nemo

ontology & conceptual modeling research group

Desenvolvimento OO com Java

2 – Tipos, Variáveis e Operadores

Vítor E. Silva Souza

(vitorsouza@inf.ufes.br)

http://www.inf.ufes.br/~vitorsouza

Departamento de Informática

Centro Tecnológico

Universidade Federal do Espírito Santo

Objetivos da parte 2

- Começaremos com os conceitos de Java que são similares à programação estruturada:
  - Tipos primitivos, variáveis, constantes, operadores, expressões, comentários, etc.
  - Veremos que são muito semelhantes à C e C++ e outras linguagens que conhecemos;
- Evoluiremos a discussão para tipos compostos e apresentaremos os conceitos de classe e objeto em Java;
- Estabeleceremos as diferenças de tratamento entre tipos primitivos e objetos.

Tipos primitivos

- Existem 8 tipos de dados básicos para valores inteiros, reais, caracteres e lógicos;
- São chamados primitivos porque:
  - Não são objetos;
  - São armazenados na pilha.
- Java não é OO pura por causa deles;
- Existem por motivos de eficiência;
- São completamente definidos na especificação Java (nome, tamanho, etc.).

Tipos primitivos inteiros

Tipo	Tamanho	Alcance
byte	1 byte	-128 a 127
short	2 bytes	-32.768 a 32.767
int	4 bytes	-2.147.483.648 a 2.147.483.647
long	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

```
/* Cuidados com o alcance do tipo: */
byte b = 127;
System.out.println("b = " + b); // b = 127

b++;
System.out.println("b = " + b); // b = -128
```

Representação de valores literais

```
// Especificando o tipo do literal.
int i = 10;
long l = 10l;
float f = 10.0f;
double d = 10.0;

// Números longos (Java 7).
int bilhao = 1_000_000_000;

// Representação em diferentes bases (binário - Java 7).
int binario = 0b0001_0000;
int octal = 020;
int decimal = 16;
int hexa = 0x10;

System.out.println(binario + ", " + octal + ", " +
decimal + ", " + hexa); // 16, 16, 16, 16
```

Tipos primitivos reais

- Também conhecido como "ponto flutuante".

Tipo	Tamanho	Alcance
float	4 bytes	aprox. ±3.402823 E+38F
double	8 bytes	aprox. ±1.79769313486231 E+308

```
// A separação decimal é feita com "."
float f = 1.0f / 3.0f;
double d = 0.1e1 / 3.; // 0.1e1 = 0,1 x 10^1

// Note a diferença na precisão.
System.out.println(f); // 0.33333334
System.out.println(d); // 0.3333333333333333
```

Precisão de ponto flutuante

- A precisão se refere ao número de casas decimais, não ao tamanho (ex.: 3.4×10^{38});
- Quando a precisão é fundamental, devemos usar a classe `BigDecimal` e não os tipos primitivos.

```
double d = 1.230000875E+2;
System.out.println(d); // 123.0000875

// Converte de double para float.
float f = (float)d;
System.out.println(f); // 123.000084
```

Tipo primitivo caractere

- Para caracteres, Java usa o tipo `char`:
  - Segue o padrão Unicode;
  - Ocupa 2 bytes;
  - Representa 32.768 caracteres diferentes;
  - Permite a construção de software internacionalizado;
  - Depende também do suporte do SO.
- Representação entre aspas simples:
  - 'a', '5', '\t', '\u0061', etc.

Um caractere é um inteiro

- Por sua representação interna ser um número inteiro (código Unicode), um char pode funcionar como um inteiro:

```
char c = 'a';
System.out.println("c: " + c); // c: a

c++;
System.out.println("c: " + c); // c: b

c = (char)(c * 100 / 80);
System.out.println("c: " + c); // c: z

int i = c;
System.out.println("i: " + i); // i: 122
```

Caracteres especiais

Código	Significado
\n	Quebra de linha (newline ou linefeed)
\r	Retorno de carro (carriage return)
\b	Retrocesso (backspace)
\t	Tabulação (horizontal tabulation)
\f	Nova página (form feed)
\'	Aspas simples (apóstrofo)
\"	Aspas
\\	Barra invertida (“\”)
\u233d	Caractere unicode de código 0x233d (hexadecimal)

Cadeias de caractere

- Algumas linguagens definem um tipo primitivo *string* para cadeias de caracteres;
- Java não possui um tipo primitivo *string*;
- Em Java, *strings* são tipos compostos (objetos);
- Veremos mais adiante...

