Aula 04 – Variáveis e operadores

Baseado nos slides oficiais do livro Java – Como programar – Deitel e Deitel – 10ª edição

Tipos primitivos e referências

- Tipos em java são divididos em 2 categorias: primitivos e de referência
- Os tipos primitivos são:
  - Caractere: **char**
  - Lógico: **boolean**
  - Inteiro: byte, short, **int**, long
  - Real: float e **double**.
- Tipos primitivos têm seus valores copiados quando há uma atribuição ou passagem de parâmetros, enquanto os demais passam apenas a referência
- Todos os outros tipos são de referência, ou seja, eles "apontam" para um objeto ou são null

Declaração de variáveis e constantes

- Variáveis são declaradas no mesmo estilo do C/C++

```
int number1;
char opção1, opção2;
String nome="Sebastião",sobrenome="Alves",login;
```

- Constantes são declaradas usando a palavra reservada `final` antes do tipo

```
final int DIAS=30;
```

- Toda variável usada em um programa Java **deve** ser inicializada em algum momento
- Constantes não precisam ser inicializadas na declaração, mas uma vez atribuídas não podem ter seu valor alterado

Operadores aritméticos

- Semelhantes aos do C/C++
- Atribuições são feitas com o sinal =
- Operações: +, -, *, /, %
  - São operadores binários, resolvidos dois a dois
  - Também têm regras de precedência como nas expressões numéricas comuns
  - Para compor expressões ou dar ordem diferentes da precedência, usa-se parênteses

Java operation	Operator	Algebraic expression	Java expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Fig. 2.11 | Arithmetic operators.

Operator(s)	Operation(s)	Order of evaluation (precedence)
*	Multiplication	Evaluated first. If there are several operators of this type, they're evaluated from *left to right*.
/	Division	
%	Remainder	
+	Addition	Evaluated next. If there are several operators of this type, they're evaluated from *left to right*.
-	Subtraction	
=	Assignment	Evaluated last.

Fig. 2.12 | Precedence of arithmetic operators.

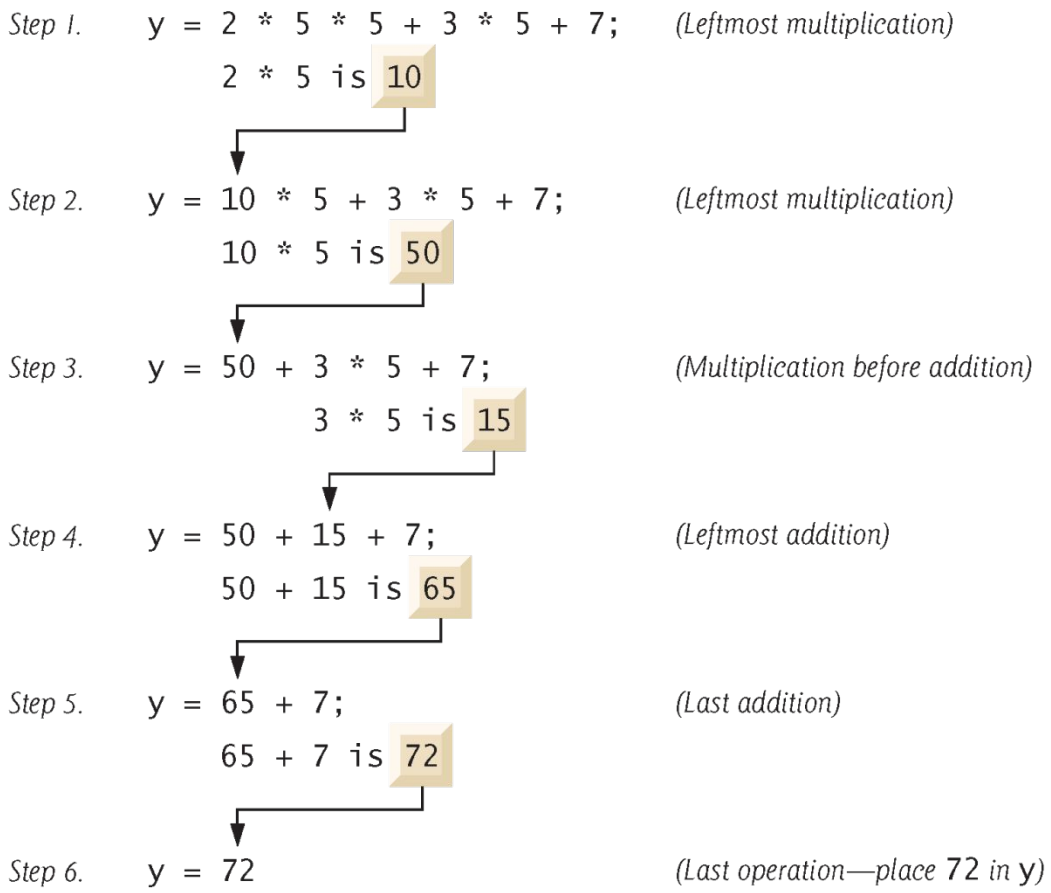


Fig. 2.13 | Order in which a second-degree polynomial is evaluated.

