

Java: como programar

Programação Orientada à Objetos

- Douglas Baptista de Godoy

 [/in/douglasbgodoy](https://www.linkedin.com/in/douglasbgodoy)

 github.com/douglasbgodoy

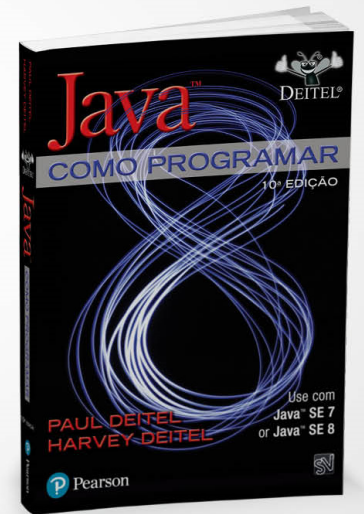
Informação

Obs: Esta aula é baseada nos livros textos, e as transparências são baseadas nas transparências providenciadas pelos autores.

DEITEL, P. J.; DEITEL, H. M. **Java**: como programar. 10. ed. São Paulo, SP: Pearson, 2017. *E-book*. Disponível em: <https://plataforma.bvirtual.com.br>. Acesso em: 27 fev. 2024.

Capítulo 2:

Introdução a aplicativos Java — entrada/saída e operadores



Nosso primeiro programa Java: imprimindo uma linha de texto

- Um aplicativo Java é um programa de computador que é executado quando você utiliza o comando **java** para carregar a Java Virtual Machine (JVM).

Nosso primeiro programa Java: imprimindo uma linha de texto

- Consideremos um aplicativo simples que exibe uma linha de texto. A Figura 2.1 mostra o programa seguido por uma caixa que exibe sua saída.

```
1 // Figura 2.1: Welcome1.java
2 // Programa de impressão de texto.
3
4 public class Welcome1
5 {
6     // método main inicia a execução do aplicativo Java
7     public static void main(String[] args)
8     {
9         System.out.println("Welcome to Java Programming!");
10    } // fim do método main
11 } // fim da classe Welcome1
```

Welcome to Java Programming!

Figura 2.1 | Programa de impressão de texto.

Nosso primeiro programa Java: imprimindo uma linha de texto

- O comentário na linha 1 começa com `//`, indicando que é um **comentário de fim de linha**, e termina no fim da linha, onde os caracteres `//` aparecem.

```
// Figura 2.1: Welcome1.java
```

- A linha 2, de acordo com nossa convenção, é um comentário que descreve o propósito do programa.

```
// Programa de impressão de texto.
```

Nosso primeiro programa Java: imprimindo uma linha de texto

Comentários tradicionais

- Podem ser distribuídos ao longo de várias linhas. Eles começam e terminam com delimitadores, `/*` e `*/`, como em:

```
/* Esse é um comentário tradicional. Ele  
   pode ser dividido em várias linhas */
```

- O compilador ignora todo o texto entre os delimitadores.
- O Java incorporou comentários tradicionais e comentários de fim de linha das linguagens de programação C e C++, respectivamente.

Nosso primeiro programa Java: imprimindo uma linha de texto

- Linhas em branco, caracteres de espaço e tabulações tornam os programas mais fáceis de ler. Juntos, eles são conhecidos como **espaços em branco**.
- O compilador ignora espaços em branco.
- Utilize linhas e espaços em branco para aprimorar a legibilidade do programa.

Nosso primeiro programa Java: imprimindo uma linha de texto

- As **palavras-chave** são reservadas para uso pelo Java e sempre são escritas com todas as letras minúsculas.
- A palavra-chave `class` introduz uma **declaração de classe**.
- Por convenção, todos os **nomes de classes** em Java começam com uma letra maiúscula e apresentam a letra inicial de cada palavra que eles incluem em maiúscula (por exemplo, `SampleClassName`).

