboBoxFame

```
import javax.swing.JFrame;

public class ComboBoxTest {
   public static void main( String args[] ) {
        ComboBoxFrame c = new ComboBoxFrame();
        c.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        c.setSize( 350, 150 );
        c.setVisible( true );
 }
}
```


JComboBoxFrame



Importações Necessárias à Classe `ComboBoxFrame`

```
import java.awt.FlowLayout;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JComboBox;
import javax.swing.Icon;
import javax.swing.ImageIcon;
```

Classe ComboBoxFrame

```
public class ComboBoxFrame extends JFrame {
   private JComboBox comboBox;
   private JLabel label;
   private String names[] = {
      "bug1.gif","bug2.gif","travelbug.gif","buganim.gif"
   };
   private Icon icons[] = {
      new ImageIcon( names[ 0 ]),
      new ImageIcon( names[ 1 ]),
      new ImageIcon( names[ 2 ]),
      new ImageIcon( names[ 3 ])
   };
   public ComboBoxFrame( ){ ... }
}
```


Construtora de ComboBoxFrame

```
public ComboBoxFrame( ){
   super("Testing JComboBox");
   setLayout( new FlowLayout( ) );

   comboBox = new JComboBox( names );
   comboBox.setMaximumRowCount( 3 );//exibe três linhas

   "ADICIONA OUVINTE ItemListener A comboBox" ;

   add(comboBox );
   label = new JLabel(icons[ 0 ]); //exibe primeiro ícone
   add(label);
}
```


ADICIONA OUVINTE ItemListener A comboBox

```
comboBox.addItemListener(
   new ItemListener( ) {
       public void itemStateChanged(ItemEvent event){
         if (event.getStateChange( ) == ItemEvent.SELECTED) {
            label.setIcon(icons[comboBox.getSelectedIndex( )]);
         }
       }
   }
)
```


SELEÇÃO SIMPLES EM UMA JLIST

OPERAÇÕES DE JList

- `JList( Object[ ] itemNames)`:
  mostra na lista os elementos de `itemNames`**, normalmente convertidos com** `toString`**.**
- `setVisibleRowCount( int n )`
- `int setSelectionMode(int mode)`:
  define o modo de seleção de itens, que pode ser `SINGLE_SELECTION`, `SINGLE_INTERVAL_SELECTION` **ou** `MULTIPLE_INTERVAL_SELECTION`
- `setListData(Object[ ] itens )`:
  configura os itens dados, por default convertidos com `toString`**, no Jlist corrente**
- `setCellRenderer(ListCellRenderer renderer)`:
  delega ao método `renderer.getListCellRendererComponent(...)`**, no lugar de** `toString`**, a produção de cada um dos itens da JList**

OPERAÇÕES DE JList ...

- `int getSelectedIndex( ) :`
- `int[ ] getSelectedIndices( ) :`
- `Object getSelectedValue( )`:
  retorna o rótulo do item selecionado
- `Object[ ] getSelectedValues( )`:
  retorna os rótulos dos itens selecionados
- `setFixedCellWidth( int w )`
- `setFixedCellHeight( int h )`

VALORES DO SelectionMode

- `static int ListSelectionModel.SINGLE_SELECTION`:
  permite selecionar de um único item de cada vez
- `static int ListSelectionModel.SINGLE_INTERVAL_SELECTION`
  permite selecionar um intervalo contíguo de itens com o auxílio da teclas SHIFT
- `static int ListSelectionModel.MULTIPLE_INTERVAL_SELECTION`:
  permite seleção de intervalos contíguo com o auxílio da tecla CTL

Seleção de Uma Cor de Uma JList

```
import javax.swing.JFrame;
public class ListTest {
  public static void main( String args[] )  {
    ListFrame listFrame = new ListFrame();
    listFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    listFrame.setSize( 350, 150 );
    listFrame.setVisible( true );
  }
}
```

Seleção de Uma Cor de Uma JList...

```
import java.awt.FlowLayout;
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.ListSelectionModel;
```


Seleção de Uma Cor de Uma JList...

```
public class ListFrame extends JFrame {
   private JList colorJList;
   private final String colorNames[] = { "Black", "Blue",
      "Cyan", "Dark Gray", "Gray", "Green", "Light Gray",
      "Magenta","Orange", "Pink", "Red", "White", "Yellow"
   };
   private final Color colors[] = { Color.BLACK, Color.BLUE,
      Color.CYAN, Color.DARK_GRAY, Color.GRAY, Color.GREEN,
      Color.LIGHT_GRAY, Color.MAGENTA, Color.ORANGE, Color.PINK,
      Color.RED, Color.WHITE, Color.YELLOW
   };
   public ListFrame(){ ... }
}
```


CONSTRUTORA DE ListFrame

```
public ListFrame() {
   super( "List Test" );  setLayout(new FlowLayout( ));

   colorJList = new JList( colorNames );
   colorJList.setVisibleRowCount( 5 );

   // não permite múltiplas seleções
   colorJList.setSelectionMode(
                   ListSelectionModel.SINGLE_SELECTION);

   "ADICIONA OUVINTE ListSelectionListener A colorJList";

   add( new JScrollPane( colorJList ) );
}
```

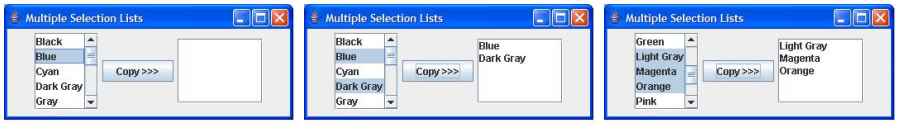

ADICIONA OUVINTE ListSelectionListener A colorJList

```
colorJList.addListSelectionListener(
   new ListSelectionListener() {
       public void valueChanged(ListSelectionEvent event){
          getContentPane().setBackground(
                  colors[colorJList.getSelectedIndex()] );
       }
   }
);
```

SELEÇÃO MÚLTIPLA EM UMA JLIST I

SELEÇÃO DE VÁRIAS CORES DE UMA LISTA

```
import javax.swing.JFrame;
public class MultipleSelectionTest {
   public static void main( String args[] ) {
      MultipleSelectionFrame m = new MultipleSelectionFrame();
      m.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE );
      m.setSize( 350, 140 );
      m.setVisible( true );
   }
}
```



SELEÇÃO DE VÁRIAS CORES DE UMA LISTA ...

```
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JButton;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
```


CLASSE MultipleSelectionFrame

```
public class MultipleSelectionFrame extends JFrame {
   private JList colorJList;
   private JList copyJList;
   private JButton copyJButton;

   private final String colorNames[] = { "Black", "Blue",
      "Cyan", "Dark Gray", "Gray", "Green", "Light Gray",
      "Magenta", "Orange", "Pink", "Red", "White", "Yellow"
   };

   public MultipleSelectionFrame() { ... }
}
```

CONSTRUTORA DE MultipleSelectionFrame

```
public MultipleSelectionFrame() {
   super( "Multiple Selection Lists" );
   setLayout( new FlowLayout() );

   "CONSTRÓI LISTA DE SELEÇÃO colorJList";
   add( new JScrollPane( colorJList ) );

   copyJButton = new JButton( "Copy >>>" );
   "ADICIONA OUVINTE ActionListener A copyJButton";
   add( copyJButton );

   "CONSTRÓI LISTA copyList";
   add( new JScrollPane( copyJList ) );
}
```


CONSTRÓI LISTA DE SELEÇÃO colorJList

```
colorJList = new JList( colorNames );

colorJList.setVisibleRowCount( 5 );

colorJList.setSelectionMode(
        ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
```


ADICIONA OUVINTE ActionListener A copyJButton

```
copyJButton.addActionListener(
   new ActionListener() {
       public void actionPerformed(ActionEvent event) {
          copyJList.setListData(colorJList.getSelectedValues());
       }
   }
);
```

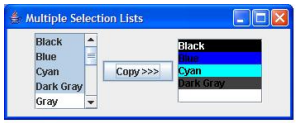

CONSTRÓI LISTA copyJList

```
copyJList = new JList();

copyJList.setVisibleRowCount( 5 );

copyJList.setFixedCellWidth( 100 );

copyJList.setFixedCellHeight( 15 );

copyJList.setSelectionMode(
       ListSelectionModel.SINGLE_INTERVAL_SELECTION);
```

SELEÇÃO MÚLTIPLA EM UMA JLIST II

SELEÇÃO DE VÁRIAS CORES DE UMA LISTA

```
import javax.swing.JFrame;
public class MultipleSelectionTest {
   public static void main( String args[] ) {
      MultipleSelectionFrame m = new MultipleSelectionFrame();
      m.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE );
      m.setSize( 350, 140 );
      m.setVisible( true );
   }
}
```



SELEÇÃO DE VÁRIAS CORES DE UMA LISTA ...

```
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JButton;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
import javax.swing.ListCellRenderer
```


CLASSE MultipleSelectionFrame

```
public class MultipleSelectionFrame extends JFrame {
   private JList colorJList, JList copyJList;
   private JButton copyJButton;
   private final String colorNames[] = { "Black", "Blue",
      "Cyan", "Dark Gray", "Gray", "Green", "Light Gray",
      "Magenta", "Orange", "Pink", "Red", "White", "Yellow"
   };
   private final Color colors[] = { Color.black, Color.blue,
      Color.cyan, Color.darkGray, Color.gray, Color.green,
      Color.lightGray, Color.magenta, Color.orange, Color.pink,
      Color.red, Color.white, Color.yellow
   };
   public MultipleSelectionFrame() { ... }
}
```

CONSTRUTORA DE MultipleSelectionFrame

```
public MultipleSelectionFrame() {
   super( "Multiple Selection Lists" );
   setLayout( new FlowLayout() );

   "CONSTRÓI LISTA DE SELEÇÃO colorJList";
   add( new JScrollPane( colorJList ) );

   copyJButton = new JButton( "Copy >>>" );
   "ADICIONA OUVINTE ActionListener A copyJButton";
   add( copyJButton );

   "CONSTRÓI LISTA copyList";
   add( new JScrollPane( copyJList ) );
}
```


CONSTRÓI LISTA DE SELEÇÃO colorJList

```
colorJList = new JList( colorNames );

colorJList.setVisibleRowCount( 5 );

colorJList.setSelectionMode(
        ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
```


ADICIONA OUVINTE ActionListener A copyJButton

```
copyJButton.addActionListener(
   new ActionListener() {
       public void actionPerformed(ActionEvent event) {
          copyJList.setListData(colorJList.getSelectedValues());
       }
   }
);
```


CONSTRÓI LISTA copyJList

```
copyJList = new JList();

copyJList.setVisibleRowCount( 5 );

copyJList.setFixedCellWidth( 100 );

copyJList.setFixedCellHeight( 15 );

copyJList.setSelectionMode(
       ListSelectionModel.SINGLE_INTERVAL_SELECTION);

copyJList.setCellRenderer(new MyRenderer(colors));
```

Produtor de Itens de JList (Versão I)...

```
interface ListCellRenderer {
   public Component getListCellRendererComponent(
          JList list, Object value, int index,
          boolean isSelected, boolean cellHasFocus);
}
class MyRenderer extends implements ListCellRenderer {
   private Color[ ] colors;
   public MyCellRenderer(Color[ ] colors) {
      this.colors = colors;
   }
   public Component getListCellRendererComponent(
          JList list, Object value, int index,
          boolean isSelected, boolean cellHasFocus){...}
}
```


...Produtor de Itens de JList (Versão I)

```
public Component getListCellRendererComponent(
       JList list, Object value,  int index,
       boolean isSelected, boolean cellHasFocus)  {
   index = index%colors.length;
   JButton item = new JButton( );
   item.setText(value.toString());
   item.setOpaque(true);
   item.setBackground(colors[index]);
   item.setForeground((colors[index] == Color.black)?
                                    Color.white : Color.black);
   return item;
}
```


Produtor de Itens de JList (Versão II)...

```
interface ListCellRenderer {
   public Component getListCellRendererComponent(
          JList list, Object value, int index,
          boolean isSelected, boolean cellHasFocus);
}
class MyRenderer extends JLabel implements ListCellRenderer {
   private Color[ ] colors;
   public MyCellRenderer(Color[ ] colors) {
      this.colors = colors;
   }
   public Component getListCellRendererComponent(
          JList list, Object value, int index,
          boolean isSelected, boolean cellHasFocus){...}
}
```


...Produtor de Itens de JList (Versão II)

```
public Component getListCellRendererComponent(
       JList list, Object value,  int index,
       boolean isSelected, boolean cellHasFocus)  {
   setText(value.toString()); index = index%colors.length;
   setBackground(colors[index]); setOpaque(true);
   setForeground((colors[index] == Color.black)?
                                    Color.white : Color.black);
   return this;
}
```

- **Retorna o componente a ser exibido como o item de posição** `index` **da** `JList list`
- **Automaticamente chamado para cada item de** `list`
- **Na falta de objeto** `ListCellRenderer` **associado ao** `JList`**, o componente exibido na posição** `index` **do** `JList` **é** `value.toString()`

PAINÉIS

CLASSE `Jpanel`

- **Painéis são usados como uma base onde os componentes gráficos podem ser fixados, inclusive outros painéis.**
- **Painéis são criados com** JPanel **– uma subclasse de** JComponent**:**
  - **cria-se uma subclasse de** `JPanel`
  - **sobrepõe-se o método** `paintComponent(Graphics g)`
  - **adicionam-se componentes ao painel com** `add(Component c)`
- `paintComponent` **não deve ser chamado diretamente. Use** `repaint( )`
- **Para limpar a área de desenho, dentro de** `paintComponent` **deve-se chamar** `super.paintComponent(g)`
- `paint(Graphics g)` **desenha no** `JFrame`
- `paintComponent(Graphics g)` **desenha no** `JPanel`**, que pode ter sido adicionado a um** `JFrame`

MÉTODOS DA CLASSE `JPanel`

- `JPanel( )`:
  cria painel com `FlowLayout`
- `void setLayout(LayoutManager layout)`:
  associa leiaute ao painel
- `JPanel(LayoutManager layout)`:
  cria painel com o leiaute indicado
- `add(Component c)`:
  Adiciona o componente gráfico `c` **no painel.**

USANDO PAINÉIS

```
>java UsaPanelDemo
```

USANDO PAINÉIS ...

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

public class PanelDemo extends JFrame {
    private JPanel  buttonPanel = new JPanel();
    private JButton buttons[]   = new JButton[5];

    public PanelDemo () { ...}

}
```


USANDO PAINÉIS ...

```
public PanelDemo () {
    super ("Demonstrando Paineis");
    Container c = getContentPane();
    buttonPanel.setLayout(new GridLayout (2,3));
    for (int i = 0; i < buttons.length; i++) {
         buttons[i] = new JButton ("Button " + (i + 1));
         buttonPanel.add (buttons[i]);
    }
    c.add (buttonPanel, BorderLayout.SOUTH);
    setSize (425, 150);
    setVisible(true);
}
```


USANDO PAINÉIS ...