Tipo primitivo lógico (booleano)

- Valores verdadeiro (`true`) ou falso (`false`);
- Não existe equivalência entre valores lógicos e valores inteiros;
  - Em C, 0 é `false` e qualquer outro valor é `true`.

```
boolean b = true;
if (b) System.out.println("OK!"); // OK!

int i = (int)b; // Erro de compilação!
i = 1;
if (i) System.out.println("??"); // Idem!
```

Declaração de variáveis

- Para declarar uma variável, é preciso dar-lhe um nome e determinar seu tipo;
- Opcionalmente, pode atribuir-lhe um valor inicial;
- Você pode declarar várias variáveis de mesmo tipo na mesma linha, separando com vírgula.

```
int i;
float f = 3.141592f;
char a = '\u0061', b = 'b', c, d = 'd';
boolean b1, b2 = true;
```

Tipos de variáveis

- Variáveis que pertencem à classe como um todo;
- Variáveis que pertencem a objetos (instâncias) da classe (atributos definidos para a classe);
- Variáveis locais.

```
public class Variavel {
  public static int c = 10; // De classe.
  public int i;             // De instância.
  public int func() {
     int n = 5;             // Local.
     i = 2 * n;
     return (i + c);
  }
}
```

Escopo de variável

- Variáveis podem ser declaradas em qualquer ponto do programa;
- O escopo define onde a variável é visível (onde podemos ler/atribuir seu valor);
- O escopo de uma variável vai do "{" anterior à sua declaração até o próximo "}".

```
int i = 10;
if (i > 0) {
  int j = 15;
  System.out.println(i + j); // 25
}
j = i + j; // Erro: variável fora de escopo!
```

Nomes de variáveis

- Deve ser iniciado por uma letra, _ ou $;
- Seguido de letras, números, _ ou $;
- Podem ter qualquer tamanho;
- Não podem ser igual a uma palavra reservada;
- Java é *case sensitive*.

Nomes válidos	Nomes inválidos
a1	1a
total	total geral
$_$$	numero-minimo
_especial	tico&teco
Preço	void

Palavras reservadas

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Conversões entre tipos numéricos

- Tipos numéricos podem se misturar em operações, seguindo as seguintes regras:
  - Se um dos operandos for `double` e o outro não, será convertido para `double`;
  - Senão, se um dos operandos for `float` e o outro não, será convertido para `float`;
  - Senão, se um dos operandos for `long` e o outro não, será convertido para `long`;
  - Nos demais casos, os operandos serão sempre convertidos para `int`, caso já não o sejam.

Conversões entre tipos numéricos

```
byte b = 2; short s = 4; int i = 8;
long l = 16; float f = 1.1f; double d = 9.9;

d = s + l + d;
```

long

double

Float é o único que precisa especificar o tipo do valor literal.

```
// i + b resulta em int, convertido pra long;
l = i + b;

// Erro: b + s resulta em int!
s = b + s;
```

Conversões entre tipos numéricos

- Conversões de um tipo menor para um tipo maior ocorrem automaticamente;
- Podemos forçar conversões no sentido contrário, sabendo das possíveis perdas;
- Utilizamos o operador de coerção (*cast*):

```
double x = 9.997;
int nx = (int)x;
System.out.println(nx); // 9

nx = (int)Math.round(x);
System.out.println(nx); // 10
```

Constantes

- Para declarar constantes, basta usar a palavra-chave final:

```
public class Constantes {
  public static void main(String[] args) {
    final double CM_POR_POLEGADA = 2.54;
    CM_POR_POLEGADA = 2.55; // Erro!

    double larguraPol = 13.94;
    double larguraCm = larguraPol * CM_POR_POLEGADA;

    System.out.println(larguraCm);
  }
}
```

Constantes

- A convenção é que seu nome seja escrito com letras maiúsculas;
- É mais comum encontrar constantes como membros de classes ao invés de propriedades de instância ou variáveis locais.

```
public class MinhaClasse {
  public static final int CONSTANTE = 100;
  private static final int CONST_INTERNA = 1;

  /* ... */
}
```

Comentários

- Ignorados pelo compilador;
- Usados pelo programador para melhorar a legibilidade do código;
- Existem três tipos:
  - Comentários de uma linha: // ...;
  - Comentários de múltiplas linhas: /* ... */;
  - Comentários JavaDoc: /** ... */ – utilizados pela ferramenta javadoc para criar uma documentação HTML das classes, atributos e métodos.
  - A ferramenta javadoc vem com o JDK;
  - Mais informações na apostila.

