



Primeiros Problemas

```
class PrimeirosExemplos
{ static int quadrado (int x) { return x*x; }

static int somaDosQuadrados (int x, int y)
{ return (quadrado(x) + quadrado(y)); }

static boolean tresIguais (int a, int b, int c)
{ return ((a==b) && (b==c)); }

static boolean eTriang (int a, int b, int c)
// a, b e c positivos e cada um menor do que a soma dos outros dois
{ return (a>0) && (b>0) && (c>0) &&
        (a<b+c) && (b<a+c) && (c<a+b); }
```

escute

analise

pergunte

pense

Continuação: *PrimeirosExemplos*

```
static int max (int a, int b)
{ if (a >= b) return a; else return b;}

static int max3 (int a, int b, int c)
{ return (max(max(a,b),c)); }

} // fim PrimeirosExemplos
```

Associe itens nos quadros abaixo:

- | | |
|---|------------------------|
| a. <code>return e</code> | b. <code>static</code> |
| c. <code>static tipo nome(lista-de-parâmetros) { corpo }</code> | |
| d. <code>nome(lista-de-argumentos)</code> | |

- | |
|--|
| 1. forma (sintaxe) de chamada de função ou procedimento (na mesma classe em que é definida(o)) |
| 2. pode ser entendido por enquanto como indicação de definição de função/procedimento |
| 3. forma (sintaxe) de definição de função ou procedimento |
| 4. deve ocorrer no corpo de função; especifica resultado de chamada |



Construções sintáticas

- e representa uma expressão
- **lista-de-parâmetros** é elemento sintático na forma $\text{tipo}_1 \text{nome}_1, \text{tipo}_2 \text{nome}_2, \dots, \text{tipo}_n \text{nome}_n$ ($n \geq 0$)
- **corpo** é uma lista de comandos $c_1; c_2; \dots c_n;$
- **lista-de-argumentos** é da forma $\text{exp}_1, \text{exp}_2, \dots, \text{exp}_n$ (para $n \geq 0$)



Operadores de comparação

Operador	Significado	Exemplo	Resultado
==	Igual a	1 == 1	true
!=	Diferente de	1 != 1	false
<	Menor que	1 < 1	false
>	Maior que	1 > 1	false
<=	Menor ou igual a	1 <= 1	true
>=	Maior ou igual a	1 >= 1	true

 Primeiros Problemas

Expressão condicional

Expressão condicional: $e \ ? \ e_1 \ : \ e_2$

Comando condicional: `if (e) c1; else c2;`

Definições alternativas de *max* e *max3*:

```
int max (int a, int b)
{ return (a >= b ? a : b); }
```

```
int max3 (int a, int b, int c)
{ return (a >= b ? max(a, c) : max(b, c)); }
```

Constantes

```
static final int NaoETriang = 0, Equilatero = 1,  
               Isosceles     = 2, Escaleno    = 3;  
  
static int t_ou_nao_t (int a, int b, int c)  
{ if (eTriang(a,b,c))  
    if (a==b && b==c) return Equilatero;  
    else if (a==b || b==c || a==c) return Isosceles;  
    else return Escaleno;  
else return NaoETriang; }
```

Operadores aritméticos

Operador	Significado	Exemplo	Resultado
+	Adição	2 + 1	3
		2 + 1.0	3.0
		2.0 + 1.0	3.0
-	Subtração	2 - 1	1
		2.0 - 1	1.0
-	Negação	-1	-1
		-1.0	-1.0
*	Multiplicação	2 * 3	6
		2.0 * 3	6.0
/	Divisão	5 / 2	2
		5 / 2.0	2.5
%	Resto	5 % 2	1
		5.0 % 2.0	1.0

Ordem de avaliação de expressões

Da esquerda para a direita,
mas respeitando a precedência de operadores e uso de parênteses

Não é boa técnica de programação definir programas em que resultado dependa de efeitos colaterais e ordem de avaliação:

```
class Exemplo
{ public static void main (String[] a)
  { int i = 10;
    int j = (i=2) * i;
    System.out.println(j);
  }
}
```

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv$

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv (i * j) / k$
$+, -$	$i + j * k \equiv$

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv (i * j) / k$
$+, -$	$i + j * k \equiv i + (j * k)$
$>, <, >=, <=$	$i < j + k \equiv$

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv (i * j) / k$
$+, -$	$i + j * k \equiv i + (j * k)$
$>, <, >=, <=$	$i < j + k \equiv i < (j + k)$
$==, !=$	$b == i < j \equiv$