```
public class UsaPanelDemo {
   public static void main (String args[]){
     PanelDemo j = new PanelDemo();
     j.addWindowListener (
         new WindowAdapter() {
             public void windowClosing (WindowEvent e) {
                    System.exit(0);
             }
         }
     );
   }
}
```


JPanel PERSONALIZADO

```
>java UsaPainelPersonalizado
```

ESTENDENDO JPanel

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

public class PainelPersonalizado extends JPanel {
    public void paintComponent (Graphics g) {
        super.paintComponent (g);
        g.fillOval(50, 10, 60, 60);
    }
}
```


JPanel PERSONALIZADO ...

```
import java.awt.*; import java.swing.event.*;
import javax.swing.*;
public TestePainelPersonalizado extends JFrame {
    public TestePainelPersonalizado () {
        super ("Teste do painel personalizado");
        PainelPersonalizado meuPainel =
                    new PainelPersonalizado();
        meuPainel.setBackground (Color.yellow);
        Container c = getContentPane();
        c.add(meuPainel, BorderLayout.SOUTH);
        setSize(300, 150); setVisible(true);
    }
}
```


JPanel PERSONALIZADO ...

```
import java.awt.*; import java.awt.event.*;
import javax.swing.*;
public class UsaPainelPersonalizado {
  public static void main (String args[]) {
    TestePainelPersonalizado j = new TestePainelPersonalizado();
        j.addWindowListener (
            new WindowAdapter () {
               public void windowClosing (WindowEvent e) {
                   System.exit(0);
               }
            }
         );
   }
}
```


REMOÇÃO E INCLUSÃO DE COMPONENTES

```
>java UsaRemoveComponente
```

REMOÇÃO E INCLUSÃO DE COMPONENTES...

```
import java.awt.*; mport java.awt.event.*;import javax.swing.*;
public class UsaRemoveComponente{
     public static void main (String args[]){
        RemoveComponente j = new RemoveComponente();
        j.addWindowListener (
            new WindowAdapter() {
                public void windowClosing (WindowEvent e) {
                    System.exit(0);
                }
            }
        );
    }

}
```


REMOÇÃO E INCLUSÃO DE COMPONENTES...

```
import java.awt.*;import java.awt.event.*;import javax.swing.*;
public class RemoveComponente extends JFrame
                                    implements ActionListener{
    private JPanel  buttonPanel = new JPanel();
    private JButton buttons[ ]  = new JButton[6];
    private JButton remocao2    = new JButton("remove 2");
    private JButton remocao4    = new JButton("remove 4");
    private JButton adicao2     = new JButton("adicao 2");
    private JButton adicao4     = new JButton("adicao 4");
    public RemoveComponente( ){...}
    public void actionPerformed(ActionEvent e) {...}
}
```


REMOÇÃO E INCLUSÃO DE COMPONENTES...

```
public RemoveComponente( ) {
    super ("Adição e Remoção de Componentes");
    buttonPanel.setLayout(new GridLayout (2,3));
    for (int i = 0; i < buttons.length; i++) {
          buttons[i] = new JButton ("Button " + i);
          buttonPanel.add(buttons[i]);
          buttonPanel.add(buttons[i]); // somente um inserido
    }
    "Adiciona Componentes na Janela";
    "Associa Ouvinte a Botões";
    setSize (425, 150);
    show();
}
```


ADICIONA COMPONENTS NA JANELA

```
    Container c = getContentPane();
    c.add(remocao2,BorderLayout.WEST);
    c.add(remocao4,BorderLayout.EAST);
    c.add(adicao2,BorderLayout.NORTH);
    c.add(adicao4,BorderLayout.CENTER);
    c.add(buttonPanel, BorderLayout.SOUTH);
```

ASSOCIA OUVINTE A BOTÕES

```
remocao2.addActionListener(this);
remocao4.addActionListener(this);
adicao2.addActionListener(this);
adicao4.addActionListener(this);
```


DEFINE AÇÃO DO OUVINTE

```
public void actionPerformed(ActionEvent e) {
       if  (e.getSource()== remocao2)
               buttonPanel.remove(buttons[2]);
       else if (e.getSource()== remocao4)
               buttonPanel.remove(buttons[4]);
       else if (e.getSource()== adicao2)
               buttonPanel.add(buttons[2]);
       else if (e.getSource()== adicao4)
               buttonPanel.add(buttons[4]);

       buttonPanel.repaint();
}
```


TRATAMENTO DE EVENTOS

Tipos de Evento

- **Eventos de Mouse**
- **Eventos de Teclado**

TRATAMENTO DE EVENTOS DE MOUSE

EVENTOS DE MOUSE

- **Eventos de mouse são capturados por objetos do tipo das interfaces:** `MouseListener`, `MouseMotionListener` **e** `MouseWheelListener`.
- **Podem ser disparados sobre qualquer componente GUI que deriva de** java.awt.Component.
- **São disparados por ações do mouse: clicar, arrastar, etc.**
- **Passam para a classe manipuladora de eventos as coordenadas** `(x,y)` **onde aconteceu o evento.**
- **Java disponibiliza duas interfaces e sete métodos virtuais para tratamento de eventos de mouse.**

INTERFACES MANIPULADORAS DE EVENTOS DE MOUSE

- MouseListener **captura eventos como cliques de mouse e mudança de posição do ponteiro do mouse.**
- MouseMotionListener **captura eventos que ocorrem quando o cursor do mouse é movido ou o botão do mouse é pressionado enquanto o cursor é movido.**
- `MouseWheelListener` **captura eventos gerados pela rotação da roda do mouse.**
- **Os métodos de** MouseListener **e** MouseMotionListener **são chamados quando têm-se as três condições:**
  1. **o mouse gera o evento**
  2. **o mouse interage com um componente**
  3. **objetos ouvintes apropriados foram registrados no componente**

EVENTOS E OUVINTES IV

Interface	Método	Parâmetro/ Acessadores	Origem

`MouseListener`	`mousePressed`
	`mouseReleased`
	`mouseEntered`
	`mouseExited`
	`mouseClicked`
		`MouseEvent`
		`getClickCount`
		`getX , getY`
		`getPoint`
		`getButton`
		`getModifiers`
		`getMouseModifierText`
		`isPopupTrigger`
			`Component`

`MouseMotion-Listener`	`mouseDragged` `mouseMoved`	`MouseEvent`	`Component`
`MouseWheel-Listener`	`mouseWheelMoved`	`MouseWheelEvent` `...`	`Component`

CLASSE MouseEvent

- int getClickCount( ):
  retorna número de cliques rápidos e consecutivos
- int getX( ), int getY( ) e Point getPoint( ):
  retornam coordenadas (x,y), relativas ao componente, do mouse onde evento ocorreu
- int getButton( ):
  retorna qual botão do mouse mudou de estado. Valor retornado é uma das constantes: `NOBUTTON`, `BUTTON1`, `BUTTON2`, `BUTTON3`

CLASSE MouseEvent ...

- int getModifier():
  retorna um int descrevendo as teclas modificadoras (Shift, Ctl+Shift, etc) ativas durante evento
- String getMouseModifierText(int):
  retorna um String descrevendo as teclas modificadoras (Shift, Ctl+Shift, etc) contidas no int
- boolean isPopUpTrigger( ):
  retorna `true` se o evento de mouse deve causar o aparecimento de um menu pop-up, conforme definido na plataforma. Verificar na plataforma como esse tipo de evento é disparado.

MÉTODOS DE MouseListener

- public void mousePressed (MouseEvent e)
  - botão do mouse pressionado com cursor sobre um componente.
- public void mouseClicked (MouseEvent e)
  - botão do mouse pressionado e liberado sobre um componente sem movimentação do cursor (click).
- public void mouseReleased (MouseEvent e)
  - botão do mouse liberado depois de ser pressionado. Sempre precedido por mousePressed.

MÉTODOS DE MouseListener

- public void mouseEntered (MouseEvent e)
  - chamado quando cursor do mouse entra nos limites de um componente.
- public void mouseExited (MouseEvent e)
  - chamado quando cursor do mouse deixa os limites de um componente.

MÉTODOS DE MouseMotionListener

- public void mouseDragged (MouseEvent e)
  - **botão do mouse é pressionado e o cursor é movido. Sempre precedido por** mousePressed.
- public void mouseMoved (MouseEvent e)
  - **mouse movido com o cursor sobre um componente.**

DEMONSTRANDO EVENTO DE MOUSE

```
>java UsaEvtMouse
```

DEMONSTRANDO EVENTO DE MOUSE ...

```
import java.awt.*;import java.awt.event.*;import javax.swing.*;
public class EvtMouseDemo extends JFrame
             implements MouseListener, MouseMotionListener {

  private JLabel status1 = new JLabel();
  private JLabel status2 = new JLabel();
  private JLabel status3 = new JLabel();

  private Container c = getContentPane();

  "CONSTRUTORA DE EvtMouseDemo";

  "Operações de tratamento de evento";
}
```


CONSTRUTORA DE EvtMouseDemo

```
  public EvtMouseDemo () {
      super ("Demonstrando Eventos de Mouse");

      c.add(status1,BorderLayout.SOUTH);
      c.add(status2,BorderLayout.NORTH);
      c.add(status3,BorderLayout.EAST);

      addMouseListener (this); addMouseMotionListener (this);

      setSize(275, 100); setVisible(true);
  }

}
```

OPERAÇÕES DE TRATAMENTO DE EVENTOS

```
public void mouseClicked (MouseEvent e) {
   status1.setText
    ("clique em [" + e.getX() + ", " + e.getY() + "]");
}

public void mousePressed (MouseEvent e) {
   status3.setText
    ("Botao pressionado em [" + e.getX()+ ", " + e.getY()+ "]");
}
```


OPERAÇÕES DE TRATAMENTO DE EVENTOS ...

```
public void mouseReleased (MouseEvent e) {
   status3.setText
     ("Botao liberado em [" + e.getX() + ", " + e.getY() + "]");
}

public void mouseEntered (MouseEvent e) {
   status2.setText ("Ponteiro dentro da janela");
}

public void mouseExited (MouseEvent e) {
   status2.setText ("Ponteiro fora da janela");
}
```


OPERAÇÕES DE TRATAMENTO DE EVENTOS ...

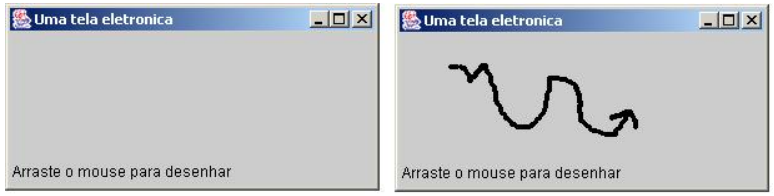
```
public void mouseDragged (MouseEvent e) {
   status1.setText ("Arrastando em [" +
              e.getX() + ", " + e.getY() + "]");
}

public void mouseMoved (MouseEvent e) {
   status1.setText ("Movendo em [" +
              e.getX() + ", " + e.getY() + "]");
}
```


DEMONSTRANDO EVENTO DE MOUSE: PRINCIPAL

```
public class UsaEvtMouse {
    public static void main (String args[]){
       EvtMouseDemo j = new EvtMouseDemo ();

       j.addWindowListener (
           new WindowAdapter() {
               public void windowClosing (WindowEvent e) {
                    System.exit(0);
               }
           }
       );
    }
}
```

UM PROGRAMA DE PINTURA

```
>java Painter
```



UM PROGRAMA DE PINTURA ...

```
public class Painter extends JFrame {
    private int x = -10; y = -10;
    public  Painter () {
        super ("Uma tela eletronica");
        getContentPane().add (
            new Label ("Arraste o mouse para desenhar"),
                                          BorderLayout.SOUTH);
        addMouseMotionListener (...);
        setSize (300, 150);  setVisible(true);
    }
    public void paint (Graphics g){g.fillOval(x,y,4,4);}
    public static void main (String args[]){...}
}
```


UM PROGRAMA DE PINTURA ...

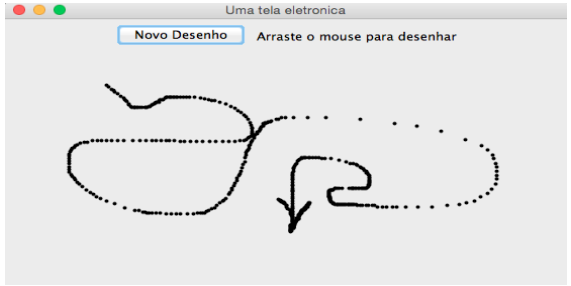
```
addMouseMotionListener (
     new MouseMotionAdapter() {
            public void mouseDragged (MouseEvent e) {
                    x = e.getX();
                    y = e.getY();
                    repaint();
            }
      }
);
```


UM PROGRAMA DE PINTURA ...

```
public static void main (String args[]){
     Painter j = new Painter ();
     j.addWindowListener (
         new WindowAdapter() {
             public void windowClosing (WindowEvent e){
                System.exit(0);
             }
        }
     );
}
```

OUTRO PROGRAMA DE PINTURA ...

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
>java Painter
```



...OUTRO PROGRAMA DE PINTURA ...

```
public class Painter extends JFrame{
    Painter( ){
      super("Uma tela eletronica");
      Tela tela = new Tela();
      Container t = this.getContentPane( );
      t.add(tela);
      setSize (600, 3000);
      setVisible(true);
    }

    public static void main (String args[]){ ... }
}
```


MAIN DE Painter

```
public static void main (String args[]){
    Painter j = new Painter( );
    j.addWindowListener (
        new WindowAdapter() {
            public void windowClosing (WindowEvent e) {
                System.exit(0);
            }
        }
    );
}
```


...OUTRO PROGRAMA DE PINTURA ...