Comentários

```
/** <i>Documentação da classe</i>.
 * @author Fulano da Silva
 * @see java.io.File
 */
public class FileData extends File {
  /** Documentação de atributo. */
  private double tamanho;

  /* Comentário
     de múltiplas linhas. */

  /** Documentação de método. */
  public void excluir() {
     int x = 1; // Comentário de uma linha.
  }
}
```

Operadores

- Símbolos especiais que recebem um ou mais argumentos e produzem um resultado;
- Os operadores Java trabalham somente com tipos primitivos (e wrappers), exceto:
  - ✓ =, == e != podem ser aplicados a objetos;
  - ✓ + e += podem ser aplicados a *strings*.

Operadores aritméticos

Símbolo	Significado	Exemplo
+	Adição	a + b
-	Subtração	a - b
*	Multiplicação	a * b
/	Divisão	a / b
%	Resto da divisão inteira	a % b
-	(Unário) inversão de sinal	-a
+	(Unário) manutenção de sinal	+a
++	(Unário) Incremento	++a ou a++
--	(Unário) Decremento	--a ou a--

Precedência de operadores

- Podemos combinar expressões:
  - ✓`x = a + b * 5 - c / 3 % 10;`
- Atenção à precedência de operadores!
  1) `b * 5;`
  2) `c / 3;`
  3) (resultado de `c / 3`) `% 10;`
  4) Da esquerda para a direita.
- Podemos usar parênteses para modificá-la:
  - ✓`x = (a + b) * (5 - (c / 3)) % 10;`
- Na dúvida, utilize parênteses.

Operadores relacionais

Símbolo	Significado	Exemplo
==	Igual	a == b
!=	Diferente	a != b
>	Maior que	a > b
>=	Maior que ou igual a	a >= b
<	Menor que	a < b
<=	Menor que ou igual a	a <= b

- Observações:
  - Os parâmetros devem ser de tipos compatíveis;
  - Não confunda = (atribuição) com == (igualdade).

Operadores lógicos

Símbolo	Significado	Exemplo
&&	AND (E)	a && b
&	AND sem curto-circuito	a & b
||	OR (OU)	a || b
|	OR sem curto-circuito	a | b
^	XOR (OU exclusivo)	a ^ b
!	NOT (NÃO, inversão de valor, unário)	! a

- Observações:
  - Só operam sobre valores lógicos;
  - Podem ser combinados em expressões grandes.

Curto-circuito

```
int x = 10, y = 15, z = 20;

// (z > y) não é avaliado.
if ((x > y) && (z > y)) { /* ... */ }

// (z == y) não é avaliado.
if ((x == 10) || (z == y)) { /* ... */ }

// Ambos são avaliados.
if ((x > y) & (z > y)) { /* ... */ }

// Ambos são avaliados.
if ((x == 10) | (z == y)) { /* ... */ }
```

Operador de atribuição

- Usado para alterar o valor de uma variável:
  - ✓ `x = 10 * y + 4;`
- Várias atribuições podem ser feitas em conjunto:
  - ✓ `x = y = z = 0;`
- O lado direito da atribuição é sempre calculado primeiro, seu resultado é armazenado na variável do lado esquerdo:
  - ✓ `int x = 5;`
  - ✓ `x = x + 2;`

Operadores bit a bit

- Operam em variáveis inteiras, bit a bit:

Símbolo	Significado	Exemplo
&	AND (E)	a & b
|	OR (OU)	a | b
^	XOR (OU exclusivo)	a ^ b
~	NOT (NÃO, unário)	~ a
>>	Deslocamento de bits para a direita	a >> b
<<	Deslocamento de bits para a esquerda	a << b

Atribuição composta

- Une-se um operador binário com o sinal de atribuição:

