



Java

Bootcamp Programador de Software Iniciante

João Paulo Barbosa Nascimento

2020

Java

João Paulo Barbosa Nascimento

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

Sumário

Capítulo 1. Introdução	4
Linguagem tipada e não tipada.....	4
Introdução ao Java	5
A JVM (Java Virtual Machine).....	7
Ambientes de programação	8
Instalando a IDE	9
Desenvolvimento, compilação e execução	10
Capítulo 2. Introdução aos tipos de dados e operadores	11
O famigerado “Olá Mundo”	11
Tipos de dados primitivos	12
O operador de atribuição	13
Manipulando dados primitivos.....	14
O tipo String	14
Manipulando String	15
Métodos Print, Println e Printf	15
Operadores aritméticos.....	16
Trabalhando com operadores aritméticos.....	16
Operadores lógicos.....	20
Contadores e acumuladores	22
Capítulo 3. Instruções de controle de programa.....	24
Leitura de dados do teclado.....	24
A instrução if	24
Ifs aninhados.....	25
If-else-if	26
Referências.....	28

Capítulo 1. Introdução

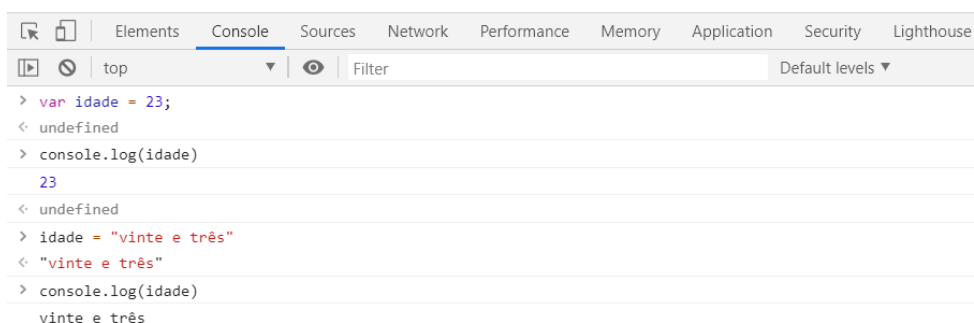
Neste capítulo iremos realizar uma introdução à linguagem de programação Java, abordando alguns conceitos fundamentais, tais como: uma comparação entre linguagens tipadas e não tipadas, o surgimento da linguagem Java, a JVM (Java Virtual Machine), o ambiente de programação e o conceito de tempo de desenvolvimento, de compilação e de execução.

Linguagem tipada e não tipada.

A linguagem não tipada tem os tipos de suas variáveis definidos no momento da execução do programa. Com isso, é possível que uma mesma variável possua diferentes tipos de conteúdo em diferentes partes do programa. Uma variável, por exemplo, pode receber o valor inteiro 16 no início do programa e depois, no meio desse mesmo programa, essa mesma variável pode receber o valor da cadeia de caracteres “São Paulo”, sem que isso afete a execução do programa. Diante do exemplo apresentado, uma linguagem não tipada não depende de conversões de dados (cast) para realizar armazenamento de tipos de dados diferentes.

Alguns exemplos de linguagens de programação não tipada ou fracamente tipada são: PHP, JavaScript, Ruby e Python. A Figura 1 apresenta um exemplo de programa em JavaScript que realiza o armazenamento de diferentes tipos de dados para a variável idade.

Figura 1 – Exemplo de diferentes tipos de dados para uma mesma variável.



```

> var idade = 23;
< undefined
> console.log(idade)
23
< undefined
> idade = "vinte e três"
< "vinte e três"
> console.log(idade)
vinte e três

```

Fonte: o autor.

Por outro lado, uma linguagem tipada ou fortemente tipada exige que os tipos das variáveis utilizadas em um programa sejam explicitamente definidos durante o desenvolvimento do código-fonte do programa. A verificação dos tipos das variáveis sempre é feita antes da realização de cada operação do programa.

Existem tipos específicos para as variáveis de uma linguagem fortemente tipada, tais como: int, float, boolean e char. Além disso, existem os tipos que são específicos e podem ser criados diretamente pelo desenvolvedor do programa. Toda a verificação dos tipos é realizada pelo compilador antes do programa ser efetivamente executado. A Figura 2 apresenta um exemplo de um programa desenvolvido em Java e que possui 4 variáveis. Nas linhas 6, 7, 8 e 9 é realizada a declaração dessas variáveis.

Figura 2 – Exemplo de declaração de variáveis em uma linguagem tipada.

```
1
2 public class Exemplo {
3
4     public static void main (String[] args) {
5
6         int numero;
7         char letra;
8         String nomeSobrenome;
9         boolean possuiCadastro;
10    }
11 }
```

Fonte: o autor.

As linguagens tipadas permitem que seja realizada a declaração de variáveis, funções e métodos de tipos específicos, gerando códigos que são mais legíveis. Algumas linguagens que possuem a tipagem forte são: C++, Java e C#.

Introdução ao Java

A linguagem Java foi concebida em 1991, possuindo inicialmente o nome de Oak. Posteriormente, já em 1995, teve o nome alterado para Java. Atualmente o Java

é uma das linguagens mais usadas no mundo e podemos dizer que é uma ferramenta para desenvolvedores de software que caracteriza e define a nossa época.

A principal motivação para a criação de uma nova linguagem foi a independência de plataforma. Antes da criação do Java, os programas desenvolvidos eram extremamente dependentes das plataformas para as quais eles foram desenvolvidos. Com isso, havia um grande trabalho de criar a portabilidade desses programas para cada uma das plataformas que eles deveriam funcionar. Essas plataformas iam desde um simples controle remoto, passando por fornos micro-ondas, TVs e chegando até os computadores e telefones, cada qual com sua arquitetura de hardware e sistemas operacionais específicos.