```
class Tela extends JPanel {
    private int xValue = -10, yValue = -10;
    private boolean limpa = false;
    public Tela( ) {...}
    public void paintComponent (Graphics g) {
        g.fillOval (xValue, yValue, 4, 4);
        if (limpa){super.paintComponent(g); limpa = false;}
    }
}
```

CONSTRUTORA DE Tela

```
public Tela( ) {
  JButton botao = new JButton ("Novo Desenho");

  add(botao,BorderLayout.NORTH);

  add(new Label ("Arraste o mouse para desenhar"),
                                  BorderLayout.SOUTH);

  "ADICIONA OUVINTE PARA EVENTO DO MOUSE"

  "ADICIONA OUVINTE PARA O BOTÃO"

}
```


ADICIONA OUVINTE PARA EVENTO DO MOUSE

```
addMouseMotionListener (
        new MouseMotionAdapter() {
                public void mouseDragged (MouseEvent e) {
                    xValue = e.getX();
                    yValue = e.getY();
                    repaint();
                }
        }
);
```


ADICIONA OUVINTE AO BOTÃO

```
botao.addActionListener(
       new ActionListener( ) {
               public void actionPerformed (ActionEvent e) {
                  limpa = true;
                  repaint();
               }
        }
);
```


TRATAMENTO DE EVENTOS DE TECLADO

EVENTOS DE TECLADO

- **Eventos de teclado são gerados quando as teclas são pressionadas e liberadas.**
- `KeyListener` **é a interface ouvinte de eventos de teclado.**
- `KeyListener` **tem três métodos:**
  - `keyPressed` **é chamado em resposta ao pressionamento de qualquer tecla.**
  - `keyTyped` **é chamado em resposta ao pressionamento de tecla que não é uma** *tecla de ação*: *Home, Page Up*, **etc.**
  - `keyReleased` **é chamado quando a tecla é liberada, depois do evento** `keyPressed` **ou** `keyTyped`**.**

CÓDIGOS VIRTUAIS DE TECLAS

- **Método** `getKeyCode()` **informa o código virtual da tecla**
- **Método** `getKeyChar()` **informa o caractere digitado**
- **Códigos virtuais, definidos pela classe** `KeyEVent`**, têm prefixo** `VK_`**. Por exemplo,** `VK_A`**,** `VK_B`**,** `VK_SHIFT`**, ect**

EXEMPLO DE EVENTOS DE TECLADO

- **Ao digitar uma letra** `"A"`**, pressionando SHIFT e a tecla A, ocorrem os seguintes eventos e ações:**
  1. **Pressionada a tecla SHIFT: chamado** `keyPressed` **com** `VK_SHIFT`
  2. **Pressionada a tecla A: chamado** `keyPressed` **com** `VK_A`
  3. **Digitada** `"A"`**: chamado** `keyTyped` **com** `"A"`
  4. **Liberada a tecla A: chamado** `keyReleased` **com** `VK_A`
  5. **Liberada a tecla SHIFT: chamado** `keyReleased` **com** `VK_SHIFT`
- **Ao digitar uma letra** `"a"`**, pressionando a tecla A, ocorrem os seguintes eventos e ações:**
  1. **Pressionada a tecla A: chamado** `keyPressed` **com** `VK_A`
  2. **Digitada** `"a"`**: chamado** `keyTyped` **com** `"a"`
  3. **Liberada a tecla A: chamado** `keyReleased` **com** `VK_A`

EVENTOS E OUVINTES V

Interface	Método	Parâmetro/ Acessadores	Origem
KeyListener	keyPressed keyReleased keyTyped	KeyEvent getKeyChar getKeyCode getKeyModifiersText getKeyText isActionKey	Component

MÉTODOS DE KeyEvent

- `char getKeyChar()`:
  devolve o caractere digitado
- `int getKeyCode()`:
  devolve o código virtual do caractere digitado
- `String getKeyText(int codTecla)`:
  devolve texto descrevendo o código virtual da tecla.
  Ex: `getKeyText(KeyEvent.VK_END)` **devolve** `"END"`
- `int getKeyModifiers( )`:
  devolve a máscara do modificador associado ao evento
- `String getKeyModifiersText(int modificadores)`:
  devolve descrição das tecla modificadoras (SHIFT, CTRL, CTRL+SHIFT)
  Parâmetro obtido pelo método `getModifiers()`

MÉTODOS DE KeyEvent ...

- `boolean isAltDown()`:
  idem para tecla ALT
- `boolean isControlDown()`:
  idem para tecla CONTROL
- `boolean isShiftDown()`:
  idem para tecla SHIFT

DEMONSTRANDO EVENTOS DE TECLADO

```
>java UsaTecladoDemo
```

DEMONSTRANDO EVENTOS DE TECLADO ...

```
import javax.swing.*; import java.awt.*;
import java.awt.event.*;
public class UsaTecladoDemo extends JFrame
                 implements KeyListener {
  private String line1 = "", line2 = "", line3 = "";
  private JTextArea textArea = new JTextArea (10, 15);
  public TecladoDemo () {...}
  public void keyPressed (KeyEvent e) {...}
  public void keyReleased (KeyEvent e) {...}
  public void keyTyped (KeyEvent e) {...}
  private void ativaLinhas2e3 (KeyEvent e) {...}
  public static void main (String args[]){...}
}
```

DEMONSTRANDO EVENTOS DE TECLADO ...

```
public UsaTecladoDemo () {
     super ("Demonstrando Eventos de Teclado");
     textArea.setText ("Aperte alguma tecla no teclado");
     textArea.setEnabled (false);
     // permite que o frame processe os eventos de teclado
     addKeyListener (this);
     getContentPane().add(textArea);
     setSize (350, 100);
     setVisible(true);
}
```


DEMONSTRANDO EVENTOS DE TECLADO ...

```
public void keyPressed (KeyEvent e) {
   line1 = "Tecla pressionada: " + e.getKeyText(e.getKeyCode());
   ativaLinhas2e3(e);
}
public void keyReleased (KeyEvent e) {
   line1 = "Tecla liberada: " + e.getKeyText(e.getKeyCode());
   ativaLinhas2e3(e);
}
public void keyTyped (KeyEvent e) {
  line1 = "Tecla pressionada: " + e.getKeyChar();
  ativaLinhas2e3(e);
}
```


DEMONSTRANDO EVENTOS DE TECLADO ...

```
private void ativaLinhas2e3 (KeyEvent e) {
  line2 = "Esta tecla " + (e.isActionKey() ? "": "não ") +
                                    "é uma tecla de ação.";

  String temp = e.getKeyModifiersText (e.getModifiers());
  line3 = "MOdificadores pressionados: " +
               (temp.equals("") ? "nenhum" : temp);

  textArea.setText(line1 + "\n" + line2 + "\n" + line3 + "\n");
}
```


DEMONSTRANDO EVENTOS DE TECLADO ...

```
public static void main (String args[]){
      UsaTecladoDemo j = new UsaTecladoDemo();
      j.addWindowListener (
         new WindowAdapter() {
             public void windowClosing (WindowEvent e) {
                 System.exit(0);
             }
         }
      );
}
```

MULTIDIFUSÃO DE EVENTOS

- As origens de eventos do AWT suportam um modelo de multidifusão para os ouvintes registrados
- Um mesmo evento pode ser enviado para mais de um ouvinte
- Multidifusão é útil quando em evento desperta o interesse em muitas partes do programa

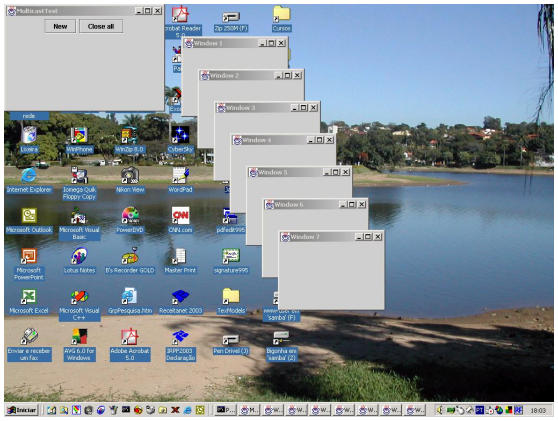
MULTIDIFUSÃO DE EVENTOS ...

- Uma janela com dois botões de comando:
  - Botão `New`: comanda a criação de uma nova janela
  - Botão `Close All`: fecha todas as janelas

```
>java MulticastTest
```

MULTIDIFUSÃO DE EVENTOS ...

```
>java MulticastTest
```



MULTIDIFUSÃO DE EVENTOS ...

```
import java.awt.*; import java.awt.event.*;
import javax.swing.*;
class MulticastPanel extends JPanel implements ActionListener {
   private static int counter=0;private JButton closeAllButton;
   public static void setCounter(int c){counter = c;}
   public MulticastPanel( ){
      JButton newButton = new JButton("New");
      add(newButton);
      newButton.addActionListener(this);
      closeAllButton = new JButton("Close all");
      add(closeAllButton);
   }
  public void actionPerformed(ActionEvent evt){ ... }
}
```

MULTIDIFUSÃO DE EVENTOS ...

```
public void actionPerformed(ActionEvent evt){
      SimpleFrame f = new SimpleFrame();
      f.setTitle("Window " + counter);
      f.setSize(200, 150);
      f.setLocation(30 * counter, 30 * counter);
      f.show();
      closeAllButton.addActionListener(f);
}
class SimpleFrame extends JFrame implements ActionListener {
   public void actionPerformed(ActionEvent evt){
      MulticastPanel.setCounter(0);
      dispose();
   }
}
```


MULTIDIFUSÃO DE EVENTOS ...

```
class MulticastFrame extends JFrame{
 public MulticastFrame() {
   setTitle("MulticastTest"); setSize(300, 200);
   addWindowListener(new WindowAdapter( ){
    public void windowClosing(WindowEvent e){System.exit(0);}});
   Container contentPane = getContentPane();
   contentPane.add(new MulticastPanel());
 }
}
public class UsaMulticast {
 public static void main(String[] args) {
    JFrame frame = new MulticastFrame(); frame.show();
 }
}
```


COMPONENTES AVANÇADOS DE INTERFACE

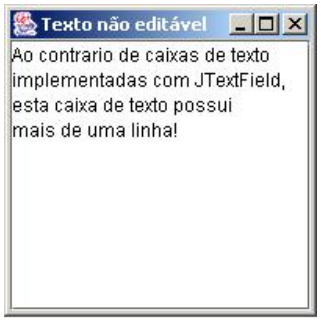
ÁREAS DE TEXTO E BARRA DE ROLAGEM

COMPONENTE JTextArea

- JTextArea fornece uma área para manipulação de múltiplas linhas de texto.
- JTextArea, assim como JTextField, herda de JTextComponent.
- Um componente JTextArea pode ser ou não editável pelo usuário.
- É possível fornecer barras de rolagem a uma JTextArea associando-a com um objeto JScrollPane.
- Métodos de JTextArea:
  - setEditable(boolean b)
  - JTextArea(String t, int largura, int altura)

JTextArea COM BARRA DE ROLAGEM

```
>java UsaTextAreaDemo
```



JTextArea COM BARRA DE ROLAGEM

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

public class TextAreaDemo extends JFrame {
    private JTextArea tArea;
    public TextAreaDemo() {....}
}
```


JTextArea COM BARRA DE ROLAGEM ...

```
public TextAreaDemo() {
    super ("Texto não editável");
    Container c = getContentPane();
    String s = "Ao contrario de caixas de texto\n" +
               "implementadas com JTextField,\n" +
               "esta caixa de texto possui\n" +
               "mais de uma linha!\n";
    tArea = new JTextArea (s, 10, 15);
    tArea.setEditable (false);
    c.add (new JScrollPane(tArea));
    setSize(200, 200);
    setVisible(true);
}
```

JTextArea COM BARRA DE ROLAGEM ...

```
import java.awt.*; import java.awt.event.*;
import javax.swing.*;
public class UsaTextAreaDemo {
  public static void main (String args[]) {
    TextAreaDemo j = new TextAreaDemo();
    j.addWindowListener (
       new WindowAdapter () {
           public void windowClosing (WindowEvent e) {
                  System.exit(0);
              }
          }
      );
  }
}
```


JANELA COM BARRAS DE ROLAGEM

```
>java UsaJanelaComScrollBar
```

JANELA COM BARRAS DE ROLAGEM...

```
public class UsaJanelaComScrollBar {
   public static void main(String [ ] args) {
      JFrame janela = new JanelaComScrollBar( );
      janela.addWindowListener(
         new WindowAdapter( ){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
         }
      );
   }
}
```


JANELA COM BARRAS DE ROLAGEM...

```
import javax.swing.*; import java.awt.*;
import java.awt.event.*;
class JanelaComScrollBar extends JFrame {
   public JanelaComScrollBar( ) {
      setTitle("Janela com Barra de Rolagem"); setSize(300,200);
      Container c = getContentPane();
      JPanel j = new JPanel( );
      c.add(new JScrollPane(j));
      String s ="aaaaaaaaaa\nbbbbbbbbbb\ncccccccccc";
      JTextArea t = new JTextArea(s,20,30);
      j.add(t);
      setVisible(true);
  }
}
```

BARRAS DESLIZANTES

COMPONENTE JSlider

- **O componente** JSlider **também chamado de** *Controle Deslizante* **permite ao usuário selecionar a partir de um intervalo de valores inteiros.**
- JSlider **são como caixas de rolagens graduadas com marcas de medida.**
- **As marcas de escala de um** JSlider **são altamente personalizáveis.**
- JSlider **suportam tanto eventos de mouse como de teclado.**
- JSlider **de orientação horizontal têm seu valor mínimo no lado esquerdo.**

MÉTODOS DE JSlider

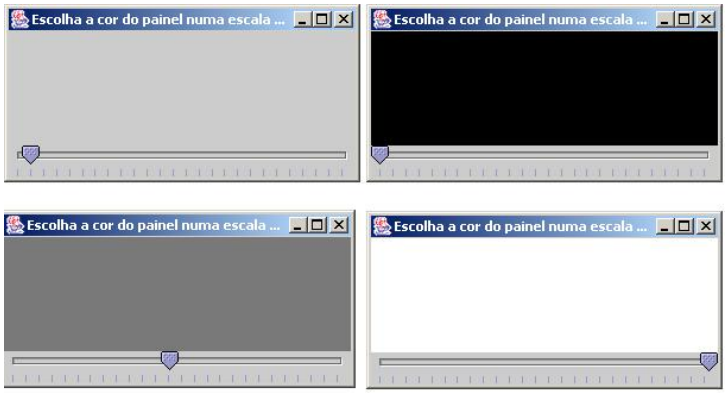
- `JSlider(int orient, int min, int max, int init)`:
  um dos construtores da classe: *orientação, valor mínimo, valor máximo* **e** *valor inicial* **do componente.**
- `setMajorTickSpacing(int)`:
  determina o intervalo entre duas marcas visíveis do componente JSlider**.**
- `setPaintTicks (boolean)`:
  determina se as marcas de graduação estarão visíveis ou não.
- `getValue()`:
  retorna o valor corrente do componente JSlider**.**

EVENTOS E OUVINTES VI

Interface	Método	Parâmetro/ Acessadores	Origem
`ChangeListener`	`stateChanged`	`ChangeEvent` `getSource`	`JSlider`

DEMONSTRADOR DE JSlider

```
>java UsaSliderDemo
```



UM JPanel DE COR CONFIGURÁVEL

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

public class SliderDemo extends JFrame {
  private JSlider tomDeCinza =
     new JSlider (SwingConstants.HORIZONTAL, 0, 255, 10);
  private JPanel pC = new JPanel(); // painel cinza
  public SliderDemo() { ... }
}
```


UM JPanel DE COR CONFIGURÁVEL ...