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv (i * j) / k$
$+, -$	$i + j * k \equiv i + (j * k)$
$>, <, >=, <=$	$i < j + k \equiv i < (j + k)$
$==, !=$	$b == i < j \equiv b == (i < j)$
$\&$	$b \& b1 == b2 \equiv$

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv (i * j) / k$
$+, -$	$i + j * k \equiv i + (j * k)$
$>, <, >=, <=$	$i < j + k \equiv i < (j + k)$
$==, !=$	$b == i < j \equiv b == (i < j)$
$\&$	$b \& b1 == b2 \equiv b \& (b1 == b2)$
\wedge	$b1 \wedge b2 \& b \equiv$

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv (i * j) / k$
$+, -$	$i + j * k \equiv i + (j * k)$
$>, <, >=, <=$	$i < j + k \equiv i < (j + k)$
$==, !=$	$b == i < j \equiv b == (i < j)$
$\&$	$b \& b1 == b2 \equiv b \& (b1 == b2)$
\wedge	$b1 \wedge b2 \& b \equiv b1 \wedge (b2 \& b)$
$ $	$i < j b1 \& b2 \equiv$

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv (i * j) / k$
$+, -$	$i + j * k \equiv i + (j * k)$
$>, <, >=, <=$	$i < j + k \equiv i < (j + k)$
$==, !=$	$b == i < j \equiv b == (i < j)$
$\&$	$b \& b1 == b2 \equiv b \& (b1 == b2)$
\wedge	$b1 \wedge b2 \& b \equiv b1 \wedge (b2 \& b)$
$ $	$i < j b1 \& b2 \equiv (i < j) (b1 \& b2)$
$\&\&$	$i + j != k \&\& b1 \equiv$

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv (i * j) / k$
$+, -$	$i + j * k \equiv i + (j * k)$
$>, <, >=, <=$	$i < j + k \equiv i < (j + k)$
$==, !=$	$b == i < j \equiv b == (i < j)$
$\&$	$b \& b1 == b2 \equiv b \& (b1 == b2)$
\wedge	$b1 \wedge b2 \& b \equiv b1 \wedge (b2 \& b)$
$ $	$i < j b1 \& b2 \equiv (i < j) (b1 \& b2)$
$\&\&$	$i + j != k \&\& b1 \equiv ((i + j) != k) \&\& b1$
$ $	$i1 < i2 b \&\& j < k \equiv$

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv (i * j) / k$
$+, -$	$i + j * k \equiv i + (j * k)$
$>, <, >=, <=$	$i < j + k \equiv i < (j + k)$
$==, !=$	$b == i < j \equiv b == (i < j)$
$\&$	$b \& b1 == b2 \equiv b \& (b1 == b2)$
\wedge	$b1 \wedge b2 \& b \equiv b1 \wedge (b2 \& b)$
$ $	$i < j b1 \& b2 \equiv (i < j) (b1 \& b2)$
$\&\&$	$i + j != k \&\& b1 \equiv ((i + j) != k) \&\& b1$
$ $	$i1 < i2 b \&\& j < k \equiv (i1 < i2) (b \&\& (j < k))$
$? :$	$i < j b ? b1 : b2 \equiv$

Precedência de operadores

Precedência maior	
Operadores	Exemplos
$*, /, \%$	$i * j / k \equiv (i * j) / k$
$+, -$	$i + j * k \equiv i + (j * k)$
$>, <, >=, <=$	$i < j + k \equiv i < (j + k)$
$==, !=$	$b == i < j \equiv b == (i < j)$
$\&$	$b \& b1 == b2 \equiv b \& (b1 == b2)$
\wedge	$b1 \wedge b2 \& b \equiv b1 \wedge (b2 \& b)$
$ $	$i < j b1 \& b2 \equiv (i < j) (b1 \& b2)$
$\&\&$	$i + j != k \&\& b1 \equiv ((i + j) != k) \&\& b1$
$ $	$i1 < i2 b \&\& j < k \equiv (i1 < i2) (b \&\& (j < k))$
$? :$	$i < j b ? b1 : b2 \equiv ((i < j) b) ? b1 : b2$
Precedência menor	

Tipos numéricos

Classificação	Nome	Tamanho ¹	Exemplos de literal
inteiro	byte	8	∅ ²
	short	16	∅
	int	32	10
	long	64	10L 10l ³
de ponto flutuante	float	32	2.718f 2e2f
	double	64	2.718 2e2

¹Número de bits usado para representar valores do tipo.

²Quando um literal inteiro é usado em um contexto que requer um valor de tipo byte ou short, ocorre automaticamente (implicitamente) uma conversão de tipo.

³A letra l minúscula pode, mas em geral não deve, ser usada, por causar confusão com 1.


