


Variáveis e expressões – Parte 2

Prof. Bruno Nogueira

Constantes

- ▶ O valor de variáveis, como o nome sugere, pode variar
- ▶ Em Java, existem também as **constantes**, ou literais
 - ▶ Um valor 2 em uma expressão `int a = 2 * b;` nunca muda
 - ▶ Constantes numéricas não usam pontuação para indicar milhares
 - ▶ Constantes numéricas de ponto flutuante podem ser escritas na forma de notação científica
 - ▶ $865000000.0 = 8.65 \times 10^8$
 - ▶ Em Java: **8.65e8**
 - O número antes do **e** pode conter parte decimal
 - O número após o **e** não pode conter parte decimal

Constantes

- ▶ Em Java, existe ainda a possibilidade de declarar variáveis cujos valores não podem ser modificados
- ▶ Sintaxe:

public static final tipo NOME_VARIÁVEL = Constante;

Qualquer tipo, de classe ou primitivo: String, int, float, double, ...

Entenderemos o significado de cada palavra mais adiante. Juntas, dizem: esse valor é imutável.

- ▶ Por exemplo, para declarar uma variável que represente o valor de Pi como 3.14159

```
public static final double PI = 3.14159;
```

- ▶ Por convenção, o nome de constantes é sempre escrito em letras maiúsculas

Constantes

► Outros exemplos

```
public static final int PONTOS_VITORIA = 3;  
public static final double TAXA_JUROS = 6.99;  
public static final String NOME_DISCIPLINA = "ALGPROG";  
public static final char CONCEITO = 'A';
```

A classe String

- ▶ Conforme já vimos, Java não tem um tipo primitivo para armazenar valores de cadeias de caracteres
- ▶ Para isso, a linguagem nos fornece o tipo classe String

- ▶ Valores são sempre declarados entre aspas

```
String fraseInicial = "Olá! Seja bem vindo!";  
System.out.println(fraseInicial);
```

- ▶ Strings podem conter tantos caracteres quantos forem necessários, inclusive nenhum

```
String vazia = "";
```

Concatenação de Strings

- ▶ É possível juntar duas strings em uma única cadeia de caracteres: concatenação de strings
- ▶ Operador de concatenação +
- ▶ Exemplo:

```
String saudacao, sentenca;  
saudacao = "Olá!";  
sentenca = saudacao + " Seja bem vindo!";  
System.out.println(sentenca);  
System.out.println(sentenca + " É um prazer recebê-lo!");
```

Métodos da classe String

- ▶ String é um tipo classe
 - ▶ Uma variável String forma um objeto da classe String
 - ▶ Como já vimos, objetos têm métodos, além de dados
 - ▶ Métodos definem o comportamento dos objetos
 - ▶ Valores dos dados (dados = variáveis) definem o estado de um objeto
 - ▶ Classe String fornecem algumas ações interessantes para manipulação de cadeias de caracteres
 - ▶ Para acionar estas ações, basta invocar os métodos correspondentes

Métodos da classe String

▶ Método length()

- ▶ Retorna o tamanho da cadeia de caracteres
- ▶ Em outras palavras, retorna o número de caracteres armazenados
- ▶ Exemplos

```
String ola = "Olá!";
```

```
int numCarcteres = ola.length();
```



Armazenará valor 4

```
int numCaracteres2 = "Olá!".length();
```



Também armazenará valor 4

```
int numCaracteres3 = "Programar é legal!".length();
```



Armazenará valor 18

Métodos da classe String

- ▶ Para diversos métodos, é utilizada a contagem de posições dentro da String
 - ▶ Posições na String começam a ser contados em 0, não em 1
 - ▶ Exemplo:
`String frase = "Java é legal!";`

Índices	0	1	2	3	4	5	6	7	8	9	10	11	12
	J	a	v	a		é		l	e	g	a	l	!