```
public SliderDemo() {
    super ("Escolha a cor do painel numa escala de cinza");
    final Container c = getContentPane();
    c.add (pC, BorderLayout.CENTER);
    tomDeCinza.setMajorTickSpacing(10);
    tomDeCinza.setPaintTicks(true);
    tomDeCinza.addChangeListener (
       new ChangeListener() {
          public void stateChanged (ChangeEvent e) {
             int cor = tomDeCinza.getValue();
             pC.setBackground(new Color (cor, cor, cor));
       }} );
    c.add(tomDeCinza); setSize(300,150); setVisible(true);
}
```


USANDO O DEMONSTRADOR DE JSlider

```
import java.awt.*; import java.awt.event.*;
import javax.swing.*;
public class UsaSliderDemo {
  public static void main (String args[]){
     SliderDemo j = new SliderDemo();
     j.addWindowListener (
        new WindowAdapter () {
            public void windowClosing(WindowEvent e){
                   System.exit(0);
            }
        }
    );
  }
}
```

MENUS

MENU DE OPÇÕES

- *Menus* **são um meio de organizar as funcionalidades de um aplicativo de maneira prática e natural.**
- **Em GUI's Swing, apenas instâncias das classes que fornecem o método** setJMenuBar **podem usar menus.**
- **As classes utilizadas para definir menus são:**
  - JMenuBar
  - JMenuItem
  - JCheckBoxMenuItem
  - JRadioButtonMenuItem
  - JMenu

CLASSES DE MENU

- **A classe** JMenuBar **contém métodos para gerenciar uma** *barra de menus*.
- **A classe** JMenuItem **contém métodos para gerenciar** *itens de menus*.
- **A classe** JMenu **contém os métodos para gerenciar** *menus*
- **A classe** JCheckBoxMenuItem **contém métodos para gerenciar itens de menu que podem ser ativados ou desativados.**
- **A classe** JRadioButtonMenuItem **contém métodos que gerenciam um conjunto de itens de menu de modo que em um dado momento apenas um item pode estar ativo.**

APLICAÇÃO COM BARRA DE MENUS, MENUS E SUBMENUS

BARRA DE MENUS, MENUS E SUBMENUS

```
import javax.swing.*;
import java.awt.event.*; import java.awt.*;
public class UsaMenu {
   public static void main( String args[] ) {
      MenuTest m = new MenuTest();
      m.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e ) {
               System.exit( 0 );
            }
         }
      );
   }
}
```


BARRA DE MENUS, MENUS E SUBMENUS...

```
>java UsaMenu
```

BARRA DE MENUS, MENUS E SUBMENUS...

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

public class MenuTest extends JFrame {
   private Color colorValues[] =
      { Color.black, Color.blue, Color.red, Color.green };
   private JRadioButtonMenuItem colorItems[], fonts[];
   private ButtonGroup fontGroup, colorGroup;
   private JCheckBoxMenuItem styleItems[];
   private JLabel display;
   private int style;
   public MenuTest( ) { ... }
}
```


CONSTRUTOR DE MenuTest

```
public MenuTest() {
   super( "Uso de Menus" );
   JMenuBar bar = new JMenuBar();  setJMenuBar( bar );

   "CONSTRÓI MENU fileMenu";   bar.add(fileMenu);
   "CONSTRÓI MENU formatMenu"; bar.add( formatMenu);

   Container c = getContentPane();
   c.setBackground( Color.cyan );
   "CONSTRÓI RÓTULO display";
   c.add( display, BorderLayout.CENTER );

   setSize( 500, 200 );  show();
}
```

CONSTRÓI RÓTULO display

```
display = new JLabel("Uso de Menus",SwingConstants.CENTER);

display.setForeground( colorValues[ 0 ] );

display.setFont(new Font( "TimesRoman", Font.PLAIN, 72 ) );
```


CONSTRÓI MENU fileMenu

```
JMenu fileMenu = new JMenu( "Arquivo" );
fileMenu.setMnemonic( 'A' );
file Menu.setToolTipText("Decida-se logo!");

"CONSTRÓI ITEM O Que é";
fileMenu.add( aboutItem );

"CONSTRÓI ITEM Sair";
fileMenu.add(exitItem );
```


CONSTRÓI ITEM O Que é

```
JMenuItem aboutItem = new JMenuItem( "O Que é..." );
aboutItem.setMnemonic( 'O' );

aboutItem.addActionListener(
  new ActionListener() {
    public void actionPerformed( ActionEvent e ) {
      JOptionPane.showMessageDialog( MenuTest.this,
       "Este é um exemplo de uso de Barra de Menus,\n" +
       "Menus e SubMenus","O Que é",JOptionPane.PLAIN_MESSAGE);
    }
  }
);
```


CONSTRÓI ITEM Sair

```
JMenuItem exitItem = new JMenuItem( "Sair" );
exitItem.setMnemonic( 'S' );

exitItem.addActionListener(
   new ActionListener() {
       public void actionPerformed(ActionEvent e){
           System.exit( 0 );
       }
   }
);
```

CONSTRÓI MENU formatMenu

```
JMenu formatMenu = new JMenu( "Formatação" );
formatMenu.setMnemonic( 'r' );
formatMenu.setToolTipText("Formata o rótulo");

"CONSTRÓI SUBMENU colorMenu";
formatMenu.add( colorMenu );

formatMenu.addSeparator();

"CONSTRÓI SUBMENU fontMenu";
formatMenu.add( fontMenu );
```


CONSTRÓI SUBMENU colorMenu

```
String colors[] ={ "Preto", "Azul", "Vermelho", "Verde" };
JMenu colorMenu = new JMenu( "Cor do Texto" );
colorMenu.setMnemonic( 'C' );
colorItems = new JRadioButtonMenuItem[ colors.length ];
colorGroup = new ButtonGroup();
ItemHandler itemHandler = new ItemHandler();
for (int i = 0; i < colors.length; i++) {
     colorItems[ i ] = new JRadioButtonMenuItem(colors[i]);
     colorMenu.add( colorItems[ i ] );
     colorGroup.add( colorItems[ i ] );
     colorItems[ i ].addActionListener( itemHandler );
}
colorItems[ 0 ].setSelected( true );
```


CONSTRÓI SUBMENU fontMenu

```
JMenu fontMenu = new JMenu( "Fonte" );
fontMenu.setMnemonic( 'n' );

"ADICIONA ITENS DE FONTES AO fontMenu";

fontMenu.addSeparator( );

"ADICIONA ITENS DE ESTILOS AO fontMenu";
```


ADICIONA ITENS DE FONTES AO fontMenu

```
String fontNames[] = { "TimesRoman", "Courier", "Helvetica" };
fonts = new JRadioButtonMenuItem[ fontNames.length ];
fontGroup = new ButtonGroup();

for ( int i = 0; i < fonts.length; i++ ) {
    fonts[ i ] = new JRadioButtonMenuItem( fontNames[ i ] );
    fontMenu.add( fonts[ i ] );
    fontGroup.add( fonts[ i ] );
    fonts[ i ].addActionListener( itemHandler );
}
fonts[ 0 ].setSelected( true );
```

ADICIONA ITENS DE ESTILOS AO fontMenu

```
String styleNames[] = { "Negrito", "Itálico" };
styleItems = new JCheckBoxMenuItem[ styleNames.length ];
StyleHandler styleHandler = new StyleHandler();

for ( int i = 0; i < styleNames.length; i++ ) {
    styleItems[i] = new JCheckBoxMenuItem(styleNames[i]);
    fontMenu.add( styleItems[i]);
    styleItems[i].addItemListener(styleHandler);
}
```


TRATADOR DE SELEÇÃO DE ITEM DE COR

```
class ItemHandler implements ActionListener {
   public void actionPerformed( ActionEvent e ) {
      for ( int i = 0; i < colorItems.length; i++ )
          if ( colorItems[ i ].isSelected() ) {
              display.setForeground(colorValues[i]); break;
          }
      for ( int i = 0; i < fonts.length; i++ )
          if (e.getSource( ) == fonts[i]) {
              display.setFont( new Font(
                fonts[i].getText(),style,72)); break;
          }
      repaint();
   }
}
```


TRATADOR DE SELEÇÃO DE ITEM DE ESTILO

```
class StyleHandler implements ItemListener {
   public void itemStateChanged( ItemEvent e ) {
      style = 0;
      if ( styleItems[ 0 ].isSelected() )
           style += Font.BOLD;
      if ( styleItems[ 1 ].isSelected() )
           style += Font.ITALIC;
      display.setFont( new Font(
           display.getFont().getName(), style, 72 ) );
      repaint();
   }
}
```


MENUS SENSÍVEIS AO CONTEXTO

MENUS SENSÍVEIS AO CONTEXTO - JPopupMenus

- **Um** *Menu sensível ao contexto* **ou** *pop-up* **revela opções específicas para o componente gráfico para o qual foi ativado.**
- **Menus** *pop-up* **são ativados por eventos denominados** *eventos de gatilho pop-up*.
- **O disparador de PopUpMenu depende da plataforma**
- **Por exemplo, clique com botão direito sobre um componente é o gatilho pop-up no Windows**
- **Método** `isPopupTrigger( )` **da classe** `MouseEvent` **informa se evento foi causado por ativação de menu** `pop-up`
- **Método** `mpu.show(componente, x, y)` **faz a exibição nas coordenadas** `(x,y)` **do menu** `pop-up mpu`

UTILIZANDO JPopupMenus

```
class UsaPopupMenu {
  private JPopupMenu menu = new JPopupMenu( );

  "construir cada item de menu e adicioná-lo ao menu pop-up;
   determinar um tratador de evento para cada item do menu";

  "definir MouseListener para o componente que exibe um
   JpopupMenu quando o evento de gatilho pop-up ocorre";

  addMouseListener (...);
  ...
}
```


UTILIZANDO JPopupMenus...

```
addMouseListener (
    new MouseAdapter() {
        public void mousePressed (MouseEvent e) {
          checkForTriggerEvent (e);
        }
        public void mouseReleased (MouseEvent e){
           checkForTriggerEvent (e);
        }
        private void checkForTriggerEvent (MouseEvent e) {
            if (e.isPopupTrigger( ) )
               menu.show(e.getComponent(),e.getX(),e.getY() );
        }
    }
);
```


JANELAS MÚLTIPLAS

JDesktopPane e JInternalFrame

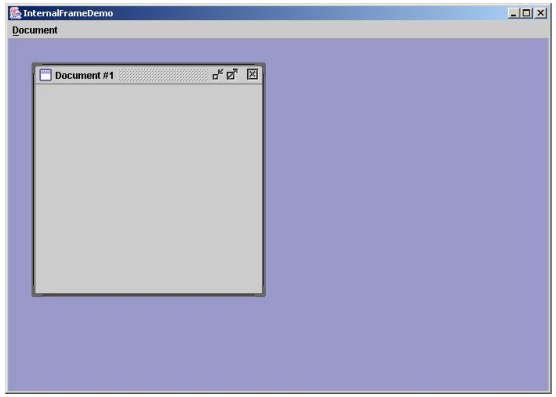
- **Uma** Interface de Múltiplos Documentos (`multiple document interface - MDI`) **consiste em uma janela principal contendo outras janelas.**
- **A janela principal é frequentemente chamada** *Janela Mãe*.
- **As janelas internas são chamadas** *Janelas Filhas*.
- **Esta interface permite gerenciar vários documentos que estão sendo processados em paralelo.**
- **As classes** JDesktopPane **e** JInternalFrame **do Swing fornecem suporte para criar interfaces de múltiplos documentos.**

GERÊNCIA DE MÚLTIPLOS DOCUMENTOS ...

```
public class UsaInternalFrame {
  public static void main(String[] args) {
     ExternalFrame frame = new ExternalFrame( );
     frame.setVisible(true);
  }
}
```


GERÊNCIA DE MÚLTIPLOS DOCUMENTOS ...

```
>javaUsa UsaInternalFrame
```



GERÊNCIA DE MÚLTIPLOS DOCUMENTOS

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

public class InternalFrame extends JInternalFrame {
    static int openFrameCount = 0;
    static final int xOffset = 30, yOffset = 30;

    public InternalFrame() {...}

}
```

GERÊNCIA DE MÚLTIPLOS DOCUMENTOS ...

```
public InternalFrame() {
   super("Document #" + (++openFrameCount),
          true, //a frame pode ter seu tamanho modificado
          true, //a frame pode ser fechada.
          true, //a frame pode ser maximizada.
          true);//a frame pode ser minimizada.
   // Define o tamanho inicial da frame.
   setSize(300,300);
   // Define a localizacao inicial da nova frame.
   setLocation(xOffset*openFrameCount,
               yOffset*openFrameCount);
}
```


GERÊNCIA DE MÚLTIPLOS DOCUMENTOS ...

```
import javax.swing.JInternalFrame;
import javax.swing.JDesktopPane;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JFrame;
import java.awt.event.*; import java.awt.*;
public class ExtenalFrame extends JFrame {
      JDesktopPane desktop;
      public ExternalFrame {...}
      protected JMenuBar createMenuBar( ) {...}
      protected void createFrame( ) {...}
}
```


GERÊNCIA DE MÚLTIPLOS DOCUMENTOS ...

```
public ExternalFrame() {
   super("InternalFrameDemo");
   // Make the big window be indented 50 pixels
   // from each edge of the screen.
   int inset = 50;
   Dimension screenSize =
         Toolkit.getDefaultToolkit().getScreenSize();
   setBounds(inset, inset, screenSize.width - inset*2,
             screenSize.height-inset*2);
   "Adiciona ouvinte para fechar janela";
   "Set up the GUI";
}
```


ADICIONA OUVINTE PARA FECHAR JANELA

```
addWindowListener(
       new WindowAdapter() {
          public void windowClosing(WindowEvent e) {
             System.exit(0);
          }
       }
)
```

SET UP THE GUI

```
desktop = new JDesktopPane();
createFrame(); //Create first window
setContentPane(desktop);
setJMenuBar(createMenuBar());
//Make dragging faster:
desktop.putClientProperty(
              "JDesktopPane.dragMode", "outline");
```


GERÊNCIA DE MÚLTIPLOS DOCUMENTOS ...

```
protected JMenuBar createMenuBar() {
   JMenuBar menuBar = new JMenuBar();
   JMenu menu = new JMenu("Document");
   menu.setMnemonic(KeyEvent.VK_D);
   JMenuItem menuItem = new JMenuItem("New");
   menuItem.setMnemonic(KeyEvent.VK_N);
   menuItem.addActionListener(new ActionListener(){...});
   menu.add(menuItem);
   menuBar.add(menu);
   return menuBar;
}
```


GERÊNCIA DE MÚLTIPLOS DOCUMENTOS ...

```
menuItem.addActionListener(
     new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                createFrame();
            }
     }
);
```


GERÊNCIA DE MÚLTIPLOS DOCUMENTOS ...