Expresão	Equivale a
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Expresão	Equivale a
x &= y	x = x & y
x |= y	x = x | y
x ^= y	x = x ^ y
x >>= y	x = x >> y
x <<= y	x = x << y

Incremento e decremento

- Somar / subtrair 1 de uma variável inteira é tão comum que ganhou um operador só para isso:
  - ✓ ++x e x++ equivalem a x = x + 1;
  - ✓ --x e x-- equivalem a x = x – 1.
- Quando parte de uma expressão maior, a forma prefixada é diferente da pós-fixada:

```
int x = 7;
int y = x++; // y = 7, x = 8.

x = 7;
y = ++x;     // y = 8, x = 8.
```

Operador ternário

- Forma simplificada de uma estrutura `if – else` (que veremos no próximo capítulo);
- Produz um valor de acordo com uma expressão:
  - ✓ `<expressão> ? <valor 1> : <valor 2>`
  - ✓ Se `<expressão>` for `true`, o resultado é `<valor 1>`, do contrário o resultado é `<valor 2>`.

```
int x = 7;

int y = (x < 10) ? x * 2 : x / 2;
System.out.println("y = " + y); // y = 14
```

Precedência de operadores

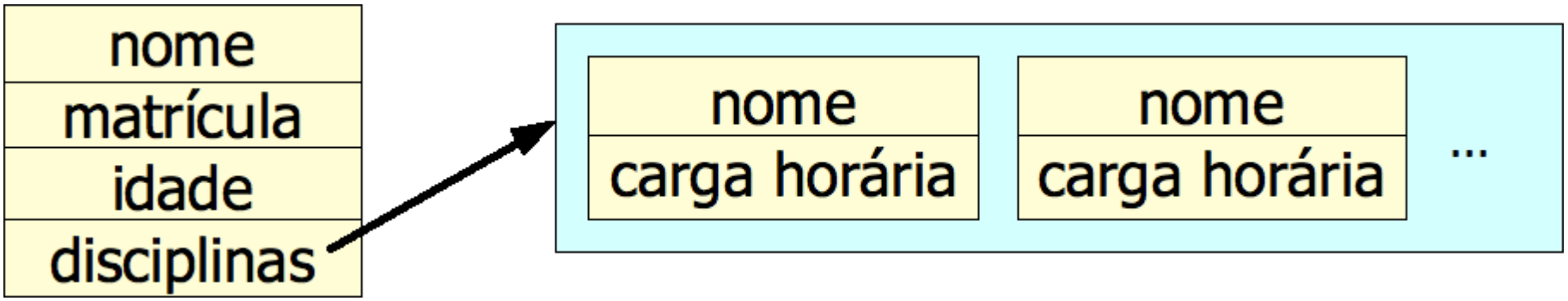
- As expressões são avaliadas segundo uma ordem de precedência dos operadores:

Ordem	Operadores
1	. [] ()
2	++ -- ~ instanceof new – (unário)
3	* / %
4	+ -
5	>> << >>>
6	> < >= <=
7	== !=

Ordem	Operadores
8	&
9	^
10	|
11	&&
12	||
13	?:
14	= += -= *= /= ...
15	,

Tipos compostos

- Tipos primitivos armazenam somente informações muito simples;
- Por exemplo, se quisermos armazenar informações sobre alunos:
  - nome, matrícula, idade, disciplinas cursadas, etc.
- É necessário criar estruturas compostas por tipos primitivos e outras estruturas.

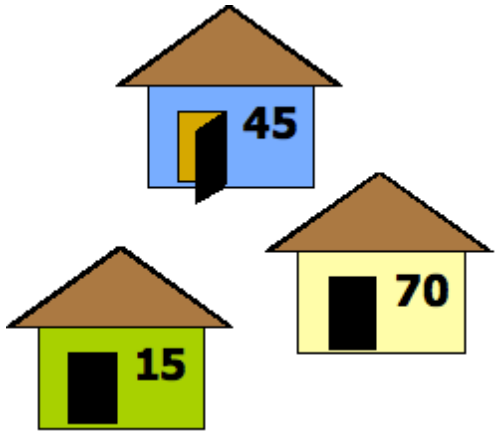


Registros e classes

- Linguagens estruturadas usam o conceito de registro:
  - Agregado de dados de tipos heterogêneos, que identifica seus elementos individuais por nome.
- Linguagens orientadas a objeto usam o conceito de classes e objetos:
  - Objetos são estruturas de dados compostas que contém dados e funções (métodos);
  - Pode armazenar internamente dados de tipos primitivos ou outros objetos;
  - Objetos são instâncias de classes.