Na época, a maioria das linguagens de programação eram projetadas para serem compiladas para o código de máquina de um tipo específico de CPU (Central Process Unit). Nessa época, era possível compilar um programa C++ para grande parte dessas CPU's, mas era preciso um compilador específico para cada uma delas. Esse processo de compilar ou criar o compilador para cada CPU era extremamente caro e demorado. Diante desta dificuldade, o Java surgiu como uma grande alternativa para a portabilidade das aplicações, pois a grande inovação por trás desta linguagem foi justamente a possibilidade de compilar o programa apenas uma vez e executá-lo diversas vezes em diferentes ambientes.

Uma força que possibilitou o crescimento do Java foi a popularização e crescimento da Web. A Web possui diversos tipos de equipamentos, cada um com suas especificidades e características. Além disso, esses equipamentos possuem diferentes sistemas operacionais e era necessário um mecanismo que pudesse permitir que esse variado grupo executasse um mesmo programa.

Em 1993 o foco da linguagem saiu dos dispositivos eletrônicos para se concentrar na Internet. O Java está relacionado a duas linguagens, o C e o C++, herdando a sintaxe do C e o modelo de objetos do C++.

A JVM (Java Virtual Machine)

A grande vantagem da utilização da linguagem Java está na possibilidade de executarmos nossos programas em diferentes CPU's, equipamentos e sistemas operacionais, sem a necessidade de realizarmos o processo de recompilar (reconstruir) nossos programas.

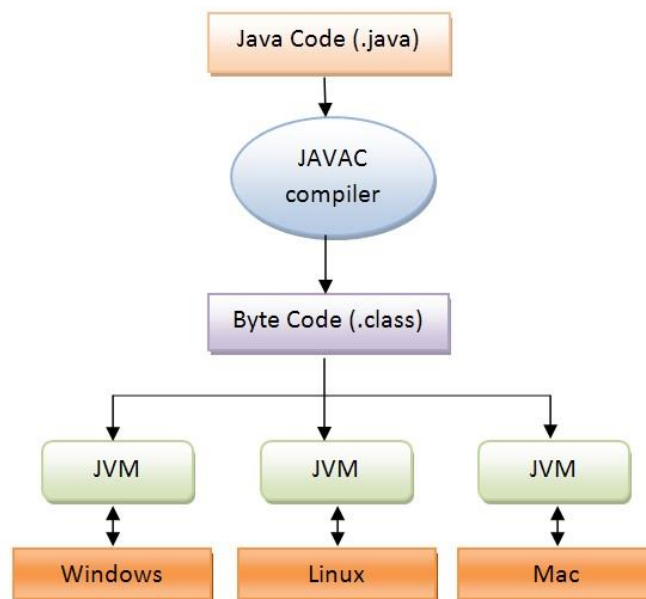
Sempre ouvimos as pessoas falarem que os programas Java “rodam” em qualquer lugar, ou seja, em qualquer ambiente. Mas como essa “mágica” acontece? O segredo está na JVM (Java Virtual Machine) ou máquina virtual do Java.

Em uma linguagem convencional, como C++ ou Pascal, o programa é compilado para uma determinada CPU ou sistema operacional, como Windows, Unix ou Linux. Para que possamos executar esse programa em um ambiente específico é preciso que executemos novamente o processo de compilação, utilizando um compilador específico para o ambiente em questão. Com esse processo, será gerado um código específico para a máquina ou sistema operacional em questão.

O Java não tem esse problema. A execução dos programas desenvolvidos nesta linguagem não está relacionada com o sistema operacional, mas sim com a JVM. Diante disso, temos várias JVMs, cada qual para um dispositivo ou sistema operacional específico, garantindo uma maior portabilidade do código. Se o programa for escrito para o ambiente Microsoft Windows, ele também executará no ambiente Linux, sem a necessidade de recompilação. Cada um dos ambientes possui a sua própria JVM, proporcionando com isso uma maior portabilidade e produtividade. A Figura 3 apresenta a arquitetura da JVM.

O código Java é gerado no topo da figura e, em seguida, o processo de compilação é realizado por meio do JAVAC Compiler. Após o processo de compilação será gerado o Byte Code (class) que é o código que as JVM's “entendem”. Esse bytecode é enviado para a JVM específica do sistema operacional ou dispositivo que irá executar a aplicação compilada. Ainda na Figura 3, podemos observar que cada um dos sistemas operacionais apresentados (Windows, Linux e Mac) possuem a sua JVM correspondente.

Figura 3 – Arquitetura da JVM.



Fonte: Allan, 2013.

A JVM é responsável ainda por outras tarefas, tais como interpretar o bytecode gerado, gerenciar memória e processamento e realizar a coleta de lixo (Garbage Colector) que é o processo de eliminar da memória virtual variáveis e objetos não utilizados. A JVM entende o bytecode e o traduz para o sistema operacional ou dispositivo em questão.