```
protected void createFrame() {
   InternalFrame frame = new InternalFrame();
   frame.setVisible(true); //necessary as of kestrel
   desktop.add(frame);
   try {
           frame.setSelected(true);
   } catch (java.beans.PropertyVetoException e) {}
}
```

APARÊNCIA E COMPORTAMENTO

APARÊNCIA E COMPORTAMENTO

- **Programas Java que utilizam apenas classes definidas no pacote AWT têm a aparência e o comportamento da plataforma em que o programa executa.**
- **Os componentes GUI em cada plataforma têm aparências diferentes que podem requer quantidades diferentes de espaço para serem exibidas. Isso pode alterar o leiaute e o alinhamento dos componentes GUI.**
- **Os componentes GUI em cada plataforma têm funcionalidades padrão diferentes.**
- **Há componentes GUI do Swing que eliminam estas questões, fornecendo funcionalidade uniforme entre plataformas.**

APARÊNCIA E COMPORTAMENTO ...

- **O conjunto de propriedades como cor e textura que determinam o aspecto geral de qualquer componente em uma certa plataforma é denominado** *look and feel*.
- **Uma mesma plataforma pode possuir diferentes conjuntos** *look and feel* **instalados, mas em um dado momento apenas um estará sendo usado.**
- **A classe** UIManager **gerencia os conjuntos** *look and feel* **de uma plataforma.**
- **Objetos da classe** UIManager.LookAndFeelInfo **têm informações sobre os** *look and feel* **instalados no sistema.**

Look and Feel - MÉTODOS RELACIONADOS

- `LookAndFeelInfo[ ] UIManager.getInstaledLookAndFeels()`: **retorna a lista de** *look and feel* **instalados no sistema.**
- `UIManager.setLookAndFeel(String lfClassname)`: **faz o** *look and feel* **de nome** `lfClassName` **ser o corrente no sistema.**

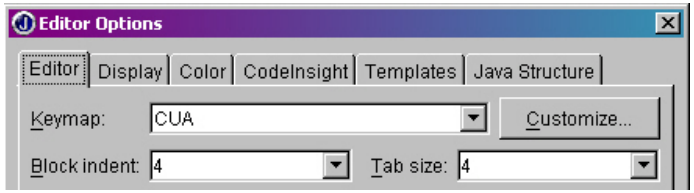
MUDANDO A APARÊNCIA DE UM APLICATIVO

- **A aparência do aplicativo pode ser mudada via chamadas a função mudaVisual. Ex.: mudaVisual(2), etc.**

```
...
UIManager.LookAndFeelInfo looks[] =
                    UIManager.getInstalledLookAndFeels();
...
private void mudaVisual (int value) {
   ...
   UIManager.setLookAndFeel(looks[value].getClassName());
   ...
}
```

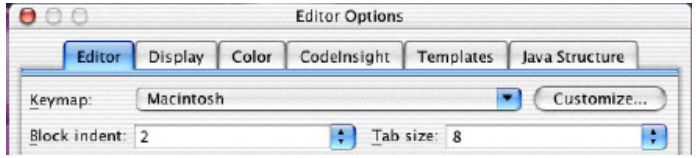

LOOK AND FEEL WINDOWS

```
UIManager.setLookAndFeel(
     "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
```



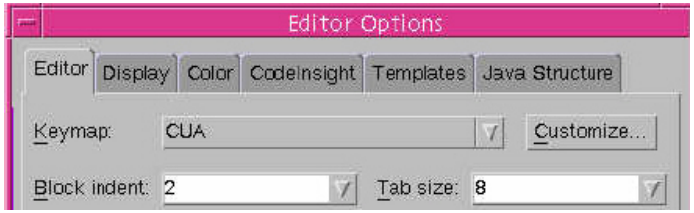
LOOK AND FEEL MAC

```
UIManager.setLookAndFeel(
           "");
```



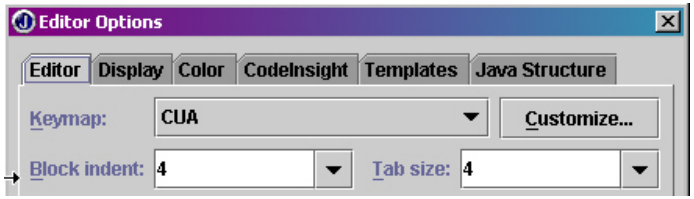
LOOK AND FEEL MOTIF

```
UIManager.setLookAndFeel(
       "com.sun.java.swing.plaf.motif.MotifLookAndFeel");
```



LOOK AND FEEL METAL

```
UIManager.setLookAndFeel(
            "javax.swing.plaf.metal.MetalLookAndFeel");
```



FIM

AMBIENTES DE PROGRAMAÇÃO
THREADS

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

24 de março de 2015

SUMÁRIO

- Criação de Threads
- Classe Thread
- Ciclo de Vida de Threads
- Prioridades de Threads
- Sincronização de Threads
- Threads Sincronizadas
- Encerramento de Threads

CRIAÇÃO DE THREADS

LINHAS CONTROLE DE EXECUÇÃO

- **Linhas de controle (`threads`) permitem que o programa execute mais de uma tarefa de cada vez.**
- **No caso de haver apenas um processador disponível a execução das linhas é entremeada.**
- **No caso de mais de um processador, pode haver paralelismo real entre as linhas.**
- **O número de processadores disponíveis não deve ser considerado para a correção de um programa multilinha (Multithreading).**
- **Em Java, linhas de controle são representados por meio de objetos.**

CRIAÇÃO DE LINHAS: MÉTODO I

- **Estender `java.lang.Thread`, sobrepondo `run()`**
- **O método `run()` de `Thread` nada faz**

```
class T extends Thread {
   public void run( ) { ...}
   ...
}
```


CRIAÇÃO DE LINHAS: MÉTODO I ...

- **Criar objetos da linha e iniciar sua execução com `start()`**

```
public class B {
  public static void main(String[] a) {
    T linha1 = new T();  // cria linha
    T linha2 = new T();  // cria outra linha
    linha1.start();  // dispara linha1
    ...
    linha2.start(); // dispara linha 2
    ....
  }
}
class T extends Thread {
   public void run( ) { ...}  ...
}
```


EXEMPLO: DOIS PROCESSOS PARALELOS

```
class PingPong extends Thread {
  String palavra; int tempo;
  PingPong(String texto, int espera) {
      palavra = texto;  tempo = espera;
  }
  public void run() {
    try {while (true) {
              System.out.print(palavra + " ");
              sleep(tempo);
        }
    }catch (InterruptedException e) { return; }
  }
}
```

EXEMPLO: DOIS PROCESSOS PARALELOS ...

```
class Jogo {

  public static void main(String[] args) {
    Thread t1 = new PingPong("Ping", 33);
    Thread t2 = new PingPong("PONG",100);
    t1.start(); t2.start();
  }

}
```


CRIAÇÃO DE LINHAS: MÉTODO II

- **A partir da Interface** `java.lang.Runnable`

```
public interface Runnable {
  public abstract void run();
}
```

- **Implementar interface** `java.lang.Runable`**, definindo** `run()`**:**

```
class R extends X implements Runnable {
  public void run() { ... }
      ...
}
```

- **Quando um objeto que implementa** `Runnable` **é usado na criação de uma** `thread`**, o disparo da** `thread` **leva a execução de** `run()`**.**

CRIAÇÃO DE LINHAS II ...

- **Assim, declara-se uma classe que implementa** `Runnable`**:**

```
class R extends X implements Runnable {
  public void run() { ... }
      ...
}
```


CRIAÇÃO DE LINHAS II ...

- **Criam-se objetos da linha e inicia sua execução com o método** `start()` **de** `Thread`**:**

```
class B {
  public static void main(String[] a) {
    Thread linha = new Thread(new R()); // cria linha
    linha.start(); ... // inicia linha
    new Thread(new R()).start(); cria e inicia outra linha
}
class R extends X implements Runnable {
  public void run() { ... }
      ...
}
```

- **Uma linha termina quando seu** `run` **termina.**

(RE)DEFINIÇÃO DE `run()`

```
class Exemplo {
   public static void main(String[] a) {
      ExtnOfThread t1 = new ExtnOfThread();
      t1.start();
      Thread t2 = new Thread (new ImplOfRunnable());
      t2.start();
   }
}
```


(RE)DEFINIÇÃO DE `run()`...

```
class ExtnOfThread extends Thread {
   public void run() {
     System.out.println("extension of Thread running");
     setPriority( getPriority() +1 );
     try {sleep(1000);}
     catch (InterruptedException ie) return;
   }
}

class ImplOfRunnable implements Runnable {
   public void run() { ... }
}
```


DEFINIÇÃO DE `run()` DE `Runnable`

- **Somente pode-se chamar métodos de `Thread` se existir um objeto do tipo `Thread` disponível:**

```
class ImplOfRunnable implements Runnable {
   public void run() {
     System.out.println("Implementation of Runnable running");
 ??? setPriority( getPriority() +1 );
 ??? try {sleep(1000);}
     catch (InterruptedException ie) return;
   }
}
```


DEFINIÇÃO DE `run()` DE `Runnable`...

- **A solução é recuperar objeto da `Thread` em execução:**

```
class ImplOfRunnable implements Runnable {
   public void run() {
     Thread t = Thread.currentThread();
     System.out.println("Implementation of Runnable running");
     t.setPriority(t.getPriority() +1 );
     try {t.sleep(1000);}
     catch (InterruptedException ie) return;
   }
}
```

CLASSE THREAD

A CLASSE `java.lang.Thread`

A classe possui os seguintes métodos:

- `Thread()`:
  Constrói uma nova linha, cujo fluxo de execução é dado pelo método `run`
- `Thread(Runnable r)`:
  Constrói uma nova linha, cujo fluxo de execução é dado pelo `run` **do objeto r.**
- `public void start()`:
  Inicia a execução da linha e dispara o método `run()`
- `public void run()`:
  Função principal que define o corpo da linha. Não deve ser chamada diretamente pelo usuário. Nada faz. Existe para ser redefinida por subclasses.

A CLASSE `java.lang.Thread`...

- `public final void stop()`:
  Encerra a execução da linha (depreciado).
- `public static void yield()`:
  Faz a linha ceder a vez. Se houver outras linhas executáveis, elas serão escaladas em seguida.
  O escalador escolhe a linha executável de maior prioridade, possívelmente a linha que cedeu a vez.
- `public static void sleep(long ms) throws InterruptException`:
  coloca a linha atualmente em execução para dormir por `ms` **milisegundos, e permite que outras linhas tenham a chance de ser executadas.**

A CLASSE `java.lang.Thread`...

- `public void interrupt()`:
  - **causa o levantamento da exceção** `InterruptException` **na linha do objeto receptor, o qual deve estar no estado dormente ou de espera.**
  - **Se uma linha** `t1` **executa** `t2.interrupt()`**, quem recebe a exceção levantada é** `t2` **e não** `t1`.
  - **linha** `t2` **acima somente recebe a interrupção se estiver bloqueada**
- `public static boolean interrupted()`:
  informa se a linha que executa a instrução recebeu algum pedido de interrupção e faz o status de interruped igual a false.

A CLASSE `java.lang.Thread`...

- `public final void join() throws InterruptedException`:
  Espera até que a linha (this) encerre sua execução.

- `public final void join(long millis)`
  `throws InterruptedException`:
  Espera até que a linha (this) encerre sua execução no máximo o tempo `millis` **(milissegundos) especificado.**
  O tempo `0` **significa espera indefinida.**

A CLASSE `java.lang.Thread`...

- `public final`
  `void join(long millis, int nanos)`
  `throws InterruptedException`:
  Espera até que a linha (this) encerre sua execução no máximo o tempo de `millis*1000 + nanos` **(nanossegundos) especificado.**
  O tempo `0`**(zero) nanossegundos significa espera indefinida.**

A CLASSE `java.lang.Thread`...

- `public final void suspend()`:
  Suspende a execução da linha (depreciado por perigo de deadlock).

- `public final void resume()`:
  Retoma a execução da linha. Somente é válido depois que `suspend()` **tiver sido chamado (depreciado).**

- `public final boolean isAlive()`:
  Retorna verdadeiro se a linha tiver sido iniciada e ainda não parou.
  Retorna falso se a linha ainda for nova e não for executável, ou se tiver sido encerrada.

A CLASSE `java.lang.Thread`...

- `public final String getName()`:
  Retorna o string passado ao construtor de `Thread`.

- `public final setName(String name)`
  `throws securityException`

A CLASSE `java.lang.Thread`...

- `public final static int MIN_PRIORITY`:
  Prioridade mínima (1) que uma linha pode ter.
- `public final static int NORM_PRIORITY`:
  Prioridade default de uma linha (5).
- `public final static int MAX_PRIORITY`:
  Prioridade máxima (10) de uma linha
- `public static Thread currentThread()`:
  retorna referência da thread que executou o método;

A CLASSE `java.lang.Thread`...

- `public final void setPriority(int p) throws IllegalArgumentException`:
  Define prioridade da linha
  (`MIN_PRIORITY <= p <= MAX_PRIORITY`)
- `public final int getPriority()`:
  Informa a prioridade da linha.
- **Há outros métodos.**

CICLO DE VIDA DE THREADS

CICLO DE VIDA DE UMA LINHA

- **Linha x do tipo T é criada com a criação do objeto**
  ```
  T x = new T();
  ```
- **Linha x iniciada com a chamada**
  ```
  x.start();
  ```
- **Execução inicia-se pelo método** `x.run()`
- **Linha x termina com o fim de** `x.run()` **ou com a chamada** `x.stop()` **– depreciado.**
- **Linha x pode ser interrompida temporariamente via** `x.suspend()` **– depreciado**
- **Linha x pode ser retomada via** `x.resume()` **– depreciado.**

CICLO DE VIDA DE UMA LINHA ...

- Linhas estão sujeitas à preempção por uma outra linha de maior prioridade
- Uma linha pode ceder o controle do processador via `this.yield()`
- Prioridades vão de 1 (baixa) a 10 (alta)
- Pode-se ajustar a prioridade da linha x, como em
  `x.setPriority(x.getPriority() + 1)`
- Cada linha possui uma pilha tamanho padrão de 400kb aproximadamente

ESTADOS DE UMA LINHA

- NOVO:
  Estado inicial após a criação da linha com o operador `new`.
- EXECUTÁVEL:
  Estado após a execução do método `start()`.
  - A linha pode estar rodando ou não.
  - Em alguns sistemas uma linha executa até que outra de maior prioridade tome-lhe o controle.
  - Em outros sistemas, usa-se a política de tempo compartilhado.

ESTADOS DE UMA LINHA ...

- BLOQUEADO:
  Uma linha entra no estado bloqueado quando é:
  - chamado o método `sleep()` da linha
  - chamado o método `suspend()` da linha (depreciado)
  - chamado o método `wait()` da linha
  - chamada uma operação de entrada/saída
- ENCERRADO:
  Uma linha é encerrada quando:
  - seu método `run()` terminou
  - o método `stop()` da linha foi executado (depreciado)

SAINDO DE ESTADO BLOQUEADO

- Quando uma linha bloqueada é reativada, o escalonador somente lhe dá o controle se ela tiver prioridade mais alta do que a atualmente em execução.
- A passagem estado BLOQUEADO para EXECUTÁVEL ocorre:
  - se a linha que dormia completou o tempo `ms` do `sleep(ms)`
  - se a linha suspensa teve seu método `resume()` chamado por uma outra.
  - se linha estava bloqueada devido a um `wait(..)`, e `notify` ou `notifyAll` foi chamado
  - se a operação de E/S que a linha requisitou terminou.

SAINDO DE ESTADO BLOQUEADO ...

- A ativação de uma linha somente é efetiva se combinar com a razão que a bloqueou. Por exemplo:
  - Se uma linha está bloqueada por I/O, uma chamada ao seu método `resume()` não muda seu estado.
- A ativação de um método incompatível com o estado da linha levanta a exceção `IllegalThreadStateException`.

PRIORIDADES DE THREADS

PRIORIDADES DE LINHAS

- Toda linha tem uma prioridade.
- Toda linha herda a prioridade de seu criador.
- Pode-se aumentar ou diminuir a prioridade de uma linha `x` com chamada a `x.setPriority(p)`.
- O escalador de linhas sempre escolhe a linha de maior prioridade.
- O sistema de execução Java pode suspender uma linha de maior prioridade para que outras tenham a chance, mas não se tem garantia de quando isto ocorrerá.

PRIORIDADES DE LINHAS ...

- A linha executável de maior prioridade continua sendo executada até que ela:
  - ceda a vez via `yield()`
  - deixe de ser executável
  - seja substituída por linha de maior prioridade que se torne executável (acordou ou teve I/O completado ou alguém chamou `resume` ou `notify`)
  - sua fatia de tempo venceu (em algumas implementações)
- Escalador usa alguma política que não faz parte da linguagem Java quando houver mais de uma linha executável com a mesma prioridade.

USO DE `yield`

- **Aplicação recebe uma lista de palavras e cria uma linha para cada palavra.**
- **O primeiro parâmetro da linha de comando indica se cada linha deve ou nao ceder a vez após o comando de impressão.**
- **O segundo parâmetro é o número de vezes a palavra será impressa.**
- **Demais parâmetros são a lista de palavras.**

POR EXEMPLO:

```
>java Babel false 3 Sim Nao
>Sim Sim Sim Nao Nao Nao
>java Babel true 3 Sim Nao
>Sim Nao Sim Nao Sim Nao
```


USO DE `yield`...