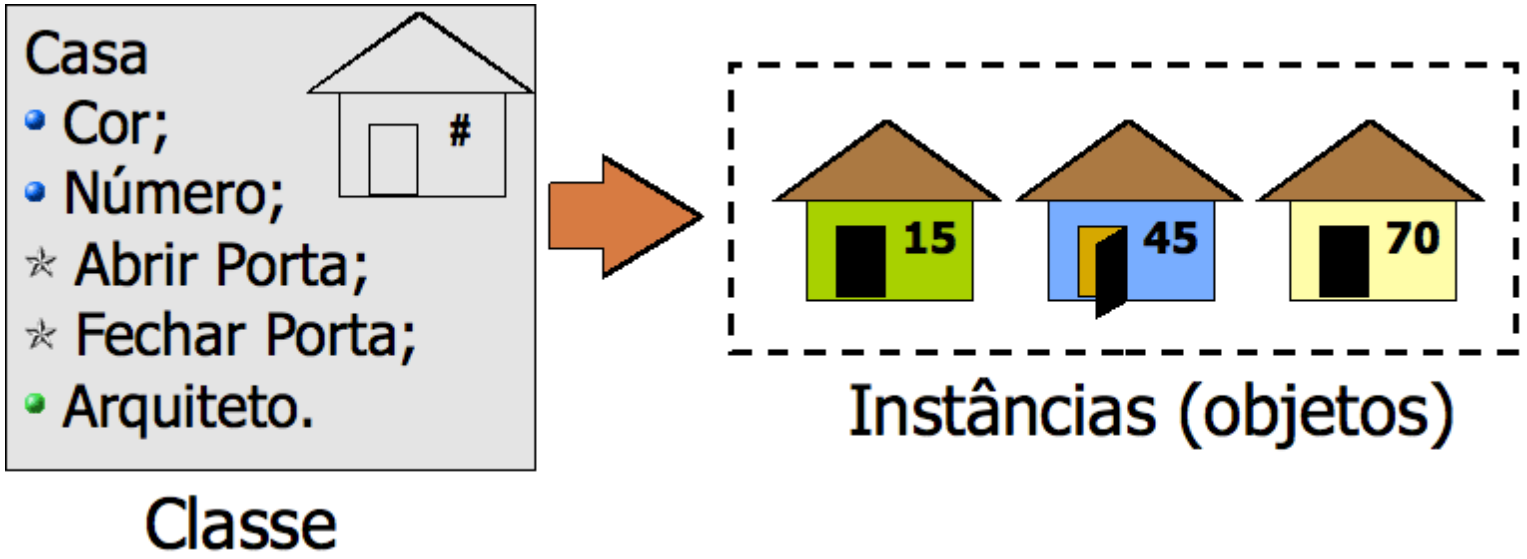
Objetos

- Um objeto é uma entidade que incorpora uma abstração relevante no contexto de uma aplicação;
- Podem ser coisas abstratas (ex.: uma reserva de passagem aérea, um pagamento) ou concretas (ex.: um documento, um produto);
- Possui três características principais:
  - Estado (estrutura de dados);
  - Comportamento (métodos);
  - Identidade (cada objeto é único).



Classes

- Uma classe descreve um conjunto de objetos com as mesmas propriedades, o mesmo comportamento, os mesmos relacionamentos com outros objetos e a mesma semântica;
- Parecido com o conceito de tipo.



Instâncias de classes

- Objeto = Instância de classe;
- Paradigma OO norteia o desenvolvimento por meio de classificação de objetos:
  - Modelamos classes, e não objetos;
  - Objetos são entidades reais – executam algum papel no sistema;
  - Classes são abstrações – capturam a estrutura e comportamento comum a um conjunto de objetos.