Conversões e promoções

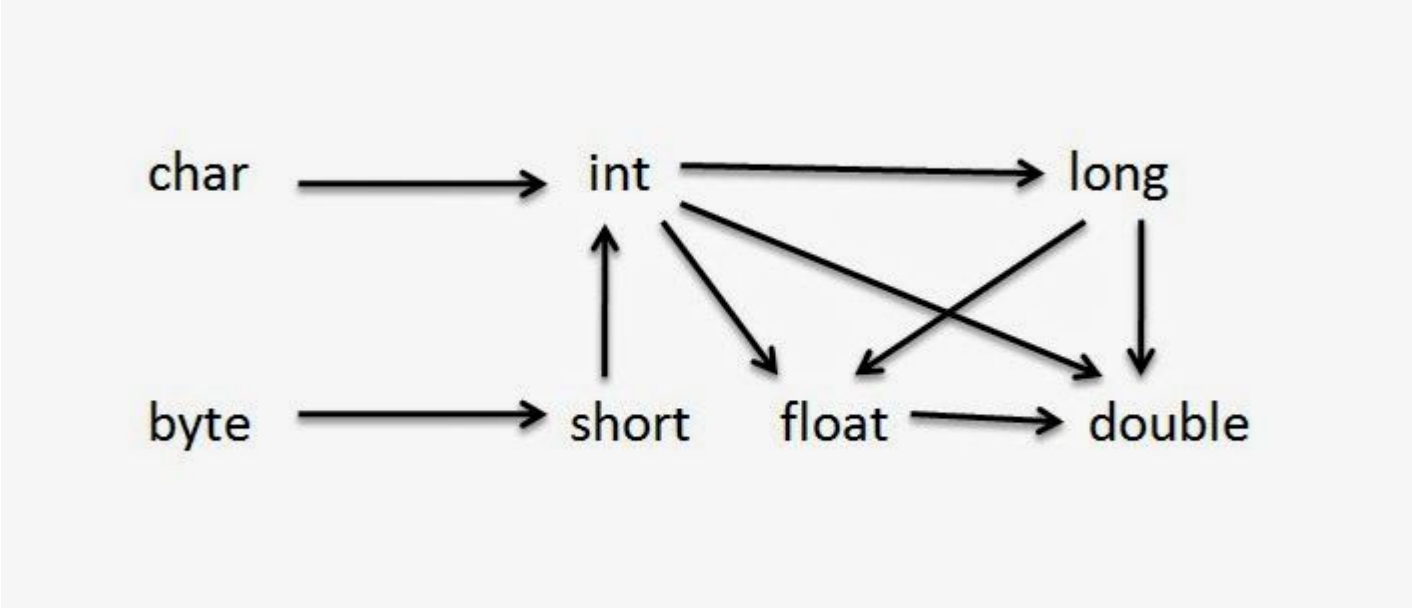
- Tipos inteiros e reais de diferentes tamanhos podem se misturar em expressões
- Se a capacidade de representação é menor, o tipo menor é automaticamente promovido ao maior

```
short s = -1;
int i = s;
```

- Se a capacidade de representação é maior, é preciso forçar a conversão

```
int i = -1;
short s = (short) i;
```

Tabela de promoção automática



Tipos primitivos e classes wrapper

- Cada tipo primitivo tem uma classe Wrapper em java equivalente
  - Integer → int, Double → double, …
- Tipos primitivos e classes wrapper equivalentes podem ser atribuídos sem usar conversão. Isso é chamado Boxing ou Unboxing

```
int i = 30;
Integer boxed =i + 5;
int unboxed = boxed +10;
```

Conversão String / tipos primitivos

- Em várias situações, os dados são obtidos como String e devem ser convertidos para outros tipos
- As classes wrappers tem funções estáticas para fazer essa conversão

```
int   myNewInt  = Integer.parseInt(myString);
long  myNewLong   = Long.parseLong(myString);
double myNewDouble = Double.parseDouble(myString);
boolean myNewBoolean = Boolean.parseBoolean(myString);
```

```
int idade = Integer.parseInt(JOptionPane.showInputDialog("Digite a sua idade em anos"));
```

Igualdade e operadores relacionais

- Semelhantes ao C/C++: ==, !=, >, <, >=, <=
- Diferença: não há mapeamento entre inteiro e boolean.
- Toda comparação resulta em um booleano true ou false
- Toda condição testada deve ser booleana
  - if (x = 10) // Erro, expressão não retorna booleano
  - if (x) // Erro, x é uma variável não uma expressão

Algebraic operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.14 | Equality and relational operators.

```
1  // Fig. 2.15: Comparison.java
2  // Compare integers using if statements, relational operators
3  // and equality operators.
4  import java.util.Scanner; // program uses class Scanner
5
6  public class Comparison
7  {
8     // main method begins execution of Java application
9     public static void main(String[] args)
10    {
11       // create Scanner to obtain input from command line
12       Scanner input = new Scanner(System.in);
13
14       int number1; // first number to compare
15       int number2; // second number to compare
16
17       System.out.print("Enter first integer: "); // prompt
18       number1 = input.nextInt(); // read first number from user
19
20       System.out.print("Enter second integer: "); // prompt
21       number2 = input.nextInt(); // read second number from user
22
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 1 of 3.)

