



INSTITUTO DE GESTÃO E  
TECNOLOGIA DA INFORMAÇÃO

---

**Java**

---

**Bootcamp Programador de Software Iniciante**

João Paulo Barbosa Nascimento

2021

## Java

### Bootcamp: Programador de Software Iniciante

João Paulo Barbosa Nascimento

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

004.655.3

Nascimento, João Paulo Barbosa.

Java [recurso eletrônico] / João Paulo Barbosa Nascimento. –  
Dados eletrônicos. Belo Horizonte, 2021.  
30p.: il.

Apostila.

1. Java (Linguagem de programação). I. Título.

## Sumário

---

Capítulo 1.	Introdução .....	5
	Linguagem tipada e não tipada.....	5
	Introdução ao Java .....	7
	A JVM (Java Virtual Machine).....	8
	Ambientes de programação .....	10
	Instalando a IDE .....	11
	Desenvolvimento, compilação e execução .....	11
Capítulo 2.	Introdução aos tipos de dados e operadores .....	13
	O famigerado “Olá Mundo” .....	13
	Tipos de dados primitivos .....	14
	O operador de atribuição .....	15
	Manipulando dados primitivos.....	16
	O tipo String .....	16
	Manipulando String .....	17
	Métodos Print, Println e Printf .....	17
	Operadores aritméticos.....	18
	Trabalhando com operadores aritméticos.....	18
	Operadores lógicos.....	22
	Contadores e acumuladores .....	23
Capítulo 3.	Instruções de controle de programa.....	25
	Leitura de dados do teclado.....	25

A instrução if .....	25
Ifs aninhados.....	26
If-else-if .....	27
Switch .....	28
O laço For .....	29
O laço While.....	30
O laço do-while .....	31
Os comandos Break e Continue .....	31
Leitura de Arquivos .....	32
Gravação de Arquivos .....	33
 Capítulo 4. Orientação a Objetos .....	 35
Classes e Objetos.....	35
Atributos e Métodos .....	35
Construtores .....	37
Encapsulamento .....	37
 Referências.....	 38

## Capítulo 1. Introdução

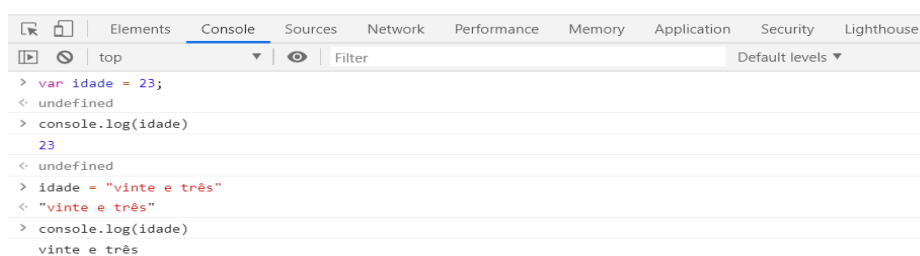
Neste capítulo, iremos realizar uma introdução à linguagem de programação Java, abordando alguns conceitos fundamentais, tais como: uma comparação entre linguagens tipadas e não tipadas, o surgimento da linguagem Java, a JVM (Java Virtual Machine), o ambiente de programação e o conceito de tempo de desenvolvimento,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