Métodos da classe String

▶ Método charAt()

- ▶ Retorna um caractere em um determinado índice
- ▶ Sintaxe

```
variavel.charAt(indice);
```

▶ Exemplo

```
String frase = "Java é legal!";  
char caractere = frase.charAt(1);
```



Armazenará valor 'a'

```
char caractere2 = "Onde está Wally?".charAt(2);
```



Armazenará valor 'd'

Métodos da classe String

▶ Método substring()

- ▶ Retorna um pedaço de uma String, começando e terminando em índices específicos

- ▶ Sintaxe:

```
variavel.substring(comeco, fim);
```

Fim é opcional.

Se não for especificado,
vai até o fim da cadeia

- ▶ Exemplo:

```
String frase = "Java é legal!";
```

```
String pedaco1 = frase.substring(0,4);
```



Armazenará valor "Java". Caractere do índice fim não é incluído!

```
String pedaco2 = frase.substring(5);
```



Armazenará valor "é legal!"

Métodos da classe String

▶ Método indexOf()

- ▶ Obtem o índice no qual começa a ocorrência de uma determinada palavra

- ▶ Sintaxe:

`variavel.indexOf(valor);`

- ▶ Exemplo:

```
String frase = "Java é legal!";  
int indice = frase.indexOf("legal");
```



Armazenará valor 7

Métodos da classe String

- ▶ Existem outros métodos, que serão explorados em momento oportuno

<code>charAt(<i>Index</i>)</code>	Returns the character at <i>Index</i> in this string. Index numbers begin at 0.
<code>compareTo(<i>A_String</i>)</code>	Compares this string with <i>A_String</i> to see which string comes first in the lexicographic ordering. (Lexicographic ordering is the same as alphabetical ordering when both strings are either all uppercase letters or all lowercase letters.) Returns a negative integer if this string is first, returns zero if the two strings are equal, and returns a positive integer if <i>A_String</i> is first.
<code>concat(<i>A_String</i>)</code>	Returns a new string having the same characters as this string concatenated with the characters in <i>A_String</i> . You can use the + operator instead of <code>concat</code> .
<code>equals(<i>Other_String</i>)</code>	Returns true if this string and <i>Other_String</i> are equal. Otherwise, returns false.
<code>equalsIgnoreCase(<i>Other_String</i>)</code>	Behaves like the method <code>equals</code> , but considers uppercase and lowercase versions of a letter to be the same.
<code>indexOf(<i>A_String</i>)</code>	Returns the index of the first occurrence of the substring <i>A_String</i> within this string. Returns -1 if <i>A_String</i> is not found. Index numbers begin at 0.
<code>lastIndexOf(<i>A_String</i>)</code>	Returns the index of the last occurrence of the substring <i>A_String</i> within this string. Returns -1 if <i>A_String</i> is not found. Index numbers begin at 0.
<code>length()</code>	Returns the length of this string.
<code>toLowerCase()</code>	Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase.
<code>toUpperCase()</code>	Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase.
<code>replace(<i>OldChar</i>, <i>NewChar</i>)</code>	Returns a new string having the same characters as this string, but with each occurrence of <i>OldChar</i> replaced by <i>NewChar</i> .
<code>substring(<i>Start</i>)</code>	Returns a new string having the same characters as the substring that begins at index <i>Start</i> of this string through to the end of the string. Index numbers begin at 0.
<code>substring(<i>Start</i>, <i>End</i>)</code>	Returns a new string having the same characters as the substring that begins at index <i>Start</i> of this string through, but not including, index <i>End</i> of the string. Index numbers begin at 0.
<code>trim()</code>	Returns a new string having the same characters as this string, but with leading and trailing whitespace removed.

Caracteres de escape em Strings

- ▶ Alguns caracteres são especiais e necessitam de um tratamento diferenciado em sequencias Strings
 - ▶ Por exemplo, as aspas delimitam Strings. Mas e se quisermos colocar aspas dentro das Strings a serem impressas?