```
class Babel extends Thread {
   static boolean deveCeder;
   static int vezes;
   String palavra;
   Babel(String oQue) { palavra = oQue; }
   public void run() {
      for (int i = 0; i < vezes; i++) {
          System.out.print(palavra + " ");
          if (deveCeder) yield();
      }
   }
   public static void main(String[] args) { ... }
}
```


USO DE `yield`...

```
 public static void main(String[] args) {
     vezes = Integer.parseInt(args[1]);
     deveCeder = new Boolean(args[0]).booleanValue();

     Thread corrente = currentThread();
     corrente.setPriority(Thread.MAX_PRIORITY);
     for (int i=2; i<args.length; i++) {
          new Babel(args[i]).start();
     }
 }
>java Babel false 3 Sim Nao
>Sim Sim Sim Nao Nao Nao
>java Babel true 3 Sim Nao
>Sim Nao Sim Nao Sim Nao
```


SINCRONIZAÇÃO DE THREADS

LINHAS RELACIONADAS (S/ SINCRONISMO)

- **Determina se número `n` é primo.**
- **Valor `n` é o primeiro parâmetro da linha de comando.**
- **São criadas várias linhas do tipo `TestRange`, que procuram, em paralelo, divisores de `n` em centenas disjuntas.**
- **Cada linha lista o primeiro divisor encontrado.**

LINHAS RELACIONADAS (S/ SINCRONISMO)...

```
class TestRange extends Thread { ... }

public class TestPrime {
  public static void main(String s[]) {
    long n = Long.parseLong(s[0]);
    int centuries = (int)(n/100) + 1;

    for(int i=0; i<centuries; i++) {
       new TestRange(n,i*100).start();
       System.out.println("starting ...");
    }
  }
}
> java TestPrime 1000000000
```


LINHAS RELACIONADAS (S/ SINCRONISMO)..

```
class TestRange extends Thread {
   long n; // numero a testar
   long from, to; // limites do intervalo

   TestRange(long n, int argFrom) {
       this.n = n;
       if (argFrom==0) {
           from=2;
       } else from=argFrom;
       to=argFrom+99;
   }
   public void run() {...}
}
```


LINHAS RELACIONADAS (S/ SINCRONISMO)..

```
public void run() {
   for (long i=from; i<=to && i<n; i++) {
        if (n%i == 0) {
           System.out.println("Factor " + i +
                    " found by thread " + getName());
           break;
        }
        yield();
   }
}
```

LINHAS MUTUAMENTE EXCLUSIVAS

- Manômetro controlado por várias linhas
- Cada linha tem um método para aumentar a pressão
- Deve-se testar se a pressão é inferior a máximo permitido antes de incrementá-la
- Programa principal imprime o valor final da pressão quando todas as linhas houverem terminado

```
public class Pressure extends Thread { ... }

public class Controle { ... }
```


LINHAS MUTUAMENTE EXCLUSIVAS ...

```
public class Controle {
   public static int pressureGauge = 0;
   public static final int INCR = 30, Limit = 200;
   public static void main(String[]args) {
      Pressure []p = new Pressure[10];
      for (int i=0; i<10; i++) {
          p[i] = new Pressure(); p[i].start();
      }
      try{ for(int i=0; i<10; i++) p[i].join();}
      catch(Exception e){ }
      System.out.println("gauge reads " +
              pressureGauge + ", safe limit is " + Limit);
   }
}
```


CLASSE `Pressure`: PROPOSTA I(errada)

```
public class Pressure extends Thread {
  void RaisePressure() {
    if (Controle.pressureGauge < Controle.Limit-Controle.INCR){
        try {sleep(1000);} catch (Exception e){ };
        Controle.pressureGauge += Controle.INCR;
    }
  }

  public void run() {RaisePressure(); }
}
```

- A solução acima não funciona devido a condição de corrida
- O valor final de `Controle.pressureGauge` é imprevisível

CLASSE `Pressure`: PROPOSTA II (correta)

```
class Pressure extends Thread {
  static synchronized void RaisePressure() {
    if (Controle.pressureGauge<Controle.Limit-Controle.INCR){
        try {sleep(1000);} catch (Exception e){};
        Controle.pressureGauge += Controle.INCR;
    };
  }

  public void run() {RaisePressure(); }
}
```

- Somente uma linha de cada vez pode executar o método `RaisePressure`, porque ele é sincronizado.
- `sleep(t)` suspende a execução por `t` nanosegundos, mas não libera o bloqueio.

CLASSE Pressure: PROPOSTA III (correta)

```
class Pressure extends Thread {
  static Object sync = new Object();
  void RaisePressure() {
   synchronized(sync) {
     if (Controle.pressureGauge<Controle.Limit-Controle.INCR){
        try {sleep(1000);} catch (Exception e){};
        Controle.pressureGauge += Controle.INCR;
     };
   }
  }

  public void run() { RaisePressure(); }
}
```


COMANDO synchronized...

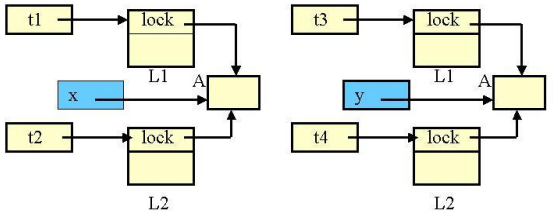
- **O comando**
  `synchronized(obj) {cmds}`
  bloqueia o objeto `obj` **durante a execução de** `cmds`**.**
- **Outras linhas que executarem o comando** `synchronized(obj)` **para o mesmo objeto ficarão em espera até que o objeto seja desbloqueado.**

COMANDO synchronized

```
class L1 extends Thread {
  Object lock;
  public L1(Object obj){lock = obj; }
  ...
  public void run(){... synchronized(lock){...r1...} ...}
}

class L2 extends Thread {
  Object lock;
  public L2(Object obj){lock = obj;}
  ...
  public void run(){... synchronized(lock){...r2...} ...}
}
```


COMANDO synchronized

```
class M {
  static public void main(String[] args) {
    A x = new A(); A y = new A();
    L1 t1 = new L1(x); L1 t3 = new L1(y);
    L2 t2 = new L2(x); L2 t4 = new L2(y);
    t1.start(); t2.start(); t3.start(); t4.start();
}
```



CLASSE `Pressure`: PROPOSTA IV (errada)

```
class Pressure extends Thread {
  synchronized void RaisePressure() {
     if (Controle.pressureGauge < Controle.Limit - 15) {
      try {sleep(100);}
          catch (Exception e){};
      Controle.pressureGauge += 15;
     };
  }
  public void run() {RaisePressure(); }
}
```


CLASSE `Pressure`: PROPOSTA IV (errada)

- **Não funciona, porque**

```
synchronized void RaisePressure() {
    corpo
}
```

é equivalente a

```
void RaisePressure() {
  synchronized(this) { corpo }
}
```

- `synchronized` **somente sincroniza métodos de um** mesmo **objeto.**

MÉTODOS SINCRONIZADOS

- **Métodos** `sincronizados` **de um mesmo objeto são mutuamente exclusivos**
- **Métodos estáticos sincronizados são mutuamente exclusivos.**
- **Métodos não-estáticos** `sincronizados` **NÃO excluem métodos não sincronizados ou métodos estáticos (sincronizados ou não)**
- **Um método sincronizado pode chamar sem bloqueio qualquer outro também sincronizado com mesmo** `this`

MÉTODOS SINCRONIZADOS ...

- **A redefinição de um método sincronizado pode ou não ser** `synchronized`**.**
- `super.method()` **pode ser sincronizado mesmo quando** `method()` **não for.**
- **Se um método não-sincronizado faz uma chamada ao seu** `super` **sincronizado, durante a execução do** `super()` **o objeto fica bloqueado.**

OPERAÇÕES DE SINCRONIZAÇÃO

- **Métodos** `wait` **e** `notify` **estão definidos na classe** `Object`
- **O método** `wait` **permite a espera por uma condição**
- **O método** `notify` **permitir informar àqueles em espera que alguma coisa aconteceu.**
- `wait` **e** `notify` **se aplicam a objetos particulares, da mesma forma que bloqueios, isto é,** `x.notify()` **notifica uma linha que executou** `y.wait()`**, se** `x` **e** `y` **denotarem o mesmo objeto.**
- `wait` **e** `notify` **se aplicam somente a objetos monitores, isto é, objetos que tem métodos ou comandos synchronized.**

OPERAÇÕES DE SINCRONIZAÇÃO ...

- **A execução do** `x.wait()` **libera bloqueio do objeto.**
- **Quando a linha é reativada, o bloqueio do objeto é restabelecido.**
- **Todo objeto** `x` **do tipo monitor têm duas filas associadas:**
  - **fila de bloqueio de threads a espera da liberação do objeto** `x`
  - **fila de espera de threads que executaram** `x.wait()`
- `x.notify()` **pode ser ativado de qualquer lugar.**
- `x.notify()` **acorda apenas uma linha, isto é, passa uma das linhas da fila de espera de** `x` **para a respectiva fila de bloqueio de sincronização do objeto.**
- **Para acordar todas as linhas na fila de espera de um objeto** `x` **usa-se** `x.notifyAll()` **em vez de** `x.notify()`**.**

ESTRUTURA DE PROGRAMA COM SINCRONIZAÇÃO

- **Linha que espera a** `condição`**:**

```
synchronized void execQuandoPuder(){
  ...
  while(!condicao) { x.wait(); };
  ... neste ponto condicao e' verdadeira
}
```

- **Linha que cria a** `condição`**:**

```
synchronized void FazPoder() {
  ...
  muda algum valor usado na condicao do execQuandoPuder();
  ...
  x.notifyAll(); ....
}
```


DETALHES DE `wait`

- `public final void wait(long timeout) throws InterruptedException`:
  A linha corrente espera ser notificada, via o objeto corrente, mas espera no máximo o tempo especificado `timeout` **(ms).**
  Se `timeout==0` **espera notificação indefinidamente.**
- `public final void wait() throws InterruptedException`:
  O mesmo que `wait(0)`**.**
- `public final void wait(long mili, int nanos) throws InterruptedException`:
  O tempo de espera é a soma de milissegundos com nanossegundos.

DETALHES DE `notify`

- `public final void notify()`:
  Notifica exatamente uma das linhas que esperam por uma condição via um `wait` no objeto corrente do `notify`.
  Não é possível escolher quem será notificado. Use com cuidado.

- `public final void notifyAll()`:
  Notifica todas as linhas que esperam por alguma condição.

THREADS SINCRONIZADAS

PROBLEMA DO PRODUTOR X CONSUMIDOR

- **Problema do Consumidor e Produtor que se comunicam via um buffer.**
- **Um produtor continuamente deposita, um a um, inteiros em um buffer**
- **Um consumidor retira continuamente um inteiro do buffer.**

PROBLEMA DO PRODUTOR X CONSUMIDOR ...

```
class Buffer { ... }

class Consumer extends Thread { ... }

class Producer extends Thread { ... }

public class Main{
  public static void main(String args[ ]) {
      Buffer b = new Buffer(100);
      Consumer c1 = new Consumer(b), c2 = new Consumer(b);
      Producer p1 = new Producer(b), p2 = new Producer(b);
      c1.start( ); p1.start( );  c2.start( ); p2.start( );
  }
}
```

DEFINIÇÃO DO CONSUMIDOR E PRODUTOR

```
class Consumer extends Thread {
   private Buffer b;
   public Consumer(Buffer b) {this.b = b; }
   public void run( ) {
       int i;
       while (true) {
             i = b.get();
             System.out.print(i);
             yield( );
       }
   }
}
```


DEFINIÇÃO DO CONSUMIDOR E PRODUTOR ...

```
class Producer Extends Thread {
    private Buffer b;
    public Producer(Buffer b) { this.b = b; }
    public void run( ) {
         int i = 0;
         while (true) {
               b.put(i++);
               yield( );
         }
    }
}
```


DEFINIÇÃO DO BUFFER

```
public class Buffer {
   private int size;
   private int [ ] contents;
   private int in, out;
   private int total = 0;
   Buffer(int size) {
      this.size = size;  contents = new int[size];
   }
   public synchronized void put(int item){ ...}
   public synchronized int get() {... }
}
```


DEFINIÇÃO DO BUFFER ...

```
public synchronized void put(int item){
    while(total >= size) {
        try {this.wait();} catch(InterrupedException e) { };
    }
    contents[in] = item;
    System.out.println("Buffer:write at "+ in + "item" + item);
    if(++in == size) in = 0;
    total++;
    this.notifyAll();
}
```

DEFINIÇÃO DO BUFFER...

```
public synchronized int get() {
    int temp;
    while(total <= 0)
        try {this.wait();} catch(InterrupedException e) { };
    temp = contents[out];
    System.out.println("Buffer:read from "+out+ "item" + temp);
    if(++out == size) out=0;
    total--;
    this.notifyAll();
    return temp;
}
```


LINHAS EGOÍSTAS E COOPERATIVAS

- Em alguns sistemas, linha monopiliza o controle até seu término.
- Em outros, como o Windows NT, o sistema implementa a política de tempo compartilhado, ou seja, toda linha é periodicamente interrompida para que todas recebam, ciclicamente, uma fatia de tempo para execução.
- ENTRETANTO, quando se programa na rede, não se sabe qual o ambiente o programa com as linhas será executado.
- RECOMENDA-SE que toda linha chame `yield` ou `sleep` quando estiver executando um loop longo, assegurando-se que não estará monopolizando o sistema.

ENCERRAMENTO DE THREADS

FIM DE UMA APLICAÇÃO ...

- Toda aplicação começa com uma linha que executa seu método `main`.
- Uma aplicação que não cria outras linhas termina quando seu `main` termina.
- Linhas podem ser user ou daemon.
- Por default, toda linha é do tipo user.
- No caso de haver outras linhas, a aplicação somente termina quando todas as linhas do tipo user terminarem.

FIM DE UMA APLICAÇÃO

- **Aplicação não espera por linhas do tipo daemon, isto é, todas linhas desse tipo morrem junto com a aplicação.**
- **Criam-se linhas daemon com o comando:**
  `setDaemon(true)`

FIM DE UMA LINHA ...

- **Uma linha `t` termina quando:**
  - **seu `run` termina**
  - **`t.stop()` é executado como último recurso (depreciado).**
- **Comando `t.stop()` lança a exceção `ThreadDeath`, uma extensão de `Error`.**
- **Se `ThreadDeath` não for capturada, a linha `t` é encerrada sem emitir qualquer mensagem.**

... FIM DE UMA LINHA

- **A exceção levantada por `this.stop()` pode ser capturada pela própria `Thread this`, anulando sua morte.**
- **Uma linha pode executar diretamente o comando**
  `throw new ThreadDeath().`

FIM DE UMA LINHA (EXEMPLO I) ...