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 }
```

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, tendo inicialmente o nome de Oak. Posteriormente, 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 em 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). 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 dessa dificuldade, o Java surgiu como uma grande alternativa para a portabilidade das aplicações, pois a grande inovação por trás dessa 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.

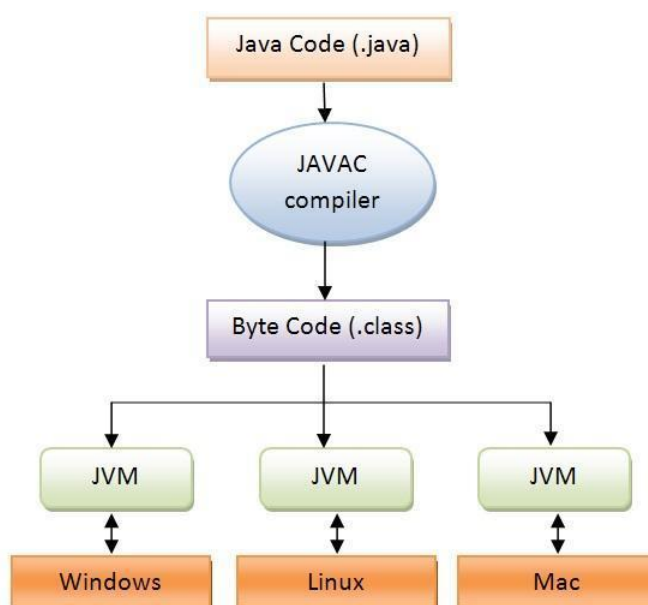
Em uma linguagem convencional, como C++ ou Pascal, o programa é compilado para uma determinada CPU ou sistema operacional, como Windows, Unix ou Linux. Para que possamos executar esse programa em um ambiente específico, é preciso que executemos novamente o processo de compilação, utilizando um compilador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

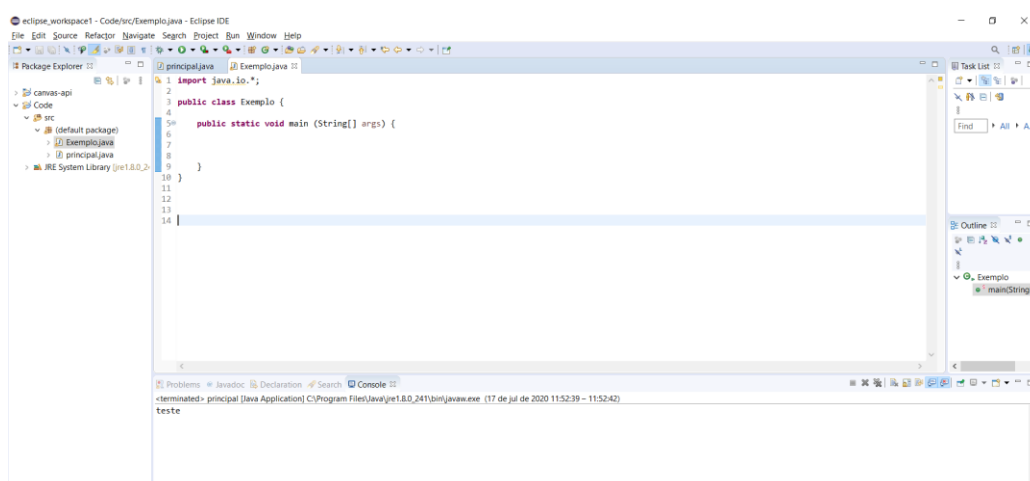
## Ambientes de programação

O principal componente presente no ambiente de programação é a IDE (Integrated Development Environment), 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 obrigatória 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.**



## 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, na qual 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

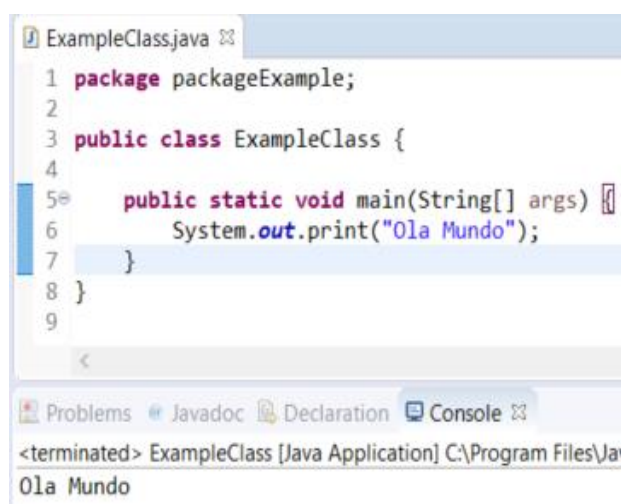
Neste capítulo, destacaremos os tipos de dados fornecidos pela linguagem Java para a construção dos nossos programas. Os dados primitivos (Inteiros, ponto-flutuante, booleano e caractere) serão apresentados. Os operadores de atribuição, aritméticos e lógicos serão apresentados. Por último, será apresentado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