~~String frase = “Java é legal “demais da conta” !”;~~

Erro de compilação

String frase = “Java é legal \“demais da conta\” !”;

Solução correta:
adicionar a barra
antes da aspa

Caracteres de escape em Strings

- ▶ A barra é o que chamamos de caractere de escape
 - ▶ Caractere numa cadeia de caracteres que modifica o significado do seu sucessor
 - ▶ Utilizado para imprimir caracteres reservados ou efetuar formatações na cadeia de caracteres

Caracteres	Significado
\”	Adiciona aspas duplas
\’	Adiciona aspas simples
\\	Adiciona uma barra
\n	Gera uma nova linha
\t	Gera uma tabulação

Caracteres de escape em Strings

▶ Exemplos:

▶ Comando:

```
System.out.println("Instalado em C:\\Java");
```

▶ Saída:

```
Instalado em C:\Java
```

▶ Comando:

```
System.out.println("Java\\né\\n\\tlegal!");
```

▶ Saída:

```
Java
```

```
é
```

```
    legal!
```


Comandos de entrada e saída

- ▶ **System.out.print() x System.out.println()**
 - ▶ Ambos imprimem uma mensagem na saída
 - ▶ Diferença é que println() cria uma nova linha após imprimir a mensagem na tela, enquanto print() não o faz
 - ▶ Exemplos:

print()	println()
<pre>System.out.print("Um, "); System.out.print("dois, "); System.out.print("três.");</pre>	<pre>System.out.println("Um, "); System.out.println("dois, "); System.out.println("três.");</pre>
Um, dois, três.	Um dois, três.

Comandos de entrada e saída

- ▶ Ainda, existe um terceiro método para imprimir saídas: o método `printf()`
 - ▶ Muito similar ao `printf()` de outras linguagens de programação
 - ▶ Dá opções de formatação das saídas, como o número de casas decimais a serem impressas e o número de espaços a serem ocupados pela mensagem
 - ▶ Exemplos:

```
double preco = 19.5;  
System.out.println("Preço com println: " + preco);  
System.out.printf("Preço com printf: %6.2f", preco);
```
 - ▶ Saída:

```
Preço com println: 19.5  
Preço com printf: 19.50
```

Comandos de entrada e saída

Format Specifier	Type of Output	Examples
%c	Character	A single character: %c
		A single character in a field of two spaces: %2c
%d	Decimal integer number	An integer: %d
		An integer in a field of 5 spaces: %5d
%f	Floating-point number	A floating-point number: %f
		A floating-point number with 2 digits after the decimal: %1.2f
		A floating-point number with 2 digits after the decimal in a field of 6 spaces: %6.2f
%e	Exponential floating-point number	A floating-point number in exponential format: %e
%s	String	A string formatted to a field of 10 spaces: %10s

Comandos de entrada e saída

```
1  import java.util.Scanner;
2  public class ImpressaoFormatada
3  {
4      public static void main (String [ ] args)
5      {
6          Scanner teclado = new Scanner(System.in);
7          double preco    = 19.5;
8          int quantidade   = 2;
9          String item      = "Widgets";
10
11          System.out.printf("%10s vendidos: %4d a %5.2f. Total = %1.2f",
12                           item, quantidade, preco, quantidade * preco);
13      }
14 }
```

Comandos de entrada e saída

- ▶ Métodos da classe Scanner nos dão opções para leitura de dados a partir da entrada (teclado)
 - ▶ Já vimos, em exemplos, o `nextInt()` e o `nextDouble()`, que leem, respectivamente, um inteiro (`int`) e um ponto flutuante `double` (`double`) a partir do teclado
 - ▶ Considera que valores estão em uma linha, separados por espaço
 - ▶ Existem métodos similares:
 - ▶ `nextFloat()`: retorna um valor do tipo `float`
 - ▶ `nextLong()`: retorna um valor do tipo `long`
 - ▶ `nextByte()`: retorna um valor do tipo `byte`
 - ▶ `nextShort()`: retorna um valor do tipo `short`
 - ▶ `nextBoolean()`: retorna um valor do tipo `boolean` (`true` ou `false`)