Nosso primeiro programa Java: imprimindo uma linha de texto

- O nome de uma classe Java é um **identificador** — uma série de caracteres consistindo em letras, dígitos e sublinhados (_).
- O Java faz **distinção entre maiúsculas e minúsculas**.
- O **corpo** de cada declaração de classe é delimitado por chaves, { e }.

Nosso primeiro programa Java: imprimindo uma linha de texto

- Os métodos realizam tarefas e retornam informações ao concluí-las.
- A palavra-chave `void` indica que um método executará uma tarefa, mas não retornará nenhuma informação.
- As instruções instruem o computador a realizar ações.
- Uma string entre aspas duplas é às vezes chamada de string de caracteres ou string literal.

Nosso primeiro programa Java: imprimindo uma linha de texto

- O objeto de saída padrão (`System.out`) exibe caracteres na janela de comando.
- O método `System.out.println` exibe seu argumento na janela de comando seguido por um caractere de nova linha para posicionar o cursor de saída no começo da próxima linha.

Modificando nosso primeiro programa Java

- `System.out.print` **exibe seu argumento** e posiciona o cursor de saída imediatamente após o último caractere exibido.
- Uma barra invertida (`\`) em uma string é um **caractere de escape**. O Java combina-o com o próximo caractere para formar uma **sequência de escape**. A sequência de escape `\n` representa o caractere de nova linha.

Modificando nosso primeiro programa Java

```
1 // Figura 2.3: Welcome2.java
2 // Imprimindo uma linha de texto com múltiplas instruções.
3
4 public class Welcome2
5 {
6     // método main inicia a execução do aplicativo Java
7     public static void main(String[] args)
8     {
9         System.out.print("Welcome to ");
10        System.out.println("Java Programming!");
11    } // fim do método main
12 } // fim da classe Welcome2
```

Welcome to Java Programming!

Figura 2.3 | Imprimindo uma linha de texto com múltiplas instruções.

Modificando nosso primeiro programa Java

```
1 // Figura 2.4: Welcome3.java
2 // Imprimindo múltiplas linhas de texto com uma única instrução.
3
4 public class Welcome3
5 {
6     // método main inicia a execução do aplicativo Java
7     public static void main(String[] args)
8     {
9         System.out.println("Welcome\nto\nJava\nProgramming!");
10    } // fim do método main
11 } // fim da classe Welcome3
```

```
Welcome
to
Java
Programming!
```

Figura 2.4 | Imprimindo múltiplas linhas de texto com uma única instrução.

Modificando nosso primeiro programa Java

Sequência de escape	Descrição
<code>\n</code>	Nova linha. Posiciona o cursor de tela no início da <i>próxima</i> linha.
<code>\t</code>	Tabulação horizontal. Move o cursor de tela para a próxima parada de tabulação.
<code>\r</code>	Retorno de carro. Posiciona o cursor da tela no início da linha <i>atual</i> — <i>não</i> avança para a próxima linha. Qualquer saída de caracteres depois do retorno de carro <i>sobrescreve</i> a saída de caracteres anteriormente gerada na linha atual.
<code>\\</code>	Barras invertidas. Utilizadas para imprimir um caractere de barra invertida.
<code>\"</code>	Aspas duplas. Utilizadas para imprimir um caractere de aspas duplas. Por exemplo, <pre>System.out.println("\"entre aspas\");</pre> exibe "entre aspas".

Figura 2.5 | Algumas sequências de escape comuns.

Exibindo texto com `printf`

- O método `System.out.printf` (**f** significa “formatado”) exibe os dados formatados.
- O primeiro argumento do método `printf` é uma string de formato contendo especificadores de texto fixo e/ou de formato. Cada especificador de formato indica o tipo de dado a ser gerado e é um espaço reservado para um argumento correspondente que aparece após a string de formato.

Exibindo texto com `printf`

- Especificadores de formato iniciam com um sinal de porcentagem (%) e são seguidos por um caractere que representa o tipo de dado.
- O especificador de formato `%s` é um espaço reservado para uma string de caracteres.
- O especificador de formato `%n` é um separador de linha portátil.