Tipos compostos em Java

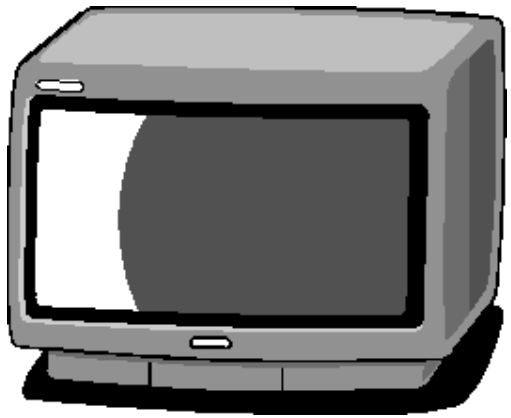
- Java é OO e, portanto, não possui registros, mas sim classes e objetos;
- Exceto os tipos primitivos já estudados, tudo em Java é um objeto, inclusive vetores e *strings*;
- Primeiro escrevemos as classes, em seguida criamos objetos, instâncias das classes;
- Em Java, comparado com C++ por exemplo, tudo isso é bastante simplificado.

Uma classe Java

```
public class Coordenadas {
  public int x;
  public int y;
  public int z;

  public static void main(String[] args) {
     Coordenadas coord = new Coordenadas();
     coord.x = 10;
     coord.y = 15;
     coord.z = 18;
  }
}
```

Manipulação de objetos em Java

- Em Java trabalhamos com referências para objetos, ao contrário de C++ (manipulação direta ou ponteiros);
- Analogia:
  - A TV é o objeto;
  - O controle é a referência;
  - Você só carrega a referência;
  - A referência pode existir sem o objeto.



Manipulação de objetos em Java

- Uma *string* é uma classe já definida pela API;
- Se quisermos usar uma *string*, precisamos de uma referência, ou seja, uma variável *string*:

```
String s;
```

- Agora possuímos uma referência. Não podemos usá-la até que o objeto real seja criado:

```
// s = "Olá Mundo!";
s = new String("Olá Mundo!");
System.out.println(s); // Olá Mundo!
```

Atribuição de valores

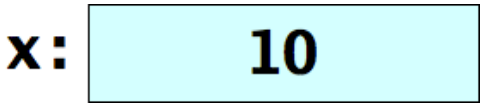
- Quando realizamos uma atribuição:

```
x = y;
```

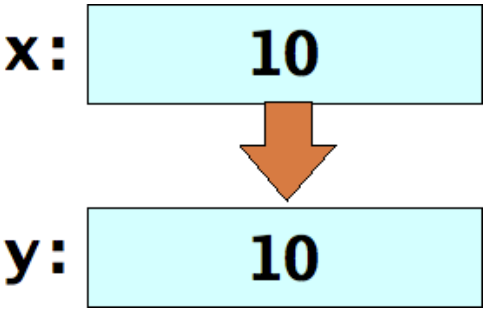
- Java faz a cópia do valor da variável da direita para a variável da esquerda;
  - Para tipos primitivos, isso significa que alterações em x não implicam alterações em y;
  - Para objetos, como o que é copiado é a referência para o mesmo objeto, alterações no objeto que x referencia altera o objeto que y referencia, pois é o mesmo objeto!

Atribuição de valores primitivos

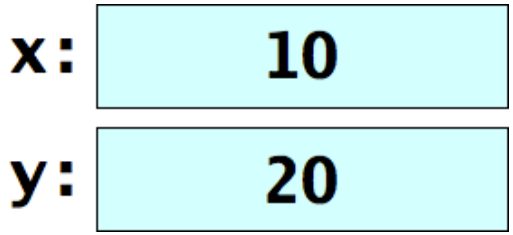
```
int x = 10;
```




```
int y = x;
```