```

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

```

The screenshot shows a Java IDE with a file named 'ExampleClass.java'. The code defines a package 'packageExample', a public class 'ExampleClass', and a public static void method 'main' that takes a String array 'args' and prints 'Ola Mundo' to the console. Below the code editor, the 'Console' tab is active, showing the output 'Ola Mundo'.

## 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 }
```

## 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 }
```

## 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 a 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 }
```

## 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 uma 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.**

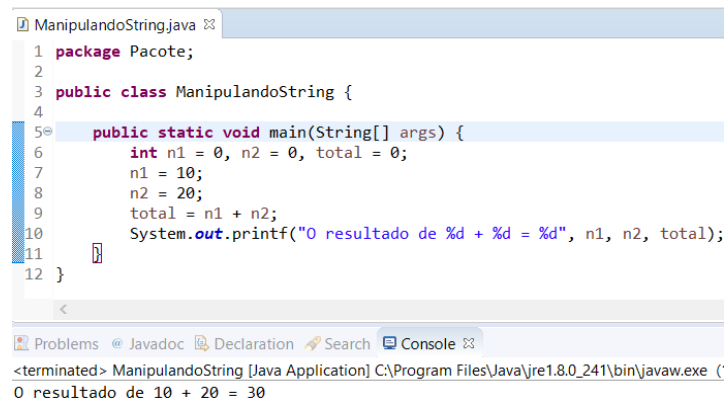
Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo
++	Incremento
--	Decremento

## 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.**



```

ManipulandoString.java
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("O resultado de %d + %d = %d", n1, n2, total);
11    }
12 }

```

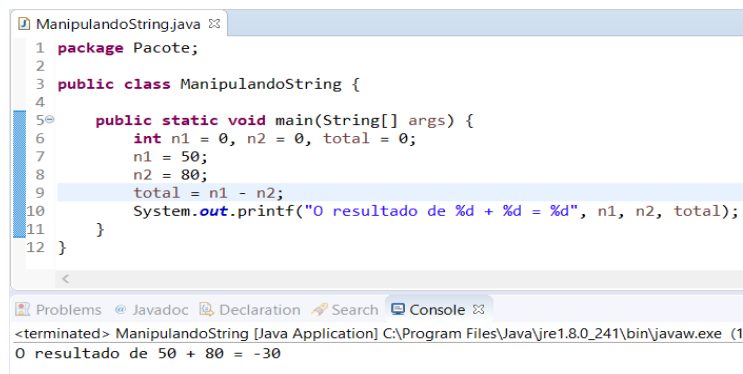
Problems Javadoc Declaration Search Console

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

O resultado de 10 + 20 = 30

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.**



```

ManipulandoString.java
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("O resultado de %d + %d = %d", 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 (1

O resultado de 50 + 80 = -30

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

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.**

```

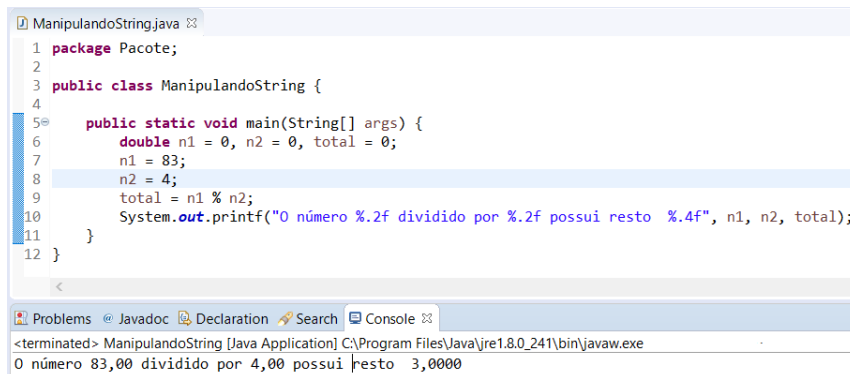
1 package Pacote;
2
3 public class ManipulandoString {
4
5     public static void main(String[] args) {
6         int n1 = 0, n2 = 0, total = 0;
7         n1 = 2;
8         n2 = 3;
9         total = n1 * n2;
10        System.out.printf("O número %d multiplicado por %d é igual a | %d", n1, n2, total);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