Exibindo texto com printf

```
1 // Figura 2.6: Welcome4.java
2 // Exibindo múltiplas linhas com o método System.out.printf.
3
4 public class Welcome4
5 {
6     // método main inicia a execução do aplicativo Java
7     public static void main(String[] args)
8     {
9         System.out.printf("%s\n%s\n",
10             "Welcome to", "Java Programming!");
11     } // fim do método main
12 } // fim da classe Welcome4
```

```
Welcome to
Java Programming!
```

Figura 2.6 | Exibindo múltiplas linhas com o método System.out.printf.

Declarações import

- Uma declaração `import` ajuda o compilador a localizar uma classe que é usada em um programa.
- O rico conjunto do Java de classes predefinidas é agrupado em pacotes — chamados de **grupos de classes**. Esses são referidos como biblioteca de **classes Java, ou Interface de Programação de Aplicativo Java (API Java)**.

Declarando Variáveis

- Uma variável é uma posição na memória do computador na qual um valor pode ser armazenado para utilização posterior em um programa.
- Todas as variáveis devem ser declaradas com um nome e um tipo antes que possam ser utilizadas.
- O nome de uma variável permite que o programa acesse o valor dela na memória.

Declarando Variáveis :adicionando inteiros

- O tipo de dado `int` é utilizado para declarar variáveis que conterão valores de inteiro.
- O intervalo de valores para um `int` é $-2.147.483.648$ a $+2.147.483.647$.

Declarando Variáveis :adicionando reais e caracteres

- Os tipos **float** e **double** especificam números reais com pontos decimais, como 3.4 e -11.19.
- Variáveis do tipo **char** representam caracteres individuais, como uma letra maiúscula (por exemplo, A), um dígito (por exemplo, 7), um caractere especial (por exemplo, * ou %) ou uma sequência de escape (por exemplo, tab, \t).

Declarando Variáveis

- Tipos como `int`, `float`, `double` e `char` são **primitivos**.
- Os nomes dos tipos primitivos são palavras-chave; portanto, todos devem aparecer em letras minúsculas.

Entrada de dados

- Um prompt direciona o usuário a tomar uma ação específica.
- O método `Scanner.nextInt()` obtém um inteiro para uso em um programa.

Operador de atribuição

- O operador de atribuição, `=`, permite ao programa atribuir um valor a uma variável. Ele é chamado operador binário, porque tem dois operandos.