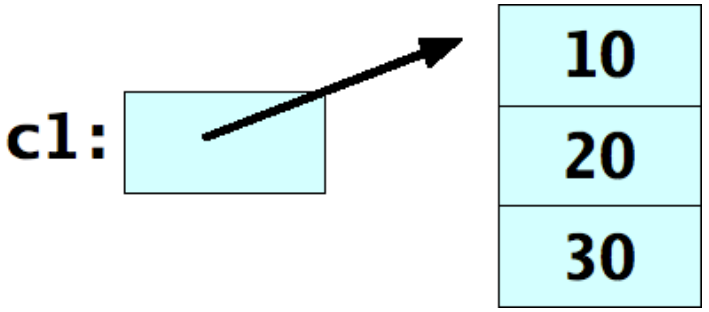



```
y = 20;
```

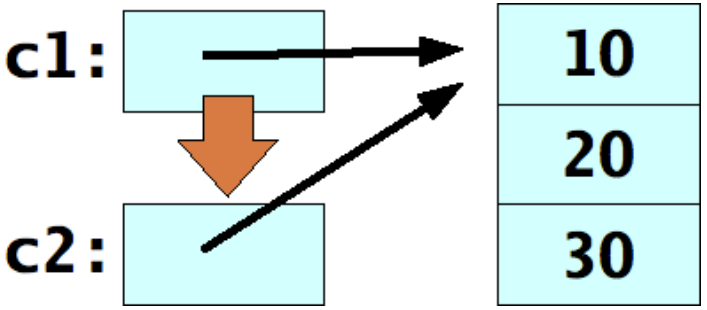


Atribuição de objetos

```
Coordenada c1;
c1 = new Coordenada();
c1.x = 10;
c1.y = 20;
c1.z = 30;
```

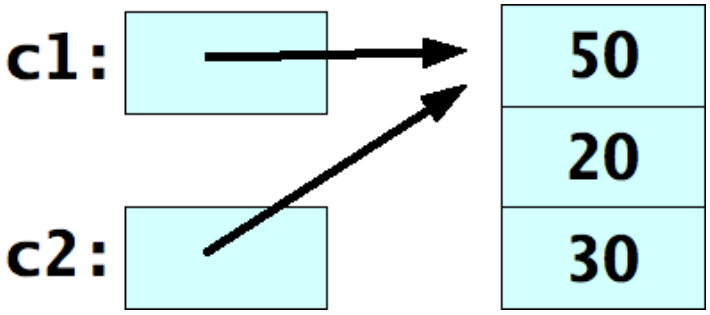



```
Coordenada c2;

// Erro comum:
// c2 = new Coordenada();

c2 = c1;
```



Atribuição de objetos

```
c2.x = 50;
```



Tenha sempre em mente a diferença entre um tipo primitivo e um objeto (referência).

Strings em Java

- Java não possui tipo primitivo para cadeia de caracteres, mas existe a classe `String`;
- Esta classe tem tratamento especial:
  - Construção facilitada usando literais ("");
  - Operador de concatenação;
  - Conversão automática de tipos primitivos e objetos para `String`.

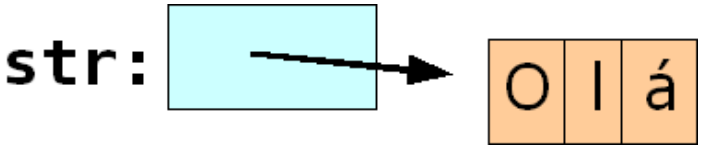
Strings em Java

```
// Equivale a new String("Olá, mundo!").
String mensagem = "Olá, mundo!";

// String vazia (tamanho 0).
String str = "";

// Concatenação.
str = "A mensagem é: " + mensagem;

// Conversão (c1 é aquele objeto Coordenada).
int i = 10; float f = 3.14f;
str = "i = " + i + ", f = " + f;
str += ", c1 = " + c1;
```

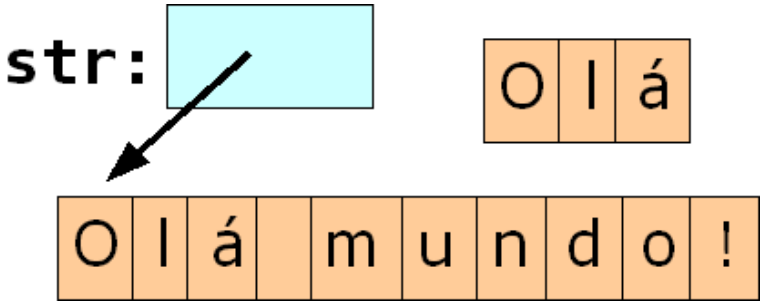
Strings são imutáveis

- Não podemos mudar o valor de um caractere da *string*. Podemos somente criar outra *string*.

```
String str = "Olá";
```




```
str += " mundo!";
```



Uma nova string é criada e a outra é abandonada para o coletor de lixo.

Lembre-se sempre: *strings* são imutáveis!

http://nemo.inf.ufes.br/