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.**



```

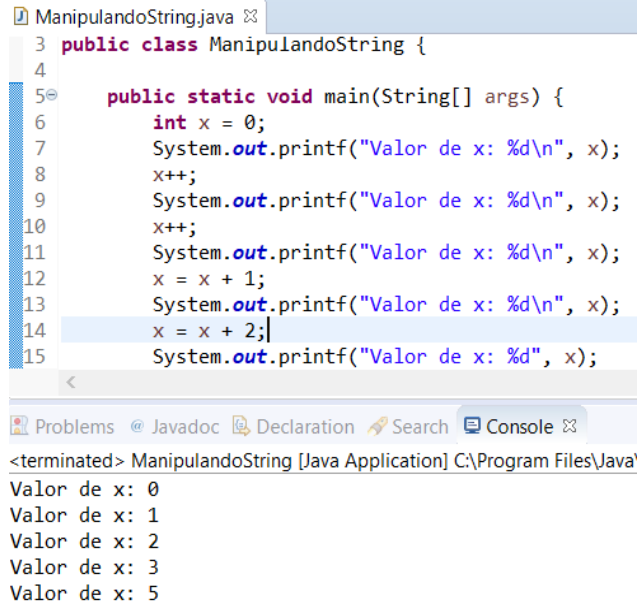
1 package Pacote;
2
3 public class ManipulandoString {
4
5     public static void main(String[] args) {
6         double n1 = 0, n2 = 0, total = 0;
7         n1 = 83;
8         n2 = 4;
9         total = n1 % n2;
10        System.out.printf("0 número %.2f dividido por %.2f possui resto %.4f", n1, n2,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

```

<terminated> ManipulandoString [Java Application] C:\Program Files\Java\jre1.8.0\_241\bin\javaw.exe  
0 número 83,00 dividido por 4,00 possui resto 3,0000

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.**



```

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);

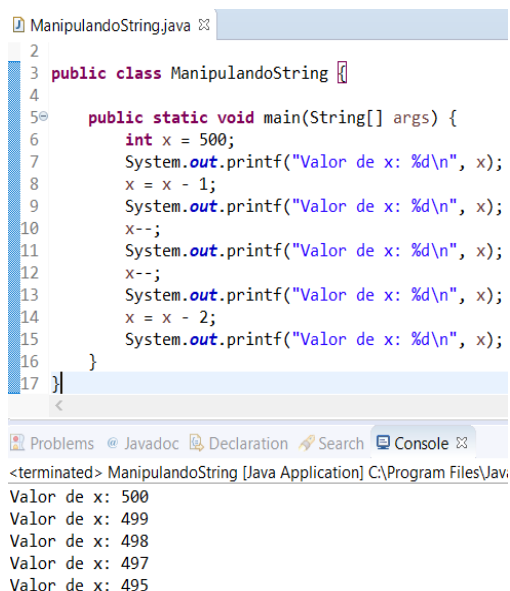
```

<terminated> ManipulandoString [Java Application] C:\Program Files\Java\jre1.8.0\_241\bin\javaw.exe  
Valor de x: 0  
Valor de x: 1  
Valor de x: 2  
Valor de x: 3  
Valor de x: 5

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 }

```

Problems Javadoc Declaration Search Console

<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

## 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

Os operadores lógicos são os valores verdadeiro ou falso que podem resultar de uma operação E, OU e Não, que podem ser aplicadas às 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

```

## 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. À 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.

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 }

```

Problems Javadoc Declaration Search Console

<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.



## 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`, `ifelse`, `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 executado. 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 ifelse.**

```

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

## 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.

## 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

```

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.

## Switch

Assim como a estrutura de seleção IF, o Switch é uma alternativa que fornece uma ramificação em diversos caminhos dentro do seu código-fonte, ou seja, essa estrutura permite que o programa escolha um entre vários caminhos diferentes dentro da aplicação.

Durante o funcionamento da estrutura Switch, o valor de uma expressão é verificado diante de uma lista de constantes. Quando uma determinada ocorrência é encontrada, as instruções associadas a ela (bloco de código do escopo) são executadas.

A Figura 26 apresenta a utilização do Switch.

**Figura 26 – Exemplo de Switch.**