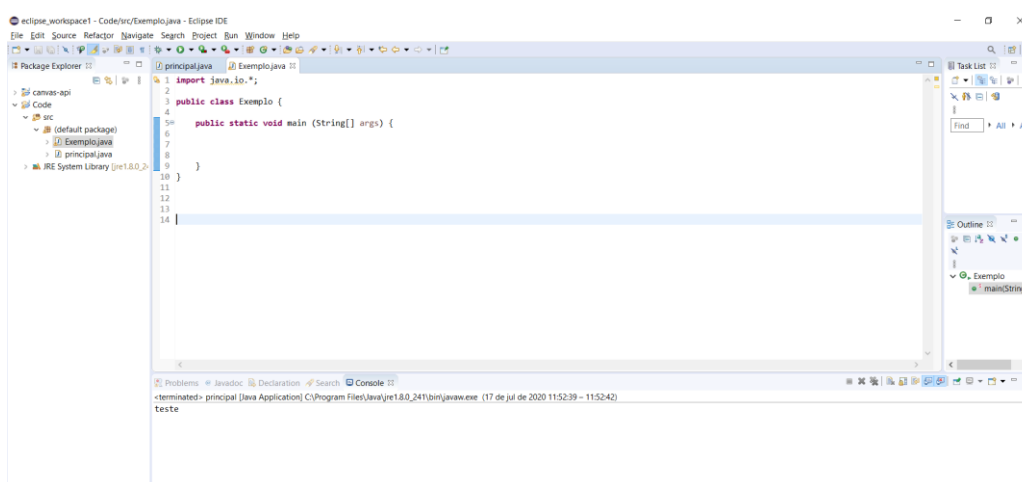
Ambientes de programação

O principal componente presente no ambiente de programação é a IDE (Integrated Development Enviroment) que é o ambiente integrado de desenvolvimento. A IDE nada mais é que um programa de computador que auxilia o desenvolvedor de software a criar outros programas de computador. Ela possui várias facilidades que auxiliam o programador em suas rotinas diárias, tais como: compilação por meio de cliques de botão, organização dos arquivos de projeto, destaques de trechos de códigos específicos com cores, complemento automático de código e depuração que auxilia o processo de localizar defeitos no código.

Para criar programas em Java não é obrigatório a utilização de uma IDE. Essas ferramentas trazem diversas facilidades para o dia a dia do desenvolvedor, porém é também possível criar os programas utilizando um editor de texto qualquer, tais como Bloco de Notas, Vi ou Notepad++. Ao utilizar editores de textos simples, será necessário realizar a compilação do programa por meio de linhas de comando.

A Figura 4 apresenta a IDE Eclipse, uma das mais utilizadas para desenvolvimento em Java. Além dela, temos também o NetBeans, o JCreator e o IntelliJ Idea, dentre várias outras como opções para criação de programas Java.

Figura 4 – A IDE Eclipse.



Fonte: o autor.

Instalando a IDE

O processo de preparação do ambiente Java passa pela instalação do JDK (Java Development Kit) – kit de desenvolvimento Java – e a IDE propriamente dita.

Para instalar a JDK, dirija-se até o link: <https://www.oracle.com/technetwork/pt/java/javase/downloads/index.html>. No momento do download deve-se escolher o pacote correspondente ao seu Sistema Operacional.

Para instalar a IDE, deve-se fazer o download do instalador em:
<https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2020-06/R/eclipse-inst-win64.exe>.

Desenvolvimento, compilação e execução

Durante a criação dos nossos programas, devemos estar muito atentos aos tempos existentes no processo, que são: desenvolvimento, compilação e execução.

O tempo de Desenvolvimento ocorre quando estamos com a nossa IDE aberta e escrevendo o nosso código. Nesse momento não há nenhuma interação humano-máquina. Simplesmente não há execução do programa, ou seja, ele se encontra estático.

Temos um segundo tempo que é o de compilação. Esse processo se inicia quando clicamos no botão RUN da IDE. Nesse momento, o JAVC Compiler ou simplesmente o compilador Java, irá realizar todas as checagens do código fonte implementado para gerar o bytecode. Durante o processo, se existir algum erro no código, este será destacado pelo compilador e os bytecodes não serão gerados. Se não tivermos nenhuma inconsistência no código, os bytecodes serão gerados e teremos o programa pronto para ser executado.

Caso o processo de compilação tenha terminado sem erros, poderemos ter o terceiro tempo, que é o de execução. Nesse tempo o programa estará em processo de execução e haverá a plena interação humano-máquina, onde os usuários poderão inserir dados de entrada e receber respostas como saída. É importante destacar que, caso haja modificações no código-fonte, o programa deverá ser novamente compilado e a execução deverá ser reiniciada, para que as alterações realizadas possam ser percebidas no tempo de execução.

Capítulo 2. Introdução aos tipos de dados e operadores

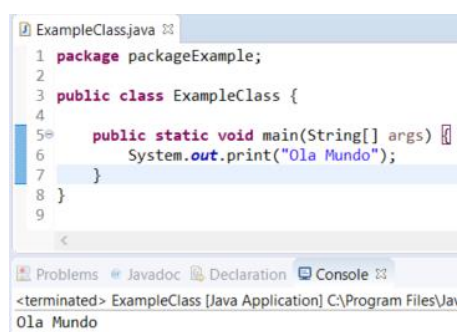
Neste capítulo destacaremos os tipos de dados fornecidos pela linguagem Java para a construção dos nossos programas. Os dados primitivos (Inteiros, ponto-flutuante, booleano e caractere) serão apresentados. Os operadores de atribuição, aritméticos e lógicos serão apresentados. Por último, será apresentado um tópico para o detalhamento dos Contadores e Acumuladores.

O famigerado “Olá Mundo”

Quando estamos iniciando os estudos em uma linguagem de programação é muito comum que o primeiro programa a ser criado realize a impressão na tela da mensagem “Olá Mundo”. Isso é exatamente o que será feito aqui por meio do comando `System.out.print (“Olá Mundo”)`.

Para criar o seu primeiro programa, crie um novo projeto no Eclipse, uma nova classe e um novo método void main. A aula em vídeo 2.1 apresenta os detalhes de como realizar essa operação. A Figura 5 apresenta o programa implementado.

Figura 5 – Um “Olá Mundo” em Java.



```
ExampleClass.java
1 package packageExample;
2
3 public class ExampleClass {
4
5     public static void main(String[] args) {
6         System.out.print("Ola Mundo");
7     }
8 }
9
```

Problems Javadoc Declaration Console

<terminated> ExampleClass [Java Application] C:\Program Files\Ja
Ola Mundo

Fonte: o autor.