Operadores booleanos

Operação	Resultado
"não"	
!true	false
!false	true

Operação	Resultado		
	("e") op = & ou &&	("ou") op = ou	("ou exclusivo") op = ^
true op true	true	true	false
true op false	false	true	true
false op true	false	true	true
false op false	false	false	false

Operadores booleanos

Operador	Significado
!	Negação (“não”)
& e &&	Conjunção estrita e não-estrita, resp.
e	Disjunção estrita e não-estrita, resp.
^	Disjunção exclusiva (“ou exclusivo”), estrita.

- Avaliação de $e_1 \ \&\& \ e_2$ retorna false se a de e_1 retornar false; caso contrário, resultado de e_2 . Note: e_2 só é avaliada se avaliação de e_1 retornar true. Def. de $||$ é análoga.
- Função estrita e não estrita: 



Funções estritas

- Considere que \perp representa “valor indefinido”: resultante de avaliação que nunca termina (como, veremos, é possível) ou que provoca a ocorrência de um erro (como, por ex., divisão por zero).
- Função f **estrita**: $f(\perp) = \perp$
- f não-estrita se avaliação puder fornecer resultado mesmo que avaliação de argumento não possa.
- **&& não-estrita** (como função que recebe dois valores).
Ex: `(false && (0/0==0))` é igual a `false`.





Caracteres

- Escritos entre aspas simples: `'a'` `'*'` `'3'` etc.
- “Caracteres de controle” (ex: fim de arquivo, tabulação, mudança de linha etc.), `'` e `\` escritos usando `\` :
 - `'\n'` indica terminação de linha `'\t'` indica tabulação
 - `'\''` indica o caractere `'` `'\\'` indica o caractere `\`
- Representação binária: chamada de *código* do caractere.
- Codificação (“código”) *Unicode* usada em Java: associa um código para cada caractere. São usados dois bytes para cada caractere.





Caracteres

```
static boolean minusc (char x)  
{ return (x >= 'a') && (x <= 'z'); }
```

```
static boolean maiusc (char x)  
{ return (x >= 'A') && (x <= 'Z'); }
```

```
static boolean digito (char x)  
{ return (x >= '0') && (x <= '9'); }
```





Caracteres

Valor de tipo char pode ser usado como inteiro
(de 16 bits, não-negativo)

```
static char minusc_maiusc (char x)  
{ int d = 'A' - 'a';  
  if (minusc(x)) return (x+d);  
  else return x;  
}
```





Caracteres

Caracteres podem ser expressos por meio do valor da sua representação no código *Unicode*.

'\u0000'	caractere nulo
'\u0009'	caractere de tabulação ('\t')
'\u0097'	letra minúscula 'a'



Primeiros Problemas



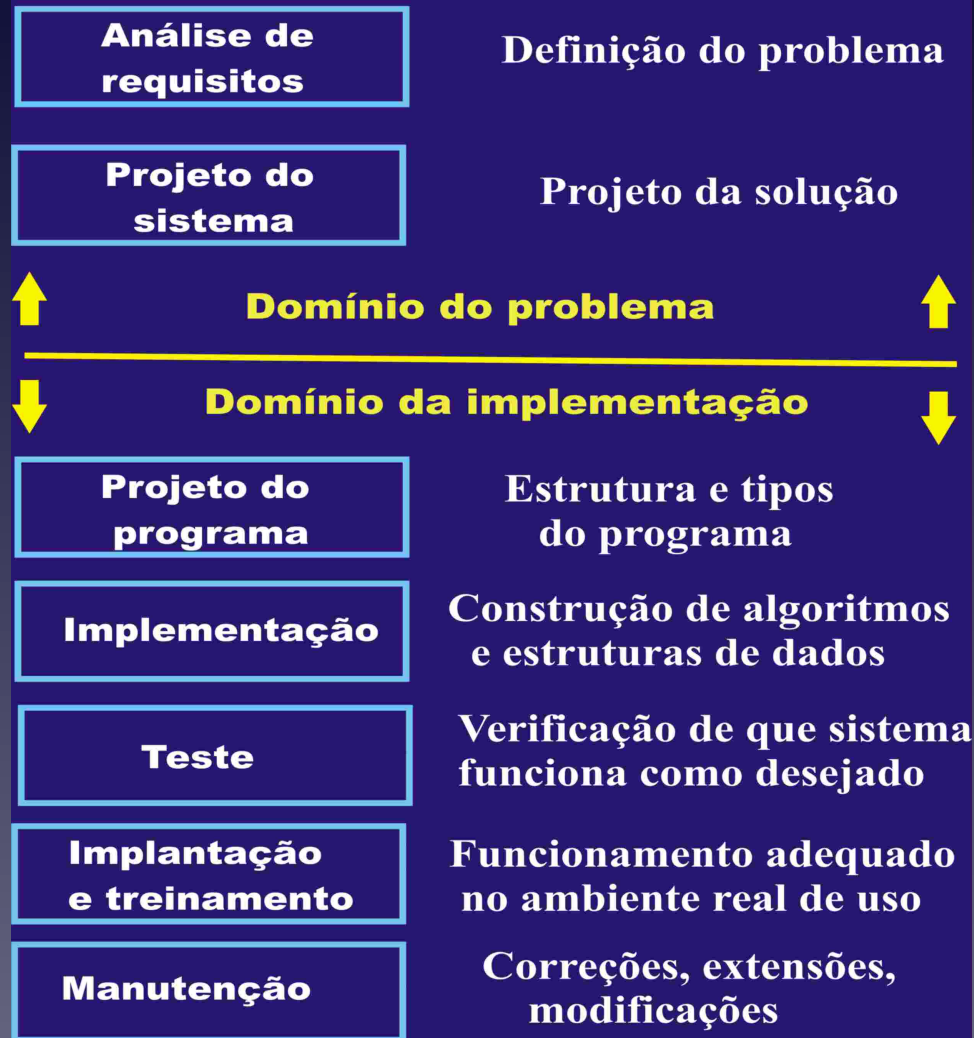
Programação baseada em objetos: primeiras noções