```
import java.io.*;
class T extends Thread {
  public void run() {
    PrintStream o = System.out;
    o.println("T1. Begin");
    stop();
    o.println("T2. Resurrection");
  }
}
SAIDA: M1. Begin
       T1. Begin
       M3. End
```

... FIM DE UMA LINHA (EXEMPLO I)

```
public class Stop1{
  static public void main(String[] args) {
    PrintStream o = System.out;
    T t = new T(); o.println("M1. Begin"); t.start();
    try {Thread.sleep(2000);}
        catch(InterruptedException e)
                    {o.println("M2. Interrupted");};
    o.println("M3. End");
  }
}
SAIDA: M1. Begin
       T1. Begin
       M3. End
```


FIM DE UMA LINHA (EXEMPLO II) ...

```
import java.io.*;
class T extends Thread {
  public void run() {
    PrintStream o = System.out;
    o.println("T1. Begin");

    try {stop();}
    catch(ThreadDeath d){o.println("T2. Post-Mortem Sigh ");}

    o.println("T3. Resurrection");
  }
}
```


... FIM DE UMA LINHA (EXEMPLO II)...

```
public class Stop2{
  static public void main(String[] args) {
    PrintStream o = System.out;
    T t = new T();
    System.out.println("M1. Begin");
    t.start();
    System.out.println("M2. Continues");
    try{ t.join();}
    catch(ThreadDeath d){o.println("M3. Faces Death");}
    catch(InterruptedException d){o.println("M4. Interrupted");}
   o.println("M5. End ");
  }
}
```


... FIM DE UMA LINHA (EXEMPLO II)

```
SAIDA: M1. Begin
       M2. Continues
       T1. Begin
       T2. Post-Mortem Sigh
       T3. Resurrection
       M5. End
```

FIM DE UMA LINHA (EXEMPLO III) ...

```
import java.io.*;
class T extends Thread {
  public void run() {
    PrintStream o = System.out;
    o.println("T1. Begin");
    try{sleep(3000);}
       catch(InterruptedException d){
         o.println("T2. Interrupted");}
    o.println("T3. End");
  }
}
```


... FIM DE UMA LINHA (EXEMPLO III)...

```
public class Stop3{
  static public void main(String[] args) {
    PrintStream o = System.out;
    T t = new T(); o.println("M1. Begin");
    t.start(); o.println("M2. Continues");
    try {Thread.sleep(1000);}
    catch(InterruptedException d){o.println("M3. Interrupted");}
    o.println("M4. Continues");
    try{ t.stop();}
    catch(ThreadDeath d){o.println("M5. Faces Death");}
    o.println("M6. End");
  }
}
```


... FIM DE UMA LINHA (EXEMPLO III)

```
SAIDA:
M1. Begin
M2. Continues
T1. Begin
M4. Continues
M6. End
```


FIM DE UMA LINHA (EXEMPLO IV) ...

```
import java.io.*;
class T extends Thread {
  public void run() {
    PrintStream o = System.out;
    o.println("T1. Begin");
    try{sleep(3000);}
       catch(InterruptedException d){
         o.println("T2. Interrupted");}
       catch(ThreadDeath d){
         o.println("T3. Faces Death");}
    o.println("T4. End");
  }
}
```

... FIM DE UMA LINHA (EXEMPLO IV) ...

```
public class Stop4{
  static public void main(String[] args) {
    PrintStream o = System.out;
    T t = new T(); o.println("M1. Begin"); t.start();
    o.println("M2. Continues");
    try {Thread.sleep(1000);}
    catch(InterruptedException d){o.println("M3. Interrupted");}
    o.println("M4. Continues");
    try{ t.stop();}
    catch(ThreadDeath d){o.println("M5. Faces Death");}
    o.println("M7. End");
  }
}
```


... FIM DE UMA LINHA (EXEMPLO IV)

```
SAIDA DE Stop4: M1. Begin
                M2. Continues
                T1. Begin
                M4. Continues
                M7. End
                T3. Faces Death
                T4. End
```


FIM DE UMA LINHA (EXEMPLO V) ...

```
import java.io.*;
class T extends Thread {
  public void run() {
    PrintStream o = System.out;
    o.println("T1. Begin");
    try{sleep(3000);}
       catch(InterruptedException d){
         o.println("T2. Interrupted");}
       catch(ThreadDeath d){
         o.println("T3. Faces Death");}
    o.println("T4. End");
  }
}
```


... FIM DE UMA LINHA (EXEMPLO V) ...

```
public class Stop5{
  static public void main(String[] args) {
    PrintStream o = System.out;
    T t = new T(); o.println("M1. Begin"); t.start();
    o.println("M2. Continues");
    try {Thread.sleep(1000);}
    catch(InterruptedException d){
         o.println("M3. Interrupted");}
    o.println("M4. Continues");
    t.interrupt();
    o.println("M5. End");
  }
}
```

... FIM DE UMA LINHA (EXEMPLO V)

```
SAIDA DE Stop5:
M1. Begin
M2. Continues
T1. Begin
M4. Continues
M5. End
T2. Interrupted
T4. End
```


FIM

AMBIENTES DE PROGRAMAÇÃO

APPLETS

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

24 de março de 2015

SUMÁRIO

- Criação de Applets
- Ciclo de Vida de Applets
- Exemplos de Applets

CRIAÇÃO DE APPLETS

APPLETS

- **Applets são um meio de executar um código Java a partir de um navegador (browser).**
- **Applets são definidas por um protocolo que governa seu tempo-de-vida e os métodos por meio dos quais pode-se inspecionar ou manipular o ambiente de execução.**
- **A superclasse** `java.applet.Applet` **define o ambiente.**
- **Em Java 2, com** `swing`**, deve-se usar**
  `import javax.swing.JApplet`

CLASSE `Applet`

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Panel
                    |
                    +--java.applet.Applet
```


CLASSE `JApplet`

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Panel
                    |
                    +--java.applet.Applet
                          |
                          +--javax.swing.JApplet
```


EXECUÇÃO DE APPLETS

- **Numa página html, quando o rótulo** `APPLET` **ou** `OBJECT` **for encontrado na página que estiver sendo carregada, o navegador:**
  - **Carrega a código (bytecode) da classe especificada.**
  - **Cria um objeto dessa classe.**
  - **Cria uma região na página da WEB para ser controlada pela applet.**
  - **Chama em sequência os métodos** `init`, `start` **e** `paint`**.**
- **Durante a navegação outros métodos de applets são automaticamente chamados.**
- **Outras classes poderão ser carregadas ao longo da execução da applet.**

CRIAÇÃO DE APPLETS

- **Para criar uma applet, deve-se:**
  - **Criar um arquivo com extensão** `.java` **para ser o tipo da applet.**
  - **Criar nesse arquivo a classe que define a applet como uma extensão de** `java.applet.Applet` **ou de** `javax.swing.JApplet`
  - **Mencionar o nome da applet na página HTML, junto com definição de parâmetros, área de trabalho, etc.**

UMA APPLET MUITO SIMPLES I

- **Arquivo Mensagem.java:**

```
import java.awt.Graphics; import java.applet.Applet;
public class Mensagem extends Applet {
   public void paint(Graphics g){
      g.drawString("Sim!", 15, 25);
   }
}
```

- **Arquivo Mensagem.java:**

```
import java.awt.Graphics; import javax.swing;
public class Mensagem extends JApplet {
   public void paint(Graphics g) {
      g.drawString("Sim!", 15, 25); }
}
```


UMA APPLET MUITO SIMPLES I ...

- **Arquivo Pagina1.html:**

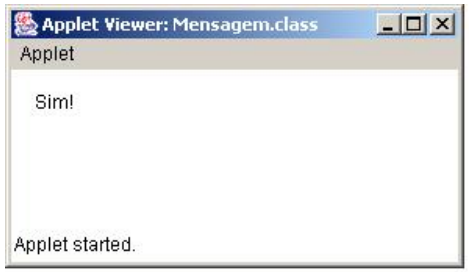
```
<title> Applet muito simples</title>
<applet code="Mensagem.class"
        width=200 height=100>
</applet>
```

- **Arquivo Pagina2.html:**

```
<title> Applet muito simples</title>
<object codetype=application/java"
        classid = "Mensagem.class"
        width=200 height=100>
</object>
```


UMA APPLET MUITO SIMPLES I...

```
Z:\>appletviewer Pagina1.html:
```



UMA APPLET MUITO SIMPLES II

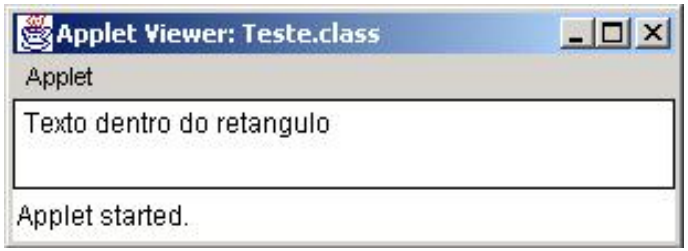
Arquivo `Teste.html`**:**

```
<HTML>
  <APPLET Code="Teste.class", width=300 height=40></APPLET>
</HTML>
```

Arquivo `Teste.java`**:**

```
import java.applet.Applet; import java.awt.Graphics;
public class Teste extends Applet {
   public void init() { repaint(); }
   public void paint(Graphics g) {
     Dimension d = getSize();
     g.drawRect(0, 0, d.width - 1, d.height - 1);
     g.drawString("Texto dentro do retangulo", 5, 15);
   }
}
```


UMA APPLET MUITO SIMPLES II...



CICLO DE VIDA DE APPLETS

MÉTODOS DO CICLO DE VIDA DA APPLET

- `void init()`:
  Chamado automaticamente quando a applet for carregada a primeira vez.
  Se a applet usa threads, `init` **deve tipicamente criar essas threads.**

- `void start()`:
  Chamado automaticamente sempre que o browser visitar a página que contém a applet.
  Isto é, chamado em resposta à pressão dos botões Back **e** Forward **do navegador.**

MÉTODOS DO CICLO DE VIDA DA APPLET ...

- `void stop()`:
  Chamado automaticamente sempre que o navegador sair da página que contém a applet.
  Isto é, chamado em resposta à pressão dos botões Back e Forward do navegador.

- `void destroy()`:
  Chamado quando a página não for mais alcançável.
  É usado para liberar recursos não mais necessários.

- `void paint(Graphics g)`:
  chamado automaticamente pelo sistema de janela quando o componente tiver que ser (re)exibido.

FLUXO DE EXECUÇÃO DE APPLETS

1. **Browser faz o primeiro acesso à página com applet embutida.**
2. **Applet referenciada é carregada.**
3. **Método `init()` é automaticamente chamado.**
4. **Browser chama o método `start()`.**
   Applet precisa de `start` quando cria linhas de execução.
5. **A seguir, o browser chama o método `paint()`, que é um método da janela para desenha a parte gráfica do applet.**
6. **Neste ponto a applet está sendo executada, isto é, seu efeito pode ser observado na página.**

FLUXO DE EXECUÇÃO DE APPLETS ...

1. **Quando o browser deixar a página, o método `stop()` é automaticamente chamado para fazer qualquer limpeza necessária.**
2. **Applet precisa de `stop()` sempre que limpeza for necessária.**
3. **Limpeza típica pode ser interromper linhas de execução geradas pela applet.**
4. **Todo retorno à página causa chamada automática a `start()` e a `paint()`.**
5. **Quando a página for descartada permanentemente, método `destroy()` é chamado.**

EXEMPLOS DE APPLETS

APPLET SOMADOR(AddtionApplet.html)

```
<html>
<hr>
<h2>Applet ilustra entrada e saida de dados via browser</h2>

<applet code = "AdditionApplet.class"
                width = "300" height = "50">
</applet>

<h2>Area da applet está logo acima</h2>
<hr>
</html>
```


APPLET SOMADOR(AddtionApplet.java)

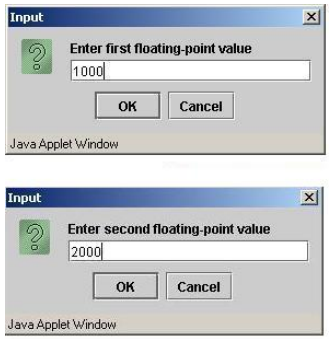
```
import java.awt.Graphics;
import javax.swing.*;
public class AdditionApplet extends JApplet {
   double sum;  // sum of values entered by user

   public void init(){  ... }

   public void paint( Graphics g ) { ...}
}
```


APPLET SOMADOR (init)

```
public void init(){
      String firstNumber;   // first string entered by user
      String secondNumber;  // second string entered by user
      double number1;       // first number to add
      double number2;       // second number to add
      firstNumber = JOptionPane.showInputDialog(
                  "Enter first floating-point value" );
      secondNumber = JOptionPane.showInputDialog(
                  "Enter second floating-point value" );
      number1 = Double.parseDouble(firstNumber);
      number2 = Double.parseDouble(secondNumber);
      sum = number1 + number2;
}
```


APPLET SOMADOR - LEITURA

Resultado após execução do método `init`:

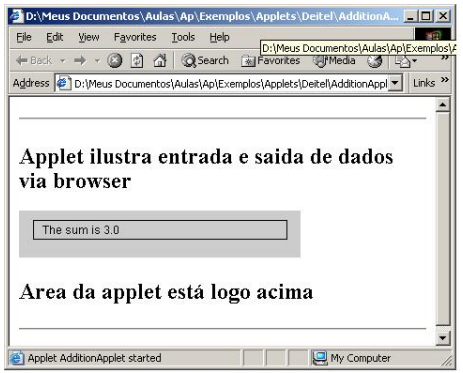


APPLET SOMADOR (paint)

```
// desenha resultados na área da applet

public void paint( Graphics g ){
  super.paint(g);

  // draw rectangle starting from (15, 10) that is 270
  // pixels wide and 20 pixels tall
  g.drawRect( 15, 10, 270, 20 );

  // draw results as a String at (25, 25)
  g.drawString( "The sum is " + sum, 25, 25 );
}
```


APPLET SOMADOR - RESULTADOS

Após execução do método `paint`:



APPLET SOMADOR(`AddtionApplet.java`) Versão II

```
import java.awt.Container;

import javax.swing.*;

public class AdditionApplet extends JApplet {
   double sum;
   double number1;
   double number2;

   public void init( ) { ... }

}
```


APPLET SOMADOR(`init`) Versão II ...