Tipos de dados primitivos

Como vimos anteriormente, os tipos estão sempre presentes em linguagens fortemente tipadas, como é o caso do Java. Essa presença pode aumentar a confiabilidade nos dados armazenados nas variáveis. Nessas linguagens não há o conceito de variável sem tipo.

No Java temos duas categorias principais de tipos de dados: orientada a objetos e não orientada a objetos. Os tipos primitivos são utilizados para indicar os tipos que não são orientados a objetos, ou seja usados para armazenar valores binários comuns. Os tipos primitivos são utilizados para formar uma base para todos os outros tipos, no caso do Java, os tipos orientados a objetos. A Figura 6 apresenta os tipos primitivos presentes no Java.

Figura 6 – Tipos primitivos do Java.

Tipo	Significado
boolean	Representa os valores verdadeiro/falso
byte	Inteiro de 8 bits
char	Caractere
double	Ponto flutuante de precisão dupla
float	Ponto flutuante de precisão simples
int	Inteiro
long	Inteiro longo
short	Inteiro curto

Fonte: Schildt, 2014.

O Java define quatro tipos inteiros: byte, short, int e long. Cada um possui seu tamanho específico e o intervalo de valores que pode armazenar. A Figura 7 apresenta essas características:

Figura 7 – Tipos inteiros em Java.

Tipo	Tamanho em bits	Intervalo
byte	8	-128 a 127
short	16	-32.768 a 32.767
int	32	-2.147.483.648 a 2.147.483.647
long	64	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

Fonte: Schildt, 2014.

O tipo ponto flutuante representa números que possuem componentes fracionários. Existem duas espécies de ponto flutuante: float e double. O float possui 32 bits de alocação e o double 64 bits.

O tipo caractere utiliza 16 bits para armazenamento de caracteres Unicode. O Unicode define um conjunto de caracteres que podem representar todos os caracteres encontrados em todos os idiomas humanos.

O tipo boolean ou booleano representa e armazena valores verdadeiro ou falso. O Java representa o conteúdo desse tipo por meio das palavras reservadas em inglês true e false. Uma variável boolean terá um e apenas um desses valores.

O operador de atribuição

Utilizamos o operador de atribuição quando desejamos armazenar um valor dentro de uma variável. Em Java, o sinal de atribuição é representado por um símbolo de igual simples (=).

A Figura 8 apresenta quatro exemplos de atribuições. O valor Z será armazenado dentro do char tipo, o valor 23 dentro do inteiro idade, o valor 3589.65 dentro do ponto flutuante salario e o valor verdadeiro dentro da variável de tipo booleano chamada estudante.

Figura 8 – Exemplo de atribuição em Java.

```
*Exemplo.java
1 package Pacote;
2
3 public class Exemplo {
4
5     public static void main(String[] args) {
6         char tipo = 'Z';
7         int idade = 23;
8         double salario = 3589.65;
9         boolean estudante = true;
10    }
11 }
```

Fonte: o autor.

Manipulando dados primitivos

A manipulação de dados primitivos no Java é uma operação relativamente simples. O formato é: [Tipo] [NomeVariavel] = [valor]. A Figura 9 apresenta a manipulação de dados em cada um dos tipos primitivos, respeitando o formato aqui apresentado.

Figura 9 – Manipulação de tipos primitivos em Java.

```
*ExampleClass.java
1 package packageExample;
2
3 public class ExampleClass {
4
5     public static void main(String[] args) {
6         int numeroFilhos = 2;
7         double altura = 1.80;
8         char possuiAutomovel = 'S';
9         boolean dadosChecados = true;
10    }
11 }
```

Fonte: o autor.

O tipo String

O tipo String não é um tipo primitivo. Ele é uma classe do Java que permite o armazenamento de sequências ou cadeias de caracteres. Uma string é imutável, ou seja, o texto que ela carrega nunca é alterado. Sempre que uma alteração é necessária, será utilizado mais espaço na memória contendo a nova versão dos dados. No Java, qualquer texto entre aspas duplas é considerado uma String. Podemos criar uma String usando o formato padrão: [String] [nome] = "valor"; A Figura 10 apresenta exemplos de criação e atribuição de valores à objetos do tipo String.

Figura 10 – Criação e atribuição de valores para objetos String em Java.

```
1 package Pacote;  
2  
3 public class ManipulandoString {  
4  
5     public static void main(String[] args) {  
6         String nome = "José Carlos Oliveira de Souza";  
7         String endereco = "Rua Duque de Caxias, 73";  
8         String complemento = "Apto 102", cidade = "Rio de Janeiro", estado = "RJ";  
9     }  
10  
11 }
```

Fonte: o autor.

Manipulando String

O String, por ser uma classe do Java, já traz implementado vários métodos que podem facilitar o dia a dia do desenvolvedor. Alguns métodos são utilizados para diversas tarefas, tais como: descobrir o tamanho de uma string, colocar todos os caracteres da variável em caixa alta ou em caixa baixa, recuperar uma parte da string, concatenar string, etc.

Métodos Print, Println e Printf

Os comandos Print, Println e Printf são comandos de saída do Java. São utilizados para produzir valores na tela do usuário.

O comando Print tem a capacidade de apresentar dados na tela para o usuário sem a quebra de linha após a impressão dos dados. O comando Println realiza a mesma função do Println, porém ao final da impressão dos dados na tela, a linha será quebrada.