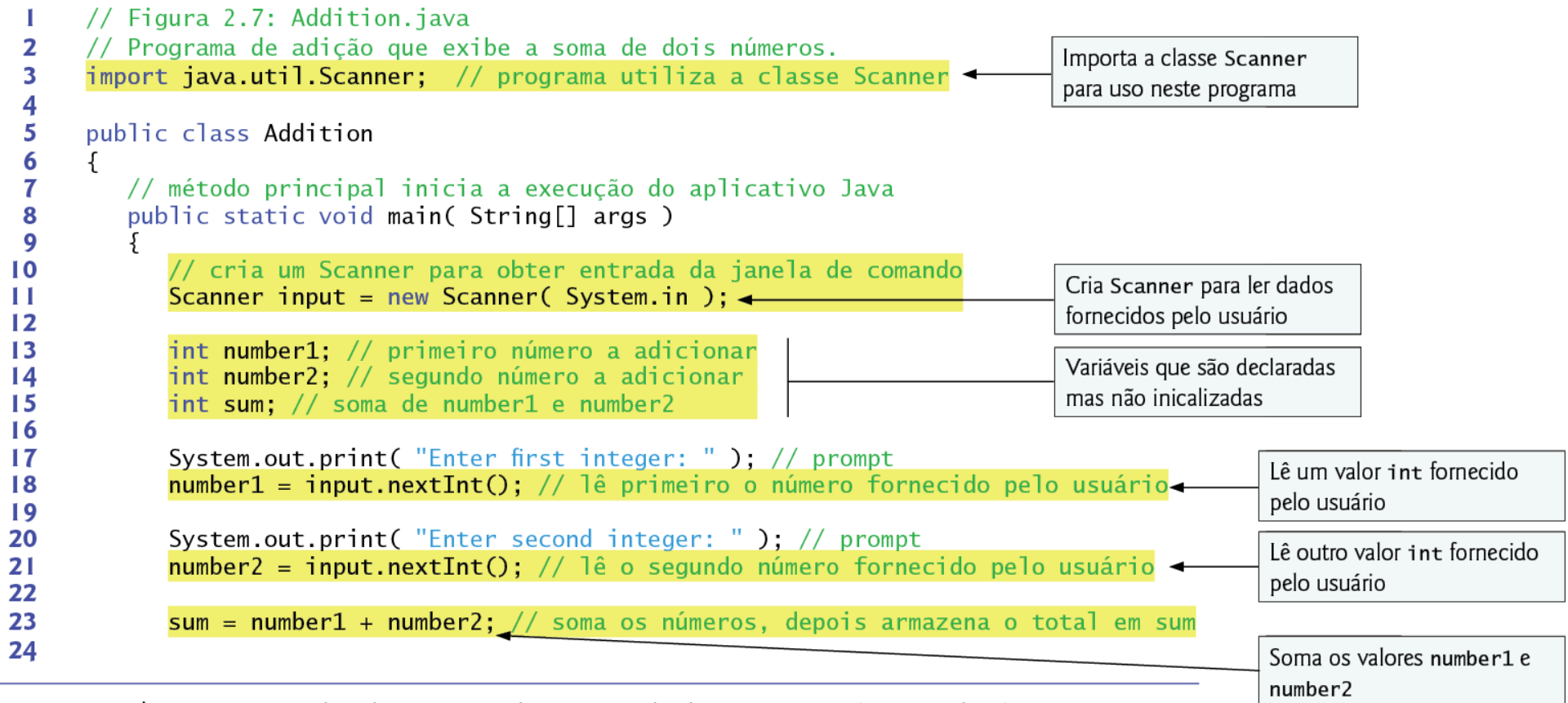


Figura 2.7 | O programa de adição que exibe a soma de dois números. (Parte I de 2)

```
25      System.out.printf( "Sum is %d\n", sum ); // exibe a soma
26  } // fim do método main
27 } // fim da classe Addition
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Figura 2.7 | O programa de adição que exibe a soma de dois números. (Parte 2 de 2.)

Aritmética

- Os operadores aritméticos são + (adição), - (subtração), * (multiplicação), / (divisão) e % (resto).
- A divisão de inteiros produz um quociente com inteiros.
- O operador de resto, %, fornece o resto depois da divisão.

Aritmética

Operação Java	Operador	Expressão algébrica	Expressão Java
Adição	+	$f + 7$	<code>f + 7</code>
Subtração	-	$p - c$	<code>p - c</code>
Multiplicação	*	bm	<code>b * m</code>
Divisão	/	x/y ou $\frac{x}{y}$ ou $x \div y$	<code>x / y</code>
Resto	%	$r \bmod s$	<code>r % s</code>

Figura 2.11 | Operadores aritméticos.

Precedência de operadores aritméticos

Operador(es)	Operação(ões)	Ordem de avaliação (precedência)
* / %	Multiplicação Divisão Resto	Avaliado primeiro. Se houver vários operadores desse tipo, eles são avaliados da <i>esquerda para a direita</i> .
+ -	Adição Subtração	Avaliado em seguida. Se houver vários operadores desse tipo, eles são avaliados da <i>esquerda para a direita</i> .
=	Atribuição	Avaliado por último.

Figura 2.12 | Precedência de operadores aritméticos.

Tomada de decisão: operadores de igualdade e operadores relacionais

- A instrução `if` toma uma decisão baseada no valor de uma condição (verdadeiro ou falso).
- As condições em instruções `if` podem ser formadas utilizando-se os operadores de igualdade (`==` e `!=`) e relacionais (`>`, `<`, `>=` e `<=`).

Tomada de decisão: operadores de igualdade e operadores relacionais

- Uma instrução `if` começa com a palavra-chave `if`, seguida por uma condição entre parênteses, e espera uma instrução no seu corpo.
- A instrução vazia é do tipo que não realiza qualquer tarefa.

Tomada de decisão: operadores de igualdade e operadores relacionais

Operador algébrico	Operador de igualdade ou relacional Java	Exemplo de condição em Java	Significado da condição em Java
<i>Operadores de igualdade</i>			
=	==	x == y	x é igual a y
≠	!=	x != y	x é não igual a y
<i>Operadores relacionais</i>			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	>=	x >= y	x é maior que ou igual a y
≤	<=	x <= y	x é menor que ou igual a y

Figura 2.14 | Operadores de igualdade e operadores relacionais.