Comandos de entrada e saída

- ▶ Existem, ainda, dois métodos para a leitura de Strings: `next()` e `nextLine()`
 - ▶ `next()` é similar aos métodos anteriores, lendo uma cadeia de caracteres até encontrar um delimitador (espaço, por padrão)
 - ▶ `nextLine()` por sua vez, lê toda a linha digitada pelo usuário, isso é, todos os caracteres até encontrar o fim de linha (`\n`)

```
1 import java.util.Scanner;
2 public class LeituraDados
3 {
4     public static void main (String [ ] args)
5     {
6         Scanner teclado = new Scanner(System.in);
7         int primInteiro, segInteiro;
8         double primDouble, segDouble;
9         String primString, segString;
10
11         System.out.print("Digite dois número inteiros");
12         System.out.println(" separados por espaços:");
13         primInteiro = teclado.nextInt();
14         segInteiro = teclado.nextInt();
15         System.out.println("Números digitados: " + primInteiro + " e " + segInteiro);
16
17         System.out.println("Agora, digite dois números com parte decimal: ");
18         primDouble = teclado.nextDouble();
19         segDouble = teclado.nextDouble();
20         System.out.println("Números digitados: " + primDouble + " e " + segDouble);
21
22         System.out.println("Agora, digite duas palavras: ");
23         primString = teclado.next();
24         segString = teclado.next();
25         System.out.println("Palavras digitadas: " + primString + " e " + segString);
26
27         System.out.println("Agora, digite uma linha de texto: ");
28         primString = teclado.nextLine();
29         System.out.println("Texto digitado: ");
30
31     }
32 }
```

```

1  import java.util.Scanner;
2  public class LeituraDados
3  {
4      public static void main (String [ ] args)
5      {
6          Scanner teclado = new Scanner(System.in);
7          int primInteiro, segInteiro;
8          double primDouble, segDouble;
9          String primString, segString;
10
11         System.out.print("Digite dois número inteiros");
12         System.out.println(" separados por espaços:");
13         primInteiro = teclado.nextInt();
14         segInteiro = teclado.nextInt();
15         System.out.println("Números digitados: " + primInteiro + " e " + segInteiro);
16
17         System.out.println("Agora, digite dois números com parte decimal: ");
18         primDouble = teclado.nextDouble();
19         segDouble = teclado.nextDouble();
20         System.out.println("Números digitados: " + primDouble + " e " + segDouble);
21
22         System.out.println("Agora, digite duas palavras: ");
23         primString = teclado.next();
24         segString = teclado.next();
25         System.out.println("Palavras digitadas: " + primString + " e " + segString);
26
27         System.out.println("Agora, digite uma linha de texto: ");
28         primString = teclado.nextLine();
29         System.out.println("Texto digitado: ");
30
31     }
32 }

```

Lê as duas
palavras, mas ainda
sobra o “\n”

Lê somente o “\n”
que sobrou


```
1 import java.util.Scanner;
2 public class LeituraDados
3 {
4     public static void main (String [ ] args)
5     {
6         Scanner teclado = new Scanner(System.in);
7         int primInteiro, segInteiro;
8         double primDouble, segDouble;
9         String primString, segString;
10
11         System.out.print("Digite dois número inteiros");
12         System.out.println(" separados por espaços:");
13         primInteiro = teclado.nextInt();
14         segInteiro = teclado.nextInt();
15         System.out.println("Números digitados: " + primInteiro + " e " + segInteiro);
16
17         System.out.println("Agora, digite dois números com parte decimal: ");
18         primDouble = teclado.nextDouble();
19         segDouble = teclado.nextDouble();
20         System.out.println("Números digitados: " + primDouble + " e " + segDouble);
21
22         System.out.println("Agora, digite duas palavras: ");
23         primString = teclado.next();
24         segString = teclado.next();
25         System.out.println("Palavras digitadas: " + primString + " e " + segString);
26
27         primString = teclado.nextLine();
28
29         System.out.println("Agora, digite uma linha de texto: ");
30         primString = teclado.nextLine();
31         System.out.println("Texto digitado: ");
32
33     }
34 }
```

Elimina o “\n” que
sobrou

Boas práticas de programação

- ▶ Já discutimos ao longo do semestre algumas boas práticas de programação
 - ▶ Nomes de variáveis devem ser significativos
 - ▶ Códigos devem seguir indentações diferentes de acordo com os escopos a que pertencem
 - ▶ Nomes de variáveis devem ser iniciadas com letras minúsculas, assim como nomes de métodos
 - ▶ Nomes de classes devem ser iniciadas com letras maiúsculas
 - ▶ Nomes de variáveis constantes devem ser escritas apenas com letras maiúsculas

Boas práticas de programação - Comentários

- ▶ Bons programadores desenvolvem códigos auto-documentados
 - ▶ Um estilo limpo e bons nomes de variáveis (identificadores)
 - ▶ Informações adicionais a quem está lendo o código são fornecidas por meio de **comentários**
 - ▶ Facilitam muito a leitura
 - ▶ São ignorados pelo compilador. Isso é, o comentário não é inserido no código de máquina do seu programa
- ▶ Java tem três maneiras de inserir comentários
 - ▶ `//` : usado para comentários simples. Tudo após as barras duplas até o fim da linha é considerado comentário
 - ▶ `/* ... */`: usado para comentários maiores, em múltiplas linhas
 - ▶ `/** ... */`: usado para documentação das classes (Javadoc)

```
1  import java.util.Scanner;
2  /**
3   Programa para computar a área de um círculo.
4   Autor: Bruno Magalhães Nogueira
5   Email: bruno@facom.ufms.br
6   Trabalho prático 2
7   Última modificação: 07/03/2015
8   */
9
10 public class CalculaAreaCirculo
11 {
12     /*
13     Para este exemplo, vamos usar PI como uma constante.
14     Por questões de aproximação, vamos considerar seu valor até a quinta casa decimal.
15     */
16     public static final double PI = 3.14159;
17
18     public static void main (String [ ] args)
19     {
20         double raio; // em centímetros
21         double area; // em centímetros quadrados
22         Scanner teclado = new Scanner (System.in);
23
24         System.out.println("Entre o raio do círculo em centímetros: ");
25         raio = teclado.nextDouble();
26
27         area = PI * raio * raio;
28         System.out.print("O círculo de raio de " + raio + " centímetros");
29         System.out.println(" tem uma área de " + area + " centímetros quadrados.");
30
31     }
32 }
```

Exercício

- ▶ Faça um programa Java que receba do usuário uma frase com 5 palavras e mostre :
 - ▶ cada uma das palavras que compõem a frase separadamente;
 - ▶ a primeira letra de cada palavra;
 - ▶ a primeira ocorrência de cada uma das vogais dentro da frase.

Exercício II

- ▶ Em partidas de futebol, a pontuação por uma vitória é de três pontos; a pontuação por um empate é de um ponto; e derrotas não dão pontos. Faça um programa Java que peça ao usuário para digitar:
 - ▶ o nome do time;
 - ▶ o número de vitórias do time
 - ▶ o número de empates do time
 - ▶ o número de derrotas do time

Após, o seu programa deve exibir ao usuário a mensagem: “O time XXXX teve YYYYY pontos, com ZZZZ % de aproveitamento”. Nessa mensagem:

- ▶ o nome do time deve ser formatado para ocupar 15 espaços
- ▶ a pontuação total do time deve ser formatada para ocupar 3 espaços
- ▶ o aproveitamento total do time deve ser formatado para ocupar 8 espaços e ter exatamente 2 casas decimais