Por último, o comando Printf permite que conteúdos de variáveis sejam concatenados à string. Para isso, fazemos uso de curingas representados pelo caractere %. Para concatenar o conteúdo de um string, usamos %s. Para inteiro %d,

boolean %b, char %c e ponto flutuante %.2d, onde o 2 representa a quantidade de casas decimais do ponto flutuante.

Operadores aritméticos

Os operadores aritméticos são importantes e muito utilizados durante o desenvolvimento de programas e a linguagem Java traz um rico ambiente de operadores para serem utilizados. Um operador é um símbolo que solicita ao compilador que execute uma operação aritmética ou lógica. A Figura 11 apresenta os operadores aritméticos disponíveis no Java.

Figura 11 – Operadores aritméticos do Java.

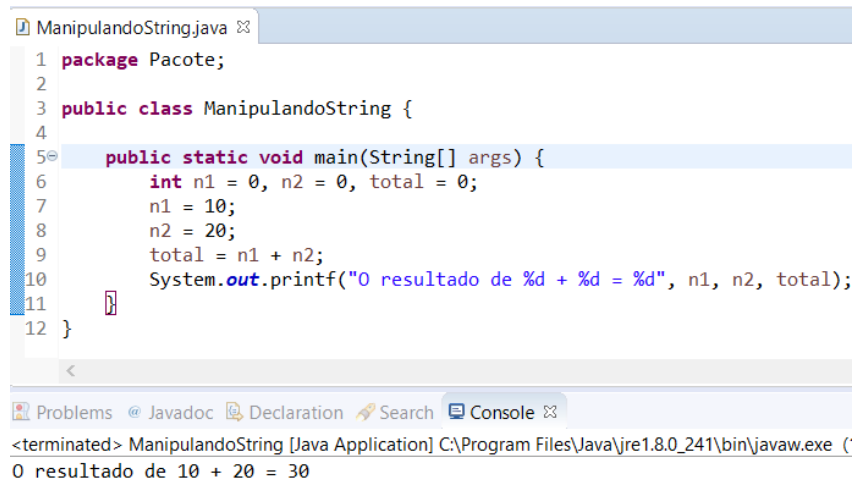
Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo
++	Incremento
--	Decremento

Fonte: o autor.

Trabalhando com operadores aritméticos

O operador de adição é representado pelo símbolo “ + ” e permite a soma aritmética de uma ou mais variáveis. A Figura 12 apresenta a implementação de um programa que soma dois números e armazena o resultado em uma variável.

Figura 12 – Uso do operador de adição.



```

1 package Pacote;
2
3 public class ManipulandoString {
4
5     public static void main(String[] args) {
6         int n1 = 0, n2 = 0, total = 0;
7         n1 = 10;
8         n2 = 20;
9         total = n1 + n2;
10        System.out.printf("0 resultado de %d + %d = %d", n1, n2, total);
11    }
12 }

```

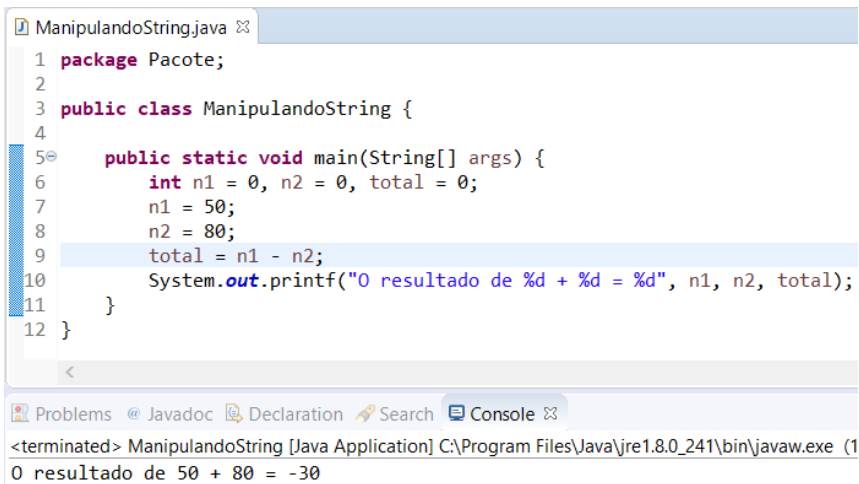
<terminated> ManipulandoString [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (

0 resultado de 10 + 20 = 30

Fonte: o autor.

O operador de subtração é representado pelo símbolo “ - ” e permite a subtração aritmética de uma ou mais variáveis. A Figura 13 apresenta a implementação de um programa que subtrai dois números e armazena o resultado em uma variável.

Figura 13 – Uso do operador de subtração.



```

1 package Pacote;
2
3 public class ManipulandoString {
4
5     public static void main(String[] args) {
6         int n1 = 0, n2 = 0, total = 0;
7         n1 = 50;
8         n2 = 80;
9         total = n1 - n2;
10        System.out.printf("0 resultado de %d + %d = %d", n1, n2, total);
11    }
12 }

```

<terminated> ManipulandoString [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (1

0 resultado de 50 + 80 = -30

Fonte: o autor.

O operador de multiplicação é representado pelo símbolo “ * ” e permite a multiplicação de uma ou mais variáveis. A Figura 14 apresenta a implementação de um programa que multiplica dois números e armazena o resultado em uma variável.

Figura 14 – Uso do operador de multiplicação.

```

1 package Pacote;
2
3 public class ManipulandoString {
4
5     public static void main(String[] args) {
6         int n1 = 0, n2 = 0, total = 0;
7         n1 = 2;
8         n2 = 3;
9         total = n1 * n2;
10        System.out.printf("O número %d multiplicado por %d é igual a | %d", n1, n2, total);
11    }
12 }

```

<terminated> ManipulandoString [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe
0 número 2 multiplicado por 3 é igual a 6

Fonte: o autor.