```

1 package pacote01;
2
3 import java.util.Scanner;
4
5 public class Classe01 {
6
7     public static void main(String[] args) {
8         char opcao;
9         Scanner ent = new Scanner(System.in);
10        opcao = ent.nextLine().charAt(0);
11        switch (opcao) {
12            case 'A':
13                System.out.println("Ativo");
14                break;
15            case 'I':
16                System.out.println("Inativo");
17                break;
18            case 'E':
19                System.out.println("Excluído");
20                break;
21            default :
22                System.out.println("Opção desconhecida");
23        }
24    }
25 }

```

Cada valor especificado nas instruções case deve ser uma expressão de constante exclusiva, ou seja, se tentarmos inserir expressões repetidas, será apresentado um erro de compilação. O tipo de cada valor deverá ser compatível com o tipo da expressão que o controla.

Por fim, ocorre a instrução default, que acontece quando nenhuma opção case corresponde à expressão. O default tem similaridade com a instrução else dos comandos ifs.

## O laço For

Os laços também são conhecidos por loop ou estruturas de repetição. Eles permitem que parte do seu código, que se encontra dentro de um escopo, seja repetido.

Podemos dizer que o laço for é utilizado quando sabemos o número de vezes que o nosso bloco de código deverá repetir. Podemos dizer ainda que o teste de saída do laço for se dá no início da estrutura, sendo assim uma estrutura de repetição com teste no início.

O laço for possui três partes principais: inicialização, condição e testes. A Figura 27 apresenta um exemplo de utilização da estrutura de repetição for.

**Figura 27 – Exemplo do For.**

```
1 package pacote01;
2
3 import java.util.Scanner;
4
5 public class Classe01 {
6
7     public static void main(String[] args) {
8         for (int i = 0; i < 10; i++) {
9             System.out.println("Valor de i: " + i);
10            System.out.println("Valor de i vezes 10: " + i * 10);
11        }
12    }
13 }
```

O laço For é controlado por uma variável de controle. No caso do nosso exemplo, essa variável é a *i*. Ela age como uma contadora que controla as iterações do laço. A condição é uma expressão booleana que determina se o laço deverá ou não ser repetido. Por fim, a variável de controle mudará seu valor sempre que o laço for repetido. O laço continuará sua execução enquanto a condição for verdadeira. No momento que ela for falsa, ele se encerrará.

## O laço While

---

O laço While também se trata de uma estrutura de repetição, porém é mais indicado para ser usado quando não sabemos exatamente quantas vezes a repetição ocorrerá. O laço vai repetindo enquanto uma determinada condição for verdadeira. Observe o fragmento de código-fonte apresentado na Figura 28. Nele podemos

perceber que o laço irá executar enquanto o usuário informar um valor negativo para a posição npessoas do vetor idades.

**Figura 28 – Exemplo do For.**

```
while (idades[npessoas] < 0) {  
    System.out.printf("Idade inválida. Digite a idade de %s: ", nomes[npessoas]);  
    idades[npessoas] = entNumeros.nextInt();  
}
```

## O laço do-while

---

O laço do-while, diferentemente dos laços For e While, verifica sua condição ao fim do laço. Com isso, temos a garantia que o laço do-while será executado pelo menos uma vez. A sintaxe geral do laço é:

**Figura 29 –Exemplo do do-while**