```
1 // Figura 2.15: Comparison.java
2 // Compara inteiros utilizando instruções if, operadores
3 // relacionais e operadores de igualdade.
4 import java.util.Scanner; // programa utiliza a classe Scanner
5
6 public class Comparison
7 {
8     // método principal inicia a execução do aplicativo Java
9     public static void main( String[] args )
10    {
11        // cria Scanner para obter entrada da janela de comando
12        Scanner input = new Scanner( System.in );
13
14        int number1; // primeiro número a comparar
15        int number2; // segundo número a comparar
16
17        System.out.print( "Enter first integer: " ); // prompt
18        number1 = input.nextInt(); // lê o primeiro número fornecido pelo usuário
19
20        System.out.print( "Enter second integer: " ); // prompt
21        number2 = input.nextInt(); // lê o segundo número fornecido pelo usuário
22    }
```

Figura 2.15 | Compare números inteiros usando instruções if, operadores relacionais e operadores de igualdade. (Parte I de 3).

23	<code>if (number1 == number2)</code>	
24	<code>System.out.printf("%d == %d\n", number1, number2);</code>	A instrução de saída só executa se os dois números forem iguais
25		
26	<code>if (number1 != number2)</code>	
27	<code>System.out.printf("%d != %d\n", number1, number2);</code>	A instrução de saída só executa se os dois números não forem iguais
28		
29	<code>if (number1 < number2)</code>	
30	<code>System.out.printf("%d < %d\n", number1, number2);</code>	A instrução de saída só executa se number1 for menor que number2
31		
32	<code>if (number1 > number2)</code>	
33	<code>System.out.printf("%d > %d\n", number1, number2);</code>	A instrução de saída só executa se number1 for maior que number2
34		
35	<code>if (number1 <= number2)</code>	
36	<code>System.out.printf("%d <= %d\n", number1, number2);</code>	A instrução de saída só executa se number1 for menor ou igual a number2
37		
38	<code>if (number1 >= number2)</code>	
39	<code>System.out.printf("%d >= %d\n", number1, number2);</code>	A instrução de saída só executa se number1 for maior ou igual a number2
40	<code>} // fim do método main</code>	
41	<code>} // fim da classe Comparison</code>	

Figura 2.15 | Compare números inteiros usando instruções `if`, operadores relacionais e operadores de igualdade. (Parte 2 de 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
```

Figura 2.15 | Compare números inteiros usando instruções `if`, operadores relacionais e operadores de igualdade. (Parte 3 de 3.)

Operadores de precedência e de associatividade

Operadores	Associatividade	Tipo
* / %	da esquerda para a direita	multiplicativo
+ -	da esquerda para a direita	aditivo
< <= > >=	da esquerda para a direita	relacional
== !=	da esquerda para a direita	igualdade
=	da direita para a esquerda	atribuição

Figura 2.16 | Operadores de precedência e de associatividade discutidos.

Referências Bibliográficas

- DEITEL, P. J.; DEITEL, H. M. **Java:** como programar. 10. ed. São Paulo, SP: Pearson, 2017.