O operador de divisão é representado pelo símbolo “ / ” e permite a divisão de uma ou mais variáveis. A Figura 15 apresenta a implementação de um programa que divide dois números e armazena o resultado em uma variável.

Figura 15 – Uso do operador de divisão.

```

1 package Pacote;
2
3 public class ManipulandoString {
4
5     public static void main(String[] args) {
6         int n1 = 0, n2 = 0, total = 0;
7         n1 = 2;
8         n2 = 3;
9         total = n1 * n2;
10        System.out.printf("O número %d multiplicado por %d é igual a | %d", n1, n2, total);
11    }
12 }

```

<terminated> ManipulandoString [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe
0 número 2 multiplicado por 3 é igual a 6

Fonte: o autor.

O operador de resto da divisão ou módulo é representado pelo símbolo % e permite recuperar o resto da divisão entre dois números. A Figura 16 apresenta a implementação de um programa que calcula o resto da divisão entre a divisão de dois números.

Figura 16 – Uso do operador de módulo.

```

ManipulandoString.java
1 package Pacote;
2
3 public class ManipulandoString {
4
5     public static void main(String[] args) {
6         double n1 = 0, n2 = 0, total = 0;
7         n1 = 83;
8         n2 = 4;
9         total = n1 % n2;
10        System.out.printf("O número %.2f dividido por %.2f possui resto %.4f", n1, n2, total);
11    }
12 }

```

Problems @ Javadoc Declaration Search Console

<terminated> ManipulandoString [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe

O número 83,00 dividido por 4,00 possui resto 3,0000

Fonte: o autor.

O operador de incremento tem a capacidade de aumentar o valor de uma variável em uma unidade e é representado por “++”. Assim, “x++” e “x = x + 1” possuem o mesmo resultado prático. A Figura 17 apresenta um exemplo de utilização do operador de incremento.

Figura 17 – Uso do operador de incremento.

```

ManipulandoString.java
3 public class ManipulandoString {
4
5     public static void main(String[] args) {
6         int x = 0;
7         System.out.printf("Valor de x: %d\n", x);
8         x++;
9         System.out.printf("Valor de x: %d\n", x);
10        x++;
11        System.out.printf("Valor de x: %d\n", x);
12        x = x + 1;
13        System.out.printf("Valor de x: %d\n", x);
14        x = x + 2;
15        System.out.printf("Valor de x: %d", x);

```

Problems @ Javadoc Declaration Search Console

<terminated> ManipulandoString [Java Application] C:\Program Files\Java\

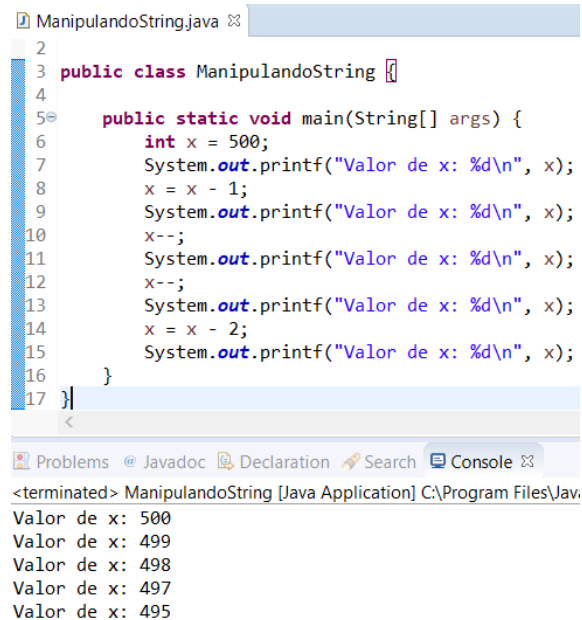
Valor de x: 0
 Valor de x: 1
 Valor de x: 2
 Valor de x: 3
 Valor de x: 5

Fonte: o autor.

O operador de decremento tem a capacidade de diminuir o valor de uma variável em uma unidade e é representado por “--”. Assim, “x--” e “x = x - 1” possuem

o mesmo resultado prático. A Figura 18 apresenta um exemplo de utilização do operador de decremento.

Figura 18 – Uso do operador de decremento.



```
ManipulandoString.java
2
3 public class ManipulandoString {
4
5     public static void main(String[] args) {
6         int x = 500;
7         System.out.printf("Valor de x: %d\n", x);
8         x = x - 1;
9         System.out.printf("Valor de x: %d\n", x);
10        x--;
11        System.out.printf("Valor de x: %d\n", x);
12        x--;
13        System.out.printf("Valor de x: %d\n", x);
14        x = x - 2;
15        System.out.printf("Valor de x: %d\n", x);
16    }
17 }
```

<terminated> ManipulandoString [Java Application] C:\Program Files\Jav...
Valor de x: 500
Valor de x: 499
Valor de x: 498
Valor de x: 497
Valor de x: 495

Fonte: o autor.

Operadores lógicos

Existem dois tipos de operadores: os relacionais e os lógicos. Os operadores relacionais referem-se aos relacionamentos que os valores podem ter uns com os outros. O resultado da aplicação desses operadores será sempre um tipo booleano. A Figura 19 apresenta a lista de operadores relacionais.

Figura 19 – Operador relacionais.

Operador	Significado
==	Igual a
!=	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Fonte: o autor.

Os operadores lógicos são os valores verdadeiro ou falso que podem resultar de uma operação E, OU e Não que podem ser aplicadas à variáveis ou expressões booleanas. No Java, a operação E é representada pelo símbolo “ && ”, a operação OU pelo “ || ” e a operação não pelo símbolo “ ! ”. A Figura 20 apresenta uma aplicação que usa os 3 operadores lógicos aqui apresentados.

Figura 20 – Uso dos operadores lógicos E, OU e Não.

```

4 public static void main(String[] args) {
5     boolean p, q, s;
6     p = false; q = false;
7     s = p && q;
8     System.out.printf("p: %b; q: %b; s: %b\n", p, q, s);
9
10    p = true; q = false;
11    s = p && q;
12    System.out.printf("p: %b; q: %b; s: %b\n", p, q, s);
13
14    p = false; q = true;
15    s = p && q;
16    System.out.printf("p: %b; q: %b; s: %b\n", p, q, s);
17
18    p = true; q = true;
19    s = p && q;
20    System.out.printf("p: %b; q: %b; s: %b", p, q, s);
21 }
22 }

```

Console

```

<terminated> ManipulandoString [Java Application] C:\Program Files\Java\jre1.8.0_241\bin
p: false; q: false; s: false
p: true; q: false; s: false
p: false; q: true; s: false
p: true; q: true; s: true

```

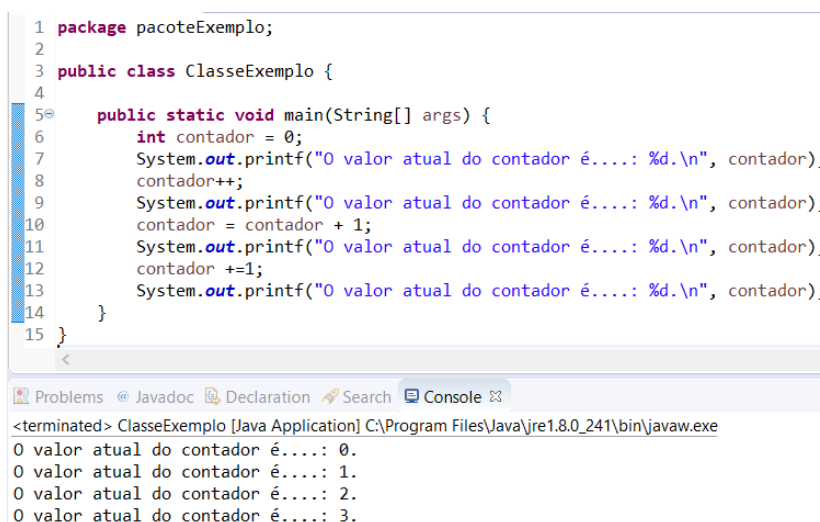
Fonte: o autor.

Contadores e acumuladores

Os contadores e os acumuladores são variáveis, conforme outras que já vimos anteriormente. O que caracteriza cada um dos conceitos que veremos neste tópico é a forma como os valores são atribuídos nessas variáveis.

A variável contadora é normalmente do tipo inteiro e iniciada com valor zero. A medida que o programa executa, essa variável é incrementada de 1 em 1 a cada vez que uma determinada ocorrência acontece. Não necessariamente a variável precisa ser incrementada de 1 em 1. Pode ser que seja de 2 em 2 ou de 5 em 5. O importante é que o formato seja: variável = variável + constante. A Figura 21 apresenta um exemplo de variável contadora.

Figura 21 – Exemplo de variável contadora.



```

1 package pacoteExemplo;
2
3 public class ClasseExemplo {
4
5     public static void main(String[] args) {
6         int contador = 0;
7         System.out.printf("O valor atual do contador é....: %d.\n", contador);
8         contador++;
9         System.out.printf("O valor atual do contador é....: %d.\n", contador);
10        contador = contador + 1;
11        System.out.printf("O valor atual do contador é....: %d.\n", contador);
12        contador +=1;
13        System.out.printf("O valor atual do contador é....: %d.\n", contador);
14    }
15 }

```

Problems Javadoc Declaration Search Console

<terminated> ClasseExemplo [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe

O valor atual do contador é....: 0.
O valor atual do contador é....: 1.
O valor atual do contador é....: 2.
O valor atual do contador é....: 3.

Fonte: o autor.

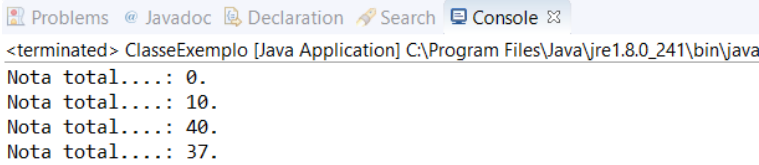
A variável acumuladora é normalmente utilizada em duas situações: na criação de somatórios e produtórios. Ela possui a característica de ser incrementada baseada no valor de uma outra variável qualquer. A Figura 22 apresenta um exemplo de variável acumuladora.

Figura 22 – Exemplo de variável acumuladora.

```

5 public static void main(String[] args) {
6     int notaAtividade = 0, notaTotal = 0;
7     System.out.printf("Nota total....: %d.\n", notaTotal);
8     notaAtividade = 10;
9     notaTotal = notaTotal + notaAtividade;
10    System.out.printf("Nota total....: %d.\n", notaTotal);
11    notaAtividade = 30;
12    notaTotal += notaAtividade;
13    System.out.printf("Nota total....: %d.\n", notaTotal);
14    notaAtividade = -3;
15    notaTotal = notaTotal + notaAtividade;
16    System.out.printf("Nota total....: %d.\n", notaTotal);
17 }
18 }
19

```



 <terminated> ClasseExemplo [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\java

 Nota total....: 0.

 Nota total....: 10.

 Nota total....: 40.

 Nota total....: 37.

Fonte: o autor.

Capítulo 3. Instruções de controle de programa

Este capítulo apresenta os comandos mais importantes para que o controle sobre o fluxo do programa seja mantido. Aqui estudaremos os desvios condicionais `if`, `if else`, `if aninhado` e `if-else-if`. Além disso trabalharemos com os comandos condicionais `switch` e as estruturas de repetição `For`, `While` e `Do-While`. Fecharemos o capítulo com a leitura e gravação de arquivos em disco.