```
do {  
    instruções;  
} while (condição);
```

## Os comandos Break e Continue

---

O comando break é utilizado para sair imediatamente de um laço, mesmo que a condição de saída ainda não tenha sido satisfeita. Quando esse comando é encontrado, o laço imediatamente se encerra e o programa irá automaticamente para o comando posterior ao laço.

O continue é o comando utilizado para forçar o laço a executar uma iteração antecipada, ignorando sua estrutura de controle normal.

## Leitura de Arquivos

---

A volatilidade é algo que está presente na memória principal de um computador e, por isso, não é possível o armazenamento permanente das informações. Se por algum motivo o programa encerrar a execução ou o computador desligar, os dados serão imediatamente perdidos. A solução para esses casos é o armazenamento em memória secundária, que permite que os dados não sejam perdidos caso haja o encerramento do programa ou a interrupção do fornecimento de energia para a máquina. Podemos realizar esse tipo de armazenamento persistente em uma estrutura de dados denominada arquivo.

Inicialmente, vamos falar da leitura de arquivos que estão em disco (HD). Logo em seguida, falaremos da gravação dessas estruturas de dados.

O primeiro passo será identificar e criar um objeto que permitirá o acesso a um determinado arquivo que se encontra no disco. O objeto que abre o arquivo e permite a leitura é da classe `FileReader`. Assim como vimos no início da nossa disciplina o objeto `Scanner` que lê os dados do teclado e joga para uma variável, aqui nós temos o `FileReader` que lê os dados de um arquivo e também armazena em uma variável. Logo abaixo faremos a declaração de um objeto chamado arquivo do tipo `FileReader`. Notem que no construtor da classe passaremos o endereço físico desse arquivo no disco. A Figura 30 apresenta o exemplo de uso do `FileReader`. Lembre-se que é necessário realizar o `import` para que o `FileReader` possa funcionar (`java.io.FileReader`).

**Figura 30 – Exemplo do objeto `FileReader`.**

```
public class ClasseExemplo {  
    public static void main(String[] args) {  
        try {  
            FileReader arquivo = new FileReader("C:\\Users\\João\\arquivoDados.txt");  
        } catch {  
        }  
    }  
}
```



Para poder realizar a leitura linha a linha do arquivo, iremos utilizar o objeto `BufferedReader` (`java.io.BufferedReader`). É importante destacar que no construtor da classe `BufferedReader` precisamos passar o objeto `FileReader`, para que seja possível identificar o arquivo que será aberto e lido. A Figura 31 demonstra a criação do objeto `BufferedReader` (linha 13). Para percorrer o arquivo linha a linha, precisaremos de uma estrutura de repetição que possa iterar sobre as linhas desse arquivo enquanto novas linhas existirem. Como cada arquivo pode ter uma quantidade de linhas diferente e não temos essa informação de maneira prévia, a estrutura de repetição mais indicada seria a `while`. Na linha 17 fazemos a primeira leitura, ou seja, da primeira linha do arquivo. Se o arquivo estiver vazio, a variável `linha` na linha 17 terá valor nulo, o teste do `while` dará falso na linha 19 e o programa se encerrará. Se o arquivo tiver conteúdo, a variável terá valor e o `while` imprimirá o valor da primeira linha do arquivo e em seguida irá ler as próximas linhas por meio do método `readLine` até o final do arquivo, sempre imprimindo o conteúdo na tela.

**Figura 32 - Exemplo completo de leitura de um arquivo.**

```
12     try {
13         FileReader arquivo = new FileReader("C:\\Users\\João\\arquivoDados.txt");
14         BufferedReader lerArquivo = new BufferedReader(arquivo);
15
16         String linha;
17         linha = lerArquivo.readLine();
18
19         while (linha != null) {
20             System.out.printf("%s\n", linha);
21             linha = lerArquivo.readLine();
22         }
23
24     } catch (IOException e){
25         System.out.println("Erro lendo dados: " + e.getMessage());
26     }
```

## Gravação de Arquivos

O processo de gravação de arquivos é a ação de enviar dados da memória do programa para a persistência em disco.

Para o processo de gravação, iremos utilizar o objeto `FileWriter`, presente no `java.io.FileWriter`. Esse objeto permitirá manipular o arquivo e devemos passar o endereço do arquivo no disco como argumento para o construtor da classe. Adicionalmente, iremos utilizar também o objeto `PrintWriter`, presente no pacote `java.io`. Devemos passar como argumento para o construtor da classe `PrintWriter` o objeto `FileWriter` criado anteriormente. É importante destacar que o `PrintWriter` tem o método `printf`, similar ao já estudado anteriormente. A Figura 33 apresenta um exemplo completo de gravação de arquivos onde, por meio de uma estrutura de repetição `for`, iremos gravar 1000 linhas no arquivo.

**Figura 33 - Exemplo completo de gravação de um arquivo.**