Projeto de sistemas baseado em paradigma de simulação:

entidade do sistema \Leftrightarrow objeto

Objeto criado durante execução do programa, com atributos e comportamento descritos no programa, simula comportamento de entidade do sistema.

Possíveis fases do desenvolvimento de software





Classes e objetos

- **Classe**: parte de um programa que define “estrutura” e “comportamento” de grupo de objetos
- **Objeto** — *instância* de uma classe — existe durante a execução de um programa
- Estrutura definida por **variáveis de objeto** (ou variáveis de instância)
- Comportamento definido por **métodos** e **construtores**.





```
class Ponto
{ int x, y;

    Ponto (int a, int b) { x = a; y = b; }

    void move (int dx, int dy) {x+=dx; y+=dy; }

    double distancia (Ponto p)
    { int dx = this.x - p.x;
      int dy = this.y - p.y;
      return Math.sqrt(dx*dx + dy*dy); }
}

class TestePonto
{ public static void main(String[] a)
  { Ponto p1 = new Ponto(0,0);
    Ponto p2 = new Ponto(10,20);
    p1.move(3,25);
    p2.move(1,14);
    System.out.println(p1.distancia(p2)); }
}
```

escute

analise

pergunte

pense





Criação de objetos

- Objeto criado por *efeito colateral* de expressão de criação de objeto — ex: `new Ponto(0,0)` — que retorna referência ao objeto criado.
- Referência ao objeto criado significa, em termos de implementação, endereço do início da área de memória alocada para as variáveis do objeto.



Valor inicial de variáveis

Variável de objeto ou de classe tem valor inicial *default*, atribuído se nenhum valor inicial for especificado na declaração, que depende do tipo da variável:

byte, short, int, long	0
char	'\u0000'
boolean	false
float, double	0.0
classes	null

Ao contrário de variáveis locais de métodos.



Chamada de método

$exp.método(param_1, \dots, param_n)$

- Tipo de *exp* deve ser alguma classe
- Deve existir definição do método *método* nessa classe
- Essa definição deve especificar tipos para os parâmetros formais que são compatíveis com os tipos dos parâmetros reais *param₁, ..., param_n*, respectivamente.



Associe itens nos quadros abaixo:

- a. variáveis de objeto
- b. método estático (método-de-classe) da classe *TestePonto*
- c. tipo do resultado da chamada *p1.distancia(p2)*
- d. referência ao objeto corrente
- e. parâmetro de método
- f. expressão de criação de objeto (retorna referência ao objeto criado)
- g. método estático da classe *Math*
- h. métodos

- | | |
|---|--------------------------|
| 1. <i>main</i> | 2. <i>sqrt</i> |
| 3. <i>p</i> , do tipo <i>Ponto</i> | 4. <i>new Ponto(0,0)</i> |
| 5. <i>x</i> , <i>y</i> | 6. <i>double</i> |
| 7. <i>move</i> , <i>distancia</i> , <i>main</i> | 8. <i>this</i> |



Indique Falso ou Verdadeiro

- Definição de construtor não precisa especificar tipo do resultado, como no caso de métodos, porque “tipo = nome do construtor”
- Construtor tem sempre mesmo nome da classe em que ocorre
- Uma classe pode ter mais de um construtor
- Método estático funciona como função ou procedimento
- `static void` em método indica que tal método tem comportamento de procedimento: chamada sem especificação de *objeto alvo* e nenhum valor retornado.




Associe itens nos quadros abaixo:

- a. resultado de `new Ponto(0,0)`
- b. valor *default* armazenado em variável de tipo-classe, se nenhuma inicialização for especificada na sua declaração
- c. avaliação de expressão de criação de objeto (`new`)
- d. objetivo de parâmetros de construtores
- e. denominação dada ao objeto denotado por `p1` em `p1.move(3,25)`

- 1. consiste em criar objeto (i.e. suas variáveis), executar corpo do construtor, e finalmente retornar referência ao objeto criado
- 2. `null`
- 3. referência ao objeto criado, e não o objeto propriamente dito
- 4. objeto alvo da chamada
- 5. permitir atribuição de valores iniciais a variáveis de objetos/classes

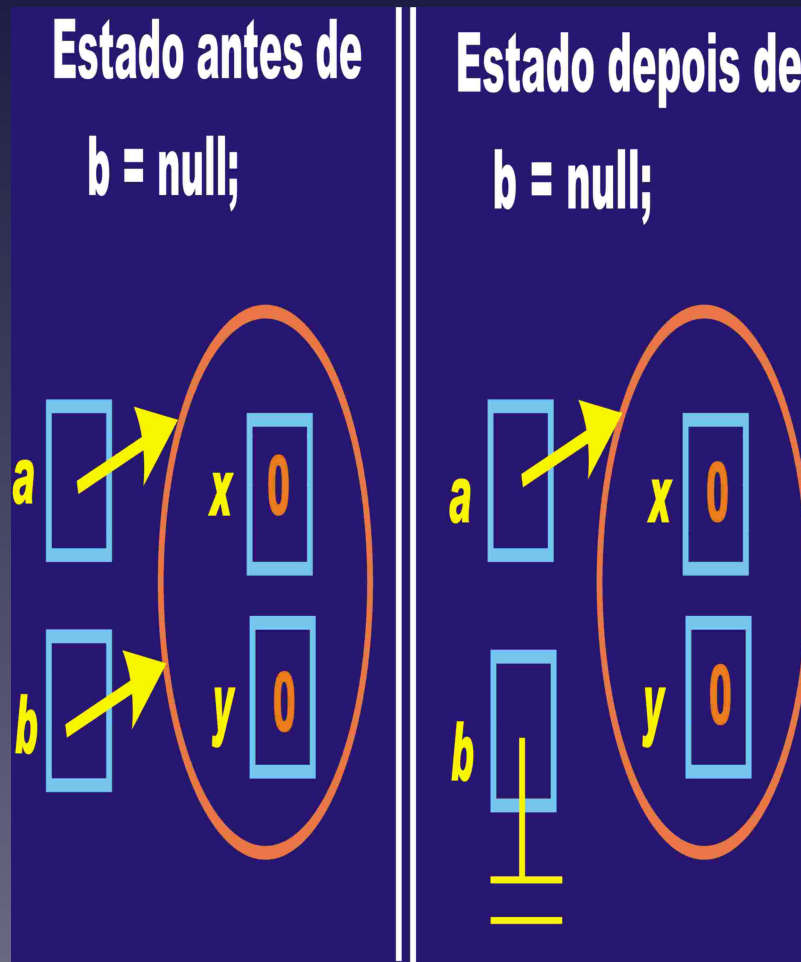


Referências

- Em Java, uma variável de tipo-classe contém uma referência a um objeto (não o objeto propriamente dito)
- Atribuição apenas copia referências 



Atribuição de objetos = atribuição de referências



```
Ponto a =  
    new Ponto(0,0);  
Ponto b;  
  
b = a;  
b = null;
```




Variáveis e métodos de objeto e de classe

Variáveis de classe (variáveis estáticas) armazenam valores (informação) comuns a todos os objetos da classe.

Um método de classe só pode usar (diretamente) variáveis de classe

```
class C
{ static int x; int y;

  static int m1 (int p)
  { x = p+1; }

  int m2 (int p)
  { x = p+2; y = p+1; }
}
```





Variáveis de classe

Variáveis de classe

são comuns a

todos os objetos

```
class C
{ static int x=0;

    public static void main(String[] a)
    { C c1 = new C();
      C c2 = new C();

      c1.x = 1;
      System.out.println(c2.x);
    }
}
```