```
public void init() {
   String firstNumber;  String secondNumber;
   JTextArea outputArea = new JTextArea();
   Container container = getContentPane();
   container.add(outputArea);
   firstNumber = JOptionPane.showInputDialog(
         "Enter first floating-point value" );
   secondNumber = JOptionPane.showInputDialog(
         "Enter second floating-point value" );
   number1 = Double.parseDouble( firstNumber );
   number2 = Double.parseDouble( secondNumber );
   sum = number1 + number2;
   outputArea.setText("The sum is " + sum);
}
```

TRANSFERINDO PARÂMETROS

- **Obedece a mesma convenção de argumentos na linha de comando de main, isto é, os parametros são transferidos como** `Strings`.
- **Função** `getParameter(nome do param)` **retorna o string que representa o valor.**
- **No arquivo anagrama.html:**

```
...
<applet code="Anagrama.class" width=500 height=500>
   <param name="objetivo" value="surfando na rede">
   <param name="palavras" value="palavra.txt">
   <param name="tamanho" value="2">
   <param name="valor" value="3.14">
</applet>
...
```


TRANSFERINDO PARÂMETROS...

- **No arquivo Anagrama.java:**

```
import java.applet.Applet;
import java.awt.Event;
import java.awt.TextField;
class Anagrama extends Applet {
   public void paint( ) {
     String s1 = getParameter("tamanho");
     int k = Integer.parseInt(s1);
     String s2 = getParameter("objetivo");
     String s3 = getParameter("valor");
     double d  = Double.valueOf(s3).doubleValue();
    }
}
```


CAICULADORA - SEM APPLET

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

public class Calculator {
   public static void main(String[] args){
       JFrame frame = new CalculatorFrame();
       frame.show();
   }
}
```


SAÍDA DE `java Calculator`

QUADRO DA CALCULADORA

```
class CalculatorFrame extends JFrame {
  public CalculatorFrame() {
      setTitle("Calculator");
      setSize(200, 200);
      addWindowListener(new WindowAdapter( ) {
          public void windowClosing(WindowEvent e){
              System.exit(0);
          }
      } );

      Container contentPane = getContentPane();
      contentPane.add(new CalculatorPanel());
   }
}
```


PAINEL DO QUADRO DA CALCULADORA

```
class CalculatorPanel extends JPanel
               implements ActionListener {
   private JTextField display;
   private double arg = 0;
   private String op = "=";
   private boolean start = true;
   public CalculatorPanel() {...}
   private void addButton(Container c, String s){...}
   public void actionPerformed(ActionEvent evt){...}
   public void calculate(double n){...}
}
```


CONSTRUTOR DE `CalculatorPanel`

```
public CalculatorPanel(){
      setLayout(new BorderLayout());
      display = new JTextField("0");
      display.setEditable(false);
      add(display, "North");
      JPanel p = new JPanel();
      p.setLayout(new GridLayout(4, 4));
      String buttons = "789/456*123-0.=+";
      for (int i = 0; i < buttons.length(); i++)
         addButton(p, buttons.substring(i, i + 1));
      add(p, "Center");
}
```


MÉTODO `addButton`

```
private void addButton(Container c, String s) {

       JButton b = new JButton(s);
       c.add(b);
       b.addActionListener(this);

}
```

MÉTODO actionPerformed

```
public void actionPerformed(ActionEvent evt){
  String s = evt.getActionCommand();
  if ('0'<=s.charAt(0) && s.charAt(0)<='9'||s.equals(".")){
     if (start) {display.setText(s); start = false;}
     else display.setText(display.getText() + s);
  } else {
     if (start) {
        if (s.equals("-")){display.setText(s); start = false;}
        else op = s;
     } else {
         double x = Double.parseDouble(display.getText());
         calculate(x); op = s; start = true; }
  }
}
```


MÉTODO Calculate

```
public void calculate(double n, String op){
      if (op.equals("+")) arg += n;
      else if (op.equals("-")) arg -= n;
      else if (op.equals("*")) arg *= n;
      else if (op.equals("/")) arg /= n;
      else if (op.equals("=")) arg = n;
      display.setText("" + arg);
}
```


A PÁGINA DA CALCULADORA

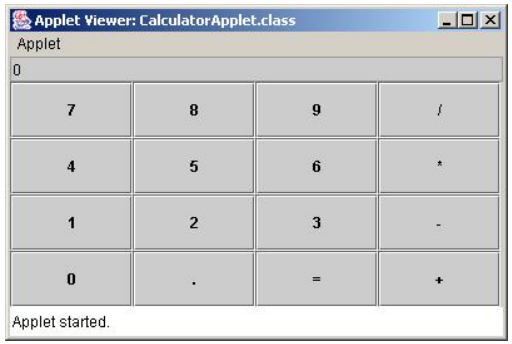
```
<HTML>
<TITLE> Uma Calculadora </TITLE>
<BODY>
  Eis aqui uma calculadora.
<HR>

<APPLET code = "CalculatorApplet.class" width=250 height=200>
</APPLET>

<HR>

</BODY>
</HTML>
```


CAICULADORA APPLET

- **No lugar das classes** `Calculator` **e** `CalculatorFrame` **tem-se apenas** `CalculatorApplet`
- **Classe** `CalculatorPanel` **fica inalterada.**
- **Define-se o Applet com sendo:**

```
public class CalculatorApplet extends JApplet {
   public void init( ) {

      Container contentPane = getContentPane();
      contentPane.add(new CalculatorPanel( ) );

   }
}
```

SAIDA DE `appletviewer CalculatorApplet.html`



SAIDA DO NAVEGADOR PARA `CalculatorApplet.html`

APPLETS X SEGURANÇA

- Applets são executadas sob controle sandbox
- No `sandbox` operações perigosas como acesso a arquivos, acesso à rede, execução de programa local, são proibidas ou limitadas pelo security manager.
- Classes podem ser assinadas digitalmente de forma a dar garantias quanto sua identificação.
- Applets confiáveis podem receber direitos de acessos privilegiados.
- Usuários podem configurar seus browsers com uma lista de organizações confiáveis, cujos applets podem ser executados automaticamente com mais privilégios.

FIM

AMBIENTES DE PROGRAMAÇÃO
REFLEXÃO COMPUTACIONAL EM JAVA

Roberto da Silva Bigonha

Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

19 de março de 2015

SUMÁRIO

- **Clonagem de Objetos**
- **Classe Class**

CLONAGEM DE OBJETOS

A MÃE DE TODAS AS CLASSES

Classes automaticamente estendem `Object`, cujos métodos são:

- `public boolean equals(Object obj)`
  Compara objetos receptor e referenciado.
- `public int hashcode( )`
  Retorna o código hash do objeto receptor.
- `protected Object clone( )`
  Retorna uma referência a um clone do objeto receptor.
- `public final Class getclass( )`
  Retorna o objeto do tipo Class que representa a classe do objeto receptor.
- `protected void finalize( )`
  Finaliza o objeto receptor durante a coleta de lixo.

CLONAGEM DE OBJETOS

- **O método `clone` da class `Object` é `protegido`**
- **Somente o próprio objeto pode solicitar sua duplicação.**
- **Método padrão `clone` faz duplicação bit a bit, portanto pode gerar compartilhamento de campos entre os objetos duplicados (shallow copy).**
- **É da responsabilidade do programador redefinir `clone` para realizar a propagagação das cópias**
- **Classes de objetos clonáveis devem implementar a interface `Cloneable`, senão será levantada uma exceção quando se requisitar clonagem.**

CLONAGEM DE OBJETOS ...

```
public class Dia implements Cloneable {
   private int, dia, mes, ano;
   public Dia(int dia, int mes, int ano){
       this.dia = dia; this.mes = mes; this.ano = ano;
   }
   public void altera(int dia, int mes, int ano){
       this.dia = dia; this.mes = mes; this.ano = ano;
   }

   public Object clone() {
       return super.clone( );
   }
}
```


CLONAGEM DE OBJETOS ...

```
public class Teste {
  public static void main(String[ ] args) {
       Dia d1 = new Dia(30,3,2004);
       Dia d2 = d1;
       Dia d3 = (Dia) d1.clone( );
       d1.altera(1,4,2004);
  }
}
```

CLONAGEM DE OBJETOS ...

```
public class Empregado implements Cloneable {
   private Dia dataContratacao;
   private String nome;
   public Empregado(String nome, Dia data) {
      this.nome =nome;
      this.dataContratacao = data;
   end;
   public String getname(){return nome}
   public Object clone() {
      Empregado e = (Empregado) super.clone();
      e.dataContratacao = (Dia)dataContratacao.clone();
      return e;
   }
}
```


CLASSE CLASS

CLASSE `java.lang.Class`

- `String getName()`:
  `c.getname()` **retorna o nome da class descrita por** `c`
- `Class getSuperClass()`:
  `c.getSuperClass()` **retorna a superclasse de** `c`.
- `Class[ ] getInterfaces()`:
  `c.getInterfaces()` **retorna um arranjo de objetos** `Class` **que informa as interfaces implementadas por** `c`.
- `boolean isInterface()`
- `String toString()(`:
  `c.toString()` **retorna o nome da classe ou interface descrita pelo objeto** `c`.

CLASSE `java.lang.Class` ...

- `static Class forName(String className)`:
  retorna o objeto `Class` **que representa a classe de nome** `className`
- `Object newInstance()`:
  retorna uma nova instância desta classe
- `Field[ ] getFields()`:
  retorna um arranjo contendo os objetos `Field` **dos campos públicos.**
- `Field[ ] getDeclaredFields`:
  retorna um arranjo contendo os objetos `Field` **de todos os campos.**

CLASSE `java.lang.Class` ...

- `Method[ ] getMethods()`:
  retorna um arranjo contendo os objetos `Method` dos métodos públicos.
- `Method[ ] getDeclaredMethods`:
  retorna um arranjo contendo os objetos `Methods` de todos os métodos.
- `Constructor[ ] getConstructors()`:
  retorna um arranjo contendo os objetos `Construtor` que fornece os construtores públicos.
- `Constructor[ ] getDeclaredConstructors`:
  retorna um arranjo contendo os objetos `Constructor` que fornece todos os construtores;

OBTENÇÃO DO OBJETO `Class` DADO O OBJETO

- **Uso de `getClass()`:**

```
Empregado e;
Class c = e.getClass();
...
Methods [ ] m = c.getMethods();
```


OBTENÇÃO DO OBJETO `Class` DADO O NOME DA CLASSE

- **Uso de `forName(String)`:**

```
String nome = "Empregado";
Class c = Class.forName(nome);
...
Methods [ ] m = c.getMethods();

Object x = c.newInstance();
```


OBTENÇÃO DO NOME DE UMA CLASSE EXISTENTE

- **O trecho de programa**

```
Empregado e = new Empregado("José da Silva",data);

Class c = e.getclass();

System.out.println(c.getName() + " " + e.getname());
```

- **IMPRIME** `Empregado José da Silva`

CLASSES `java.lang.reflect.Field`, `Method` e `Constructor`

- `Class getDeclaringClass()`: **retorna o objeto** `Class` **da classe que define este construtor, método ou campo.**
- `Class[ ] getExceptionTypes()`: **retorna um arranjo de objetos** `class` **que representam os tipos de exceções levantadas pelo método.**
- `int getModifiers()`: **retorna um inteiro que descreve os modificadores desse construtor, método ou campo. (Use classe** `Modifier` **para analisar o inteiro retornado).**
- `String getName()`: **retorna** `String` **que é o nome do construtor, método ou campo.**
- `Class[ ] getParameterTypes`: **retorna um arranjo de objetos** `Class` **representam os tipos dos parâmetros desse construtor ou método.**

CLASSE `java.lang.reflect.Modifier`

- **public static java.lang.String toString(int);**
- **public static boolean isInterface(int);**
- **public static boolean isPrivate(int);**
- **public static boolean isTransient(int);**
- **public static boolean isStatic(int);**
- **public static boolean isPublic(int);**
- **public static boolean isProtected(int);**
- **public static boolean isFinal(int);**
- **public static boolean isSynchronized(int);**
- **public static boolean isVolatile(int);**
- **public static boolean isNative(int);**
- **public static boolean isAbstract(int);**
- **public static boolean isStrict(int);**

TESTE DA REFLEXÃO

```
import java.lang.reflect.*;
import CoreJava.*;

public class ReflectionTest {

   public static void printConstructors(Class cl) {...}

   public static void printMethods(Class cl){ ... }

   public static void printFields(Class cl){ ... }

   public static void main(String[] args) { ...}

}
```


TESTE DA REFLEXÃO ...

```
public static void main(String[] args) {
   String name = Console.readLine
       ("Please enter a class name (e.g. java.util.Date): ");
   try {Class cl = Class.forName(name);
        Class supercl = cl.getSuperclass();
        System.out.print("class " + name);
        if (supercl != null && !supercl.equals(Object.class))
           System.out.print(" extends " + supercl.getName());
        "Imprime construtores, métodos e campos de cl";
   }
   catch(ClassNotFoundException e) {
        System.out.println("Class not found.");
   }
}
```

IMPRIME CONSTRUTORES, MÉTODOS E CAMPOS DE `cl`

```
System.out.print("\n{\n");

printConstructors(cl);
System.out.println();

printMethods(cl);
System.out.println();

printFields(cl);
System.out.println("}");
```


TESTE DA REFLEXÃO ...

```
public static void printConstructors(Class cl) {
   Constructor[] constructors = cl.getDeclaredConstructors();
   for (int i = 0; i < constructors.length; i++) {
      Constructor c = constructors[i];
      Class[] paramTypes = c.getParameterTypes();
      String name = c.getName();
      System.out.print(Modifier.toString(c.getModifiers()));
      System.out.print(" " + name + "(");
      for (int j = 0; j < paramTypes.length; j++)
         {if (j > 0) System.out.print(", ");
          System.out.print(paramTypes[j].getName()); }
      System.out.println(");");
   }
}
```


TESTE DA REFLEXÃO ...

```
public static void printMethods(Class cl){
   Method[] methods = cl.getDeclaredMethods();
   for (int i = 0; i < methods.length; i++) {
     Method m = methods[i]; String name = m.getName();
     Class retType = m.getReturnType();
     Class[] paramTypes = m.getParameterTypes();
     System.out.print(Modifier.toString(m.getModifiers()));
     System.out.print(" " +retType.getName()+ " " +name+ "(");
     for (int j = 0; j < paramTypes.length; j++)
         {if (j > 0) System.out.print(", ");
          System.out.print(paramTypes[j].getName()); }
     System.out.println(");");
   }
}
```


TESTE DA REFLEXÃO ...

```
public static void printFields(Class cl){
   Field[] fields = cl.getDeclaredFields();
   for (int i = 0; i < fields.length; i++) {
      Field f = fields[i];
      Class type = f.getType();
      String name = f.getName();
      System.out.print(Modifier.toString(f.getModifiers()));
      System.out.println(" " + type.getName() + " " + name
         + ";");
   }
}
```

SAÍDA DE `ReflectionTest` PARA `java.lang.Object`

```
class java.lang.Object {
public java.lang.Object();
public native int hashCode();
protected void finalize();
public final void wait();
public final void wait(long, int);
public final native void wait(long);
private static native void registerNatives();
public final native java.lang.Class getClass();
public boolean equals(java.lang.Object);
protected native java.lang.Object clone();
public java.lang.String toString();
public final native void notify();
public final native void notifyAll();}
```


FIM

FIM