```
try {
    FileWriter arquivo = new FileWriter("C:\\Users\\João\\saidaDados.txt");
    PrintWriter gravarArquivo = new PrintWriter(arquivo);

    for (int i = 0; i < 1000; i++) {
        gravarArquivo.printf("Valor de i: %d.", i);
    }

    I gravarArquivo.close();
    arquivo.close();
}
} catch (IOException e){
    System.out.println("Ocorreu um erro ao gravar o arquivo: " + e.getMessage());
}
```

## Capítulo 4. Orientação a Objetos

---

Neste capítulo iremos introduzir os conceitos básicos da orientação a objetos, por meio de exemplos de diferenciação de classes e objetos, atributos e métodos e também o encapsulamento.

### Classes e Objetos

---

Desde o início do curso estamos falando a todo momento as palavras classe e objeto, pois o Java é uma linguagem orientada a objetos. Durante as aulas anteriores utilizamos alguns exemplos das classes String e Scanner com alguns métodos básicos, porém essas classes são muito mais poderosas.

Uma classe é um modelo que define a forma de um objeto e podemos dizer que um objeto é uma instância de uma classe. A classe funciona como uma fôrma para a criação de objetos. Podemos destacar ainda que uma classe é uma abstração e somente quando objeto é instanciado (criado ou concebido) é que passará a existir a sua representação física na memória. Além disso, os objetos são compostos por duas características principais: dados e operações ou atributos e métodos.

### Atributos e Métodos

---

Nas classes temos os atributos e os métodos. Os atributos dizem respeito aos dados que um objeto pode armazenar e os métodos são as operações que podem ser aplicadas sobre esses dados, transformando-os.

A Figura 34 apresenta um exemplo de atributos para uma classe chamada DadosEndereco, que têm o objetivo de armazenar todas as informações sobre um determinado endereço, tais como logradouro, número, cidade etc.

**Figura 34 - Exemplo de atributos em uma classe.**

```

1 package pacotePrincipal;
2
3 public class DadosEndereco {
4
5     String logradouro;
6     int numero;
7     String complemento;
8     String cidade;
9     String estado;
10    String CEP;
11
12
13
14
15 }

```

Um método pode retornar ou não um valor. Além disso, eles podem aceitar ou não valores de entrada, que são chamados de argumentos. Após terminar a sua execução, o método retorna para o controle do programa principal, exatamente onde foi chamado. A Figura 35 apresenta um exemplo de um método chamado imprimirEndereco.

**Figura 35 - Exemplo de um método.**

```

3 public class DadosEndereco {
4
5     String logradouro;
6     int numero;
7     String complemento;
8     String cidade;
9     String estado;
10    String CEP;
11
12    String imprimirDadosEndereco() {
13        return "Rua " + logradouro + ", " + Integer.toString(numero) + ", " +
14            complemento + ". Cidade: " + cidade + ". Estado: " + estado +
15            ". CEP: " + CEP;
16    }

```

O método imprimirDadosEndereco não recebe nenhum valor como argumento (entrada), porém tem como retorno um tipo String que será o resultado da concatenação de todos os atributos da classe.

## Construtores

---

Um construtor é um método que é executado automaticamente no momento em que um objeto está sendo instanciado. Normalmente, o Java já traz um construtor por padrão, que não recebe nenhum argumento como entrada.

O construtor é obrigatório para qualquer classe e é utilizado para atribuir valores iniciais para os seus atributos. Ele não possui tipo de retorno, porém aceita valores de entrada (argumentos).

## Encapsulamento

---

O encapsulamento é uma característica da orientação a objetos que permite a proteção de dados de um objeto e que os dados sofram acessos e alterações indevidas. Isso é feito através de dois métodos, chamados *getters* e *setters*. Um método *get* tem por objetivo retornar o valor presente em um atributo e um *set* gravar um valor.

Com isso, temos um único caminho tanto para gravação quanto para leitura de dados dos atributos. As vantagens da utilização do encapsulamento dizem respeito a uma melhor manutenção do código, seu reuso e também um desenvolvimento mais simplificado, acertado e acelerado.

## 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: 30 set. 2021.

SCHILDT, H.; SKIREN, Dale. *Programação em Java: uma Introdução Abrangente*. São Paulo: McGrawHillEducation, 2013.