```
23          if (number1 == number2)
24             System.out.printf("%d == %d%n", number1, number2);
25
26          if (number1 != number2)
27             System.out.printf("%d != %d%n", number1, number2);
28
29          if (number1 < number2)
30             System.out.printf("%d < %d%n", number1, number2);
31
32          if (number1 > number2)
33             System.out.printf("%d > %d%n", number1, number2);
34
35          if (number1 <= number2)
36             System.out.printf("%d <= %d%n", number1, number2);
37
38          if (number1 >= number2)
39             System.out.printf("%d >= %d%n", number1, number2);
40       } // end method main
41    } // end class Comparison
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 2 of 3.)

```
Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777
```

```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 3 of 3.)

Operadores de atribuição compostos

- Substituem expressões do tipo
  variável = variável operador expressão;
  Por outras mais curtas com os operadores aritm
  variável operador= expressão;
- Exemplo

```
c = c + 3;
```

  Pode ser escrito como

```
c += 3;
```

- Lembrando que toda a expressão do lado direito é resolvida antes de se fazer a atribuição.

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

Fig. 4.13 | Arithmetic compound assignment operators.

Operadores de incremento e decremento

- Operador de incremento unário ++ soma 1 ao operando enquanto o -- subtrai 1 do operando
- Pode ser prefixo se vier antes da variável ou posfixo se vier depois
- Os operadores prefixos são executados antes de serem usados na expressão, enquanto os posfixos são primeiro usados na expressão e depois executados

Operator	Operator name	Sample expression	Explanation
++	prefix increment	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	postfix increment	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	prefix decrement	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	postfix decrement	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Fig. 4.14 | Increment and decrement operators.

Operador ternário

Operador Condicional (?:)

- Usa três operandos:
  - Uma condição que retorna true ou false
  - Uma expressão de retorno caso seja verdadeiro
  - Uma expressão de retorno caso seja falso
- Exemplo:

```
System.out.println(nota >= 7.0 ? "Passou" : "Não passou");
```

- O operador avalia se a nota foi maior ou igual a 7.0. Caso seja verdade a String “Passou” é exibida. Caso contrário “Não passou” é mostrado.
-
-

Operators						Associativity	Type
`++`	`--`					right to left	unary postfix
`++`	`--`	`+`	`-`	(*type*)		right to left	unary prefix
`*`	`/`	`%`				left to right	multiplicative
`+`	`-`					left to right	additive
`<`	`<=`	`>`	`>=`			left to right	relational
`==`	`!=`					left to right	equality
`?:`						right to left	conditional
`=`	`+=`	`-=`	`*=`	`/=`	`%=`	right to left	assignment

Fig. 4.16 | Precedence and associativity of the operators discussed so far.

Estruturas de controle - Condicionais

- Dois tipos de estrutura de seleção condicional
- `if`: realiza um teste se a condição é verdadeira e executa uma ação
  - `else`: define um bloco que executa uma ação caso a condição seja falsa
  - Cada else deve casar com um if ainda não fechado
- `switch`: realiza um teste baseado no resultado de uma expressão em relação a um conjunto de valores constantes
  - case: testa se a expressão é igual ao valor constante definido. Caso sim executa até encontrar um break;
- Tanto o if/else como o switch/case têm a sintaxe semelhante ao C/C++

Estruturas de controle - Repetição

- Três tipos de estrutura de repetição (ou iteração ou laços) são implementados no Java
- `while` e `for`: testam uma condição no início e executam caso seja verdadeiro, retornando a testar após cada iteração. Podem não executar se a condição inicial já for falsa.
  - O *for* possui uma seção de inicialização executada antes do teste e uma de incremento executada após cada iteração
- `do...while`: executa pelo menos uma iteração e repete se a condição ao final for verdadeira.
  - Única estrutura de controle com ; após o comando
- `if`, `else`, `switch`, `while`, `do` e `for` são palavras reservadas da linguagem e não podem ser usadas como nome de variaveis, classes, objetos ou métodos, e têm sintaxe semelhante ao C/C++

Blocos

- As estrutura de controle condicionais e laços executam o comando imediatamente posterior a elas caso a condição seja atendida.
- Caso seja necessário executar mais de um comando é necessário a criação de blocos delimitados com { e }
- Exemplo:

```
if (nota >= 7.0)
   System.out.println("Passou")
else {
   System.out.println("Não passou");
   System.out.println("Nota tem que ser >= 7.0");
}
```

Aninhamento de comandos

- Os condicionais e laços podem ser aninhados em níveis
  - Cada comando aninhado só é acessado se as condições dos blocos mais externos sejam atendidas
- Exemplo:

```
if (nota >= 7.0)
   System.out.println("Passou por média")
else {
   System.out.println("Não passou por média");
   if ((quartaProva+nota)/2 >=6.0)
      System.out.println("Passou na 4ª prova");
   else
      System.out.println("Não passou na 4ª
   prova");
 }
```