Leitura de dados do teclado

A leitura de dados do teclado é um processo de entrada de dados e extremamente importante para garantir o dinamismo de nossas aplicações. É importante que o usuário possa manipular os programas por meio da entrada de dados.

No nosso curso vamos trabalhar com a classe `Scanner` que permite a criação de um objeto para a leitura de dados do teclado. Para realizar essa leitura, podemos utilizar um método para cada tipo de dados: `nextInt`, `nextDouble` e `nextLine`.

A instrução `if`

O `if` é uma instrução que permite que fluxos ou caminhos diferentes do programa sejam executados, baseado em uma determinada condição. A estrutura do `if` é apresentada abaixo:

```
if (condição)
    instrução1;
else
    instrução2;
```

Se a condição for verdadeira, o conteúdo do `if` (`instrução1`) será executada. Caso contrário, se a condição for falsa, o conteúdo do `else` será executado. É

importante destacar que somente um será executado, ou o if ou o else, nunca os dois. Além disso, o else é opcional. Podemos ter um if sem else. Obrigatoriamente a condição que controla o if deve produzir um resultado booleano e o else nunca terá uma condição. A Figura 23 apresenta um exemplo de implementação do if com else.

Figura 23 – Exemplo de if else.

```

5 public class ClasseExemplo {
6
7     public static void main(String[] args) {
8
9         char caractereDigitado, resposta = 'K';
10        Scanner entrada = new Scanner(System.in);
11
12        System.out.println("Tente adivinhar uma letra de A até Z:");
13        caractereDigitado = entrada.nextLine().charAt(0);
14        if (caractereDigitado == resposta)
15            System.out.println("Resposta correta");
16        else
17            System.out.println("Tente novamente");
18    }
}

```

Problems @ Javadoc Declaration Search Console

<terminated> ClasseExemplo [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe

Tente adivinhar uma letra de A até Z:

A

Tente novamente

Fonte: o autor.

Ifs aninhados

Um if aninhado nada mais é do que uma instrução if que é alvo de outro if ou else. São muito utilizados, pois permitem uma nova seleção baseada em uma seleção anterior. A Figura 24 apresenta um exemplo de utilização de ifs aninhados.

Figura 24 – Exemplo de ifs aninhados.

```

6 public class ClasseExemplo {
7
8     public static void main(String[] args) {
9         int n1, n2;
10        Scanner entrada = new Scanner(System.in);
11        System.out.println("Digite o primeiro número: ");
12        n1 = entrada.nextInt();
13
14        System.out.println("Digite o segundo número: ");
15        n2 = entrada.nextInt();
16
17        if (n1 == n2)
18            System.out.println("Os números são iguais.");
19        else {
20            if (n1 > n2 )
21                System.out.println("N1 é maior que N2.");
22            else
23                System.out.println("N2 é maior que N1.");
24        }
25    }
26 }
27

```

Problems Javadoc Declaration Search Console
 <terminated> ClasseExemplo (1) [Java Application] C:\Program Files\Java\jre1.8.0_24
 Digite o primeiro número:
 15
 Digite o segundo número:
 25
 N2 é maior que N1.

Fonte: o autor.

If-else-if

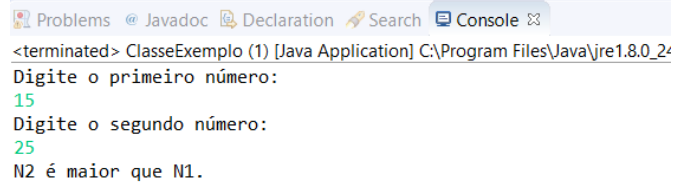
Em nossos programas, podemos ter situações em que precisamos avaliar mais do que uma condição em nossas estruturas de seleção/decisão. Para isso temos o if-else-if. Nessa estrutura, as condições são avaliadas de cima para baixo. Quando uma condição verdadeira é encontrada, a instrução associada a ela é executada e o resto é ignorado. Se nenhuma das condições for verdadeira, o else final será executado. Se nenhuma das condições for verdadeira e não houver um else final implementado, não será feita ação alguma. A Figura 25 apresenta um exemplo de um programa de cálculo de IMC (Índice de massa corporal) que bem representa o uso da estrutura if-else-if.

Figura 25 – Exemplo de if-else-if.

```

6 public class ClasseExemplo {
7
8     public static void main(String[] args) {
9         int n1, n2;
10        Scanner entrada = new Scanner(System.in);
11        System.out.println("Digite o primeiro número: ");
12        n1 = entrada.nextInt();
13
14        System.out.println("Digite o segundo número: ");
15        n2 = entrada.nextInt();
16
17        if (n1 == n2)
18            System.out.println("Os números são iguais.");
19        else {
20            if (n1 > n2 )
21                System.out.println("N1 é maior que N2.");
22            else
23                System.out.println("N2 é maior que N1.");
24        }
25    }
26 }
27

```



 <terminated> ClasseExemplo (1) [Java Application] C:\Program Files\Java\jre1.8.0_24

 Digite o primeiro número:

 15

 Digite o segundo número:

 25

 N2 é maior que N1.

Fonte: o autor.

Referências

Allan, Introdução ao Java Virtual Machine (JVM), DevMedia, 2013. Disponível em: <<https://www.devmedia.com.br/introducao-ao-java-virtual-machine-jvm/27624>>.

Acessado em 01/07/2020.

SCHILDt, H.; SKIREN, Dale. *Programação em Java: uma Introdução Abrangente*. São Paulo: McGrawHill Education, 2013.