**ÍNDICE 1º bimestre**

Lista de Figuras 3

Capítulo 1 – Conceitos Básicos 4

1.1 Lógica 4

Definição de lógica 4

Lógica do dia-a-dia 4

Lógica de programação 4

1.2 Processamento de Dados 5

1.3 Algoritmo 5

Definição 5

Método para o desenvolvimento de um algoritmo 6

Tipos de algoritmo 6

Pseudocódigo – Estrutura 8

Teste de mesa 8

Comentários em pseudocódigo 8

1.4 Programa 8

1.5 Java – linguagem de programação 9

Definição 9

Princípios básicos de um ambiente java típico 9

Plataforma java 10

Instalando o ambiente java 10

Comentários em java 10

O método main 11

1.6 Exemplo 11

1.7 Exercícios de lógica 11

Capítulo 2 – Como construir um algoritmo 13

2.1 Exemplo de como fazer um café 13

Descrição Narrativa Seqüencial 13

Descrição Narrativa de Seleção 13

Descrição Narrativa de Repetição sem a estrutura de repetição 14

Descrição Narrativa de Repetição usando a estrutura de repetição 14

2.2 Outro exemplo de construção de algoritmo envolvendo lógica 14

2.3 Exercícios de construção de algoritmo envolvendo lógica 15

Capítulo 3 – Constantes, variáveis e tipos de dados 16

3.1 Tipos de dados 16

Pseudocódigo 16

Java 16

3.2 Constantes 17

Definição 17

Pseudocódigo 18

Java 18

3.3 Variáveis 18

Definição 18

Pseudocódigo 19

Java 19

3.4 Identificadores 20

Definição 20

Formato 20

3.5 Conversões entre tipos em java 21

3.6 Exemplos com constantes, variáveis e identificadores em pseudocódigo 21

3.7 Exemplos com constantes, variáveis e identificadores em Java 22

3.8 Exercícios com constantes, variáveis e identificadores 22

Capítulo 4 – Expressões Aritméticas e Lógicas 23

4.1 Operadores aritméticos 23

Pseudocódigo 23

Java 23

Prioridades 24

Operadores unários 24

4.2 Operadores relacionais 25

Pseudocódigo 25

Java 25

Prioridades 25

4.3 Operadores lógicos 26

Pseudocódigo 26

Tabela Verdade 26

Java 26

Prioridades 27

4.4 Prioridades entre Operadores 27

4.5 Exemplo com operadores 27

4.6 Exercícios com operadores 28

Capítulo 5 – Estrutura Seqüencial 30

5.2 Operador de atribuição (←) e comandos de atribuição 30

Pseudocódigo 30

Java 30

5.3 Comandos de entrada e saída 30

Pseudocódigo – comandos de entrada 31

Java – comandos de entrada 31

Pseudocódigo – comandos de saída 33

Java – comandos de saída 34

5.4 Blocos 35

Pseudocódigo 35

Java 36

5.5 Estrutura seqüencial 36

Pseudocódigo 36

Java 37

5.6 Funções de tratamento de caracteres em Java 38

5.7 Exemplos de estrutura seqüencial em pseudocódigo 39

5.8 Exemplos de estrutura seqüencial em Java 40

5.9 Exercícios de algoritmos seqüenciais 41

Capítulo 6 – Estrutura de Decisão 44

6.1 Estrutura de Decisão Simples (se/então) 44

Pseudocódigo 44

Java 45

6.2 Estrutura de Decisão Composta (se/então/senão) 45

Pseudocódigo 45

Java 46

6.3 Estrutura de Decisão Encadeada (se/ então ... / senão ...) 47

Pseudocódigo 47

Java 48

6.4 Exemplos de estrutura de decisão simples, composta e encadeada em pseudocódigo 50

6.5 Exemplos de estrutura de decisão simples, composta e encadeada em Java 53

6.6 Exercícios de estrutura de decisão simples, composta e encadeada 56

Capítulo 7 – Estrutura de Múltipla Escolha 60

7.1 Estrutura de Múltipla Escolha Simples 60

Pseudocódigo 60

Java 62

7.2 Estrutura de Múltipla Escolha Encadeada 65

Pseudocódigo 65

Java 66

7.3 Exemplos de Estrutura de Múltipla Escolha em pseudocódigo 69

7.4 Exemplos de Estrutura de Múltipla Escolha em Java 71

7.5 Exercícios de Estrutura de Múltipla Escolha Simples e Encadeada 74

Capítulo 8 – Estrutura de Repetição 80

8.1 Estrutura de Repetição Enquanto 80

Pseudocódigo 80

Java 81

Encadeamento 82

8.2 Faça/Enquanto 84

Pseudocódigo 84

Java 84

Encadeamento 85

8.3 Para 87

Pseudocódigo 87

Java 88

Encadeamento 88

8.4 Exemplo de Estrutura de Repetição Enquanto, Faça/Eqto. e Para em pseudocódigo 90

8.5 Exemplo de Estrutura de Repetição Enquanto, Faça/Enquanto e Para em Java 92

8.6 Exercícios de Estrutura de Repetição Enquanto, Faça/Enquanto e Para 96

# Lista de Figuras

**Figura 1** – Processamento de dados 5

**Figura 2** – Fluxograma da soma de dois números 6

**Figura 3** – Fluxograma 7

**Figura 4** – O tipo lógico em Java 16

**Figura 5** – O tipo real em Java 17

**Figura 6** – O tipo caracter em Java 17

**Figura 7** – O tipo inteiro em Java 17

**Figura 8** – O tipo **String** em Java 17

**Figura 9** – Operadores aritméticos básicos 23

**Figura 10** – Operadores aritméticos auxiliares 23

**Figura 11** – Operadores aritméticos auxiliares em Java 24

**Figura 12** – Prioridades entre operadores aritméticos 24

**Figura 13** – Operadores unários em Java 24

**Figura 14** – Operadores relacionais 25

**Figura 15** – Operadores relacionais em Java 25

**Figura 16** – Operadores lógicos 26

**Figura 17** – Tabela verdade 26

**Figura 18** – Operadores lógicos em Java 26

**Figura 19** – Prioridades entre operadores lógicos 27

**Figura 20** – Prioridades entre operadores lógicos 27

# Capítulo 1 – Conceitos Básicos

## 1.1 Lógica

### Definição de lógica

Segundo o dicionário Aurélio[[1]](#footnote-1), Lógica é:

"*Filos*. Na tradição clássica, aristotélico-tomista, conjunto de estudos que visam a determinar os processos intelectuais que são condição geral do conhecimento verdadeiro."

"*Filos*. Conjunto de estudos tendentes a expressar em linguagem matemática as estruturas e operações do pensamento, deduzindo-as de números reduzidos de axiomas, com a intenção de criar uma linguagem rigorosa, adequada ao pensamento científico tal como o concebe a tradição empírico-positivista; lógica matemática, lógica simbólica."

"*Inform*. Forma pela qual as assertivas, pressupostos e instruções são organizados em um algoritmo para implementação de um programa de computador."

### Lógica do dia-a-dia

Independentemente de qualquer definição teórica, basta observarmos nosso dia-a-dia para verificarmos que a lógica está presente em quase tudo o que fazemos ou pensamos.

Por exemplo, ao pegarmos uma chave que está dentro de uma gaveta fechada, precisamos antes abrir a gaveta para conseguir pegar a chave. Isso é lógica!

Se estamos conversando com alguém, precisamos organizar as palavras numa certa ordem para dizer uma frase que faça sentido, senão a outra pessoa não entenderá nada do que você está falando. Isso é lógica!

E se você quer caminhar, você sabe que precisa colocar um pé na frente do outro numa seqüência pois, se não o fizer, ficará parado no mesmo lugar. Isso é lógica!

Enfim, muito do que fazemos e/ou pensamos, na nossa vida, está relacionado à lógica.

### Lógica de programação

A lógica de programação nada mais é do que a lógica usada na programação de computadores, que segue as leis do pensamento e os processos do raciocínio, cujo objetivo é desenvolver técnicas que ajudem as pessoas a realizar uma tarefa com o mínimo de esforço.

Por exemplo, com o uso da lógica e da computação, hoje temos acesso ao correio eletrônico, mais conhecido por *email*, em que uma mensagem escrita pode chegar quase que instantaneamente ao seu destino, em qualquer lugar do mundo. Imagine como era antes do correio eletrônico? Quanto tempo demorava uma carta para chegar ao Japão, saindo do Brasil? E antes do avião? Quando tempo esta carta demorava para chegar de navio? E antes do navio? Havia alguma possibilidade dessa carta chegar ao seu destino?

## 1.2 Processamento de Dados

Na informática, o tratamento de uma informação refere-se à entrada de dados, ao processamento desses dados de entrada e à saída dos resultados desse processamento, numa seqüência como mostra a figura a seguir:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | Entrada |  | Processamento |  | Saída |  |
|  |  |  |  |  |  |  |
| **Figura 1** – Processamento de dados | | | | | | |

A entrada de dados, que pode ser uma coleta de dados, uma depuração de dados ou um armazenamento de dados, recebe a informação necessária que será tratada no processamento de dados. Essa entrada pode ser feita através de dispositivos de entrada, por exemplo, teclado, mouse, etc.

O processamento de dados, que pode ser aritmético ou lógico, trata a informação recebida na entrada de dados, gerando resultados. Esse processamento é efetuado dentro do computador, na unidade lógica e aritmética conhecida como ULA que fica dentro da CPU (Unidade Central de Processamento), o cérebro do computador.

A saída de dados, que pode ser uma coleta de resultados ou uma distribuição de resultados, é a entrega dos dados processados para o usuário. Essa saída pode ser feita através de dispositivos de saída, por exemplo, monitor, impressora, etc.

Exemplificando, considere um problema cujo objetivo é encontrar a solução da soma de dois números. A partir dessa informação, podemos verificar que nossa entrada de dados são os dois números dos quais queremos encontrar a soma. Continuando, percebemos que nosso processamento é realizar o cálculo aritmético da soma dos dois números fornecidos na entrada de dados. E, finalizando, temos como resultado desse processamento a soma realizada desses dois números.

## 1.3 Algoritmo

### Definição

Podemos definir algoritmo como um processo que reúne um conjunto de ações que são necessárias para tratar os dados de entrada e transformá-los em resultados para um determinado objetivo.

Nesse sentido, podemos considerar que uma receita de bolo é um algoritmo, pois a partir dos ingredientes (dados de entrada), realizamos várias ações como bater os ingredientes numa batedeira, colocar numa assadeira e depois assar por determinado tempo (processamento) para, no final, termos um bolo bem-feito (resultado de saída).

De qualquer forma, trataremos a construção de algoritmos lógicos computacionais, em que os dados de entrada, o processamento e os resultados de saída são dados virtuais no formato que o computador entende.

### Método para o desenvolvimento de um algoritmo

Para a construção de um bom algoritmo, devemos seguir os seguintes passos:

1. ler e compreender o problema para o qual será construído um algoritmo;
2. determinar quais serão a entrada de dados do seu algoritmo;
3. determinar quais as ações, lógicas e/ou aritméticas, que deverão ser realizadas no seu algoritmo, bem como, as restrições, se houver, para cada uma dessas ações;
4. determinar qual será a saída de resultados do seu algoritmo;
5. construir o algoritmo em um dos tipos de algoritmo descritos na próxima seção;
6. testar o algoritmo usando o teste de mesa descrito no final deste capítulo.

### Tipos de algoritmo

As formas mais importantes para a construção de algoritmos são: a descrição narrativa, o fluxograma e o pseudocódigo (ou português estruturado ou portugol). Para exemplificar cada um dos tipos de algoritmo, vamos considerar o seguinte problema: a soma de dois números.

***Descrição narrativa.*** A descrição narrativa nada mais é do que descrever, utilizando uma linguagem natural (no nosso caso, a língua portuguesa), as ações a serem realizadas no tratamento dos dados de entrada para os resultados de saída na resolução do problema proposto.

A vantagem desse tipo de algoritmo é que a linguagem natural não oferece o esforço do aprendizado, pois ela já é bem conhecida. A desvantagem é que a linguagem natural pode ter interpretações diferentes dependendo do ponto de vista de cada pessoa, podendo dificultar a transcrição para o programa.

Para o nosso exemplo, a descrição narrativa ficaria da seguinte forma:

* iniciar o algoritmo;
* receber dois números;
* somar esses dois números;
* apresentar a soma dos dois números;
* finalizar o algoritmo.

***Fluxograma.*** O fluxograma é a forma de descrever as ações a serem realizadas no tratamento dos dados de entrada para os resultados de saída usando uma representação simbólica preestabelecida, por exemplo, como os símbolos na Figura 3.

A vantagem desse tipo de algoritmo é o entendimento. Qualquer representação gráfica sempre é mais bem entendida do que uma representação por escrito. A desvantagem desse tipo de algoritmo é a necessidade do aprendizado de cada símbolo bem como seu significado. Além disso, um algoritmo descrito por um fluxograma não apresenta detalhes, podendo dificultar a transcrição para o programa.

Para o nosso exemplo, o fluxograma ficaria da seguinte forma:

|  |
| --- |
| início  fim  N1, N2  S = N1 + N2  S |
| **Figura 2** – Fluxograma da soma de dois números |

|  |  |
| --- | --- |
|  | Símbolo utilizado para determinar o início e o término do algoritmo. |
|  | Símbolo utilizado para determinar o sentido do fluxo de dados do algoritmo e utilizado para conectar os símbolos existentes. |
|  | Símbolo utilizado para determinar uma ação ou comando que pode ser um cálculo ou uma atribuição de valores. |
|  | Símbolo utilizado para determinar a saída de dados do algoritmo. Essa entrada de dados pode ser via dispositivos de entrada. |
|  | Símbolo utilizado para determinar a saída de dados do algoritmo. Essa saída de dados pode ser via monitor. |
|  | Símbolo utilizado para determinar uma decisão que indicará qual caminho será seguido no algoritmo. |
|  | Símbolo utilizado para determinar uma conexão entre partes de um mesmo algoritmo. |
| **Figura 3** – Fluxograma | |

***Pseudocódigo.*** O pseudocódigo é a forma de descrever as ações para a resolução de um problema proposto por meio de regras preestabelecidas.

A vantagem desse tipo de algoritmo é a possibilidade da transcrição quase que imediata desse algoritmo para uma linguagem de programação, bastando saber a sintaxe e a semântica dessa linguagem de programação. A desvantagem deste tipo de algoritmo é a necessidade do aprendizado de cada uma das regras do pseudocódigo bem como seu significado. A seguir, apresentaremos cada uma dessas regras do pseudocódigo para o melhor aprendizado.

Para o nosso exemplo, o pseudocódigo ficaria da seguinte forma:

**Algoritmo Soma**

**início\_algoritmo**

**declarar** N1, N2, S **numérico\_inteiro**;

**ler** (N1, N2);

S ← N1 + N2;

**mostrar** (S);

**fim\_algoritmo.**

### Pseudocódigo – Estrutura

O pseudocódigo é o tipo de algoritmo que exploraremos a partir de agora. Como vimos no exemplo anterior, o algoritmo escrito em pseudocódigo possui uma estrutura, ou seja, um nome, um início, a declaração de variáveis e constantes, a entrada de dados, o processamento desses dados, a saída de resultados e o fim.

Todo e qualquer algoritmo escrito em pseudocódigo precisa dessa estrutura para que o tratamento da informação seja apresentado da forma mais compreensível possível.

### Teste de mesa

Como vimos na seção de método para o desenvolvimento de um algoritmo, o último item refere-se a testar o algoritmo construído. Para testar um algoritmo, utilizamos o teste de mesa que verifica se o algoritmo realiza exatamente o que se espera dele.

Para realizar o teste de mesa em um algoritmo é muito simples, basta seguir, passo-a-passo, cada comando do algoritmo como se fosse o computador. Assim, no final do teste de mesa, você pode concluir se o algoritmo está correto ou não. Se não estiver, basta corrigir o algoritmo, refazer o teste de mesa e repetir esse processo até que você certifique-se de que o algoritmo esteja realmente correto.

Mesmo que você suponha que seu algoritmo esteja correto, você deve fazer o teste de mesa, pois erros imperceptíveis podem ser detectados. Não se corre o risco de ter um algoritmo que não retorna sempre o resultado esperado.

### Comentários em pseudocódigo

Os comentários em pseudocódigos são simbolizados com duas barras (//) seguidas do comentário. Para o nosso exemplo, os comentários podem ser colocados da seguinte forma:

// nome do algoritmo

**Algoritmo Soma**

**início\_algoritmo**

// declaração das variáveis

**declarar** N1, N2, S **numérico\_inteiro**;

// entrada de dados – leitura de dois números inteiros

**ler** (N1, N2);

// processamento de dados – cálculo da soma dos dois números

S ← N1 + N2;

// saída de resultados – resultado da soma dos dois números

**mostrar** (S);

**fim\_algoritmo.**

## 1.4 Programa

Um programa de computador é a codificação de um algoritmo numa linguagem de programação específica, por exemplo, Java. Podemos distingüir dois tipos de linguagens: de alto nível e de baixo nível.

As linguagens de baixo nível são as linguagens de máquina, aquelas que o computador entende. As linguagens de alto nível são as linguagens de programação que servem de intermediárias entre a linguagem natural e a linguagem de máquina.

Sempre que codificamos um algoritmo numa linguagem de programação, esse programa precisa ser "traduzido" para a linguagem entendida pela máquina. A esse processo chamamos interpretação ou compilação.

## 1.5 Java – linguagem de programação

### Definição

No fim da década de 80, um pequeno grupo de trabalho da Sun, liderado por James Gosling, pesquisava um projeto de programação cujo principal objetivo era facilitar a interface de programação entre aparelhos eletrônicos e seus usuários, criando vários mecanismos de programação semelhantes em aparelhos diferentes como microondas, batedeira, aparelho de som, secretária eletrônica, entre outros.

Em 1990, a equipe direcionou a pesquisa para a criação de uma nova linguagem de programação, inicialmente chamada *oak* com uma arquitetura neutra e capaz de comportar-se adequadamente em plataformas de trabalho diferentes.

Em 1993, com o surgimento da WWW (*World Wide Web*), a Sun percebeu que a nova linguagem desenvolvida pelo grupo liderado por James Gosling era ideal para a programação na *Internet* por ser uma arquitetura neutra e criada para várias plataformas.

Foi assim que surgiu a linguagem de programação Java, ou simplesmente Java, uma linguagem de programação atualmente bastante aceita pela comunidade de desenvolvedores de *software* e *Internet*. Com o Java, usuários da *Internet* e da WWW foram beneficiados pelo acesso seguro e os desenvolvedores de *software*, pela portabilidade.

As principais características do Java são: simples e familiar, orientada a objeto, distribuída, interpretada, robusta, segura, arquitetura neutra e portátil, alta performance, multitarefa e dinâmica.

### Princípios básicos de um ambiente java típico

Os sistemas Java normalmente possuem um ambiente, uma linguagem, uma interface de programas aplicativos Java e várias bibliotecas de classes. Além disso, os programas Java passam por cinco fases antes de serem executados.

Na primeira fase, devemos editar o arquivo fonte. Podemos fazer isso utilizando qualquer editor de textos como o bloco de notas do Windows ou usando qualquer editor de Java que vem juntamente dos ambientes de desenvolvimento integrado (*Integrated Development Environments - IDE*). No arquivo fonte, o código java deve ser escrito de acordo com sua sintaxe e semântica, além disso, deve ter pelo menos uma classe principal que contenha o método principal, que é o **void main()**. Vale lembrar que todo nome de arquivo fonte deve terminar com a extensão **.java**. Por exemplo, **teste.java**.

Na segunda fase, devemos utilizar o comando **javac** para compilar o programa. Com esse comando, o compilador Java traduz o programa Java para *bytecodes* – a linguagem entendida pelo interpretador Java. Por exemplo, para compilar um programa chamado **teste.java** numa janela do DOS, devemos digitar, a seguir do *prompt* do DOS, o seguinte comando: **javac teste.java**. Se a compilação finalizar sem erros, um arquivo chamado **teste.class** será gerado. Este é o arquivo contendo os *bytecodes* que serão interpretados durante a fase de execução. Os *bytecodes* podem ser considerados instruções de máquinas para o JVM (*Java Virtual Machine*).

Na terceira fase, o programa deve ser colocado na memória primária antes de ser executado. Isso é feito pelo carregador de classe que copia o arquivo **.class** para a memória.

Na quarta fase, os *bytecodes* passam pelo verificador de *bytecodes* que verifica se o programa Java atende aos padrões de segurança de Java.

Na quinta fase, o programa é interpretado *bytecode* por *bytecode*, sob controle da CPU, realizando assim cada ação a ser executada pelo programa.

Para o arquivo **teste.java** compilado anteriormente, foi criado o arquivo **teste.class**, **teste** por ser o nome da classe principal do programa. Para executar esse arquivo **teste.class**, devemos digitar o seguinte comando no *prompt* do DOS: **java teste**. A partir daí, teremos o resultado do programa desenvolvido.

### Plataforma java

Por plataforma entendemos o conjunto de *hardware* e *software* no qual um programa é executado. Alguns exemplos de plataformas conhecidas são: Windows, Linux, MacOs, etc. Java também é uma plataforma, no entanto, sem *hardware*.

A plataforma Java tem dois componentes: JVM – *Java Virtual Machine* e Java API – *Java Application Programming Interface*. O Java API é uma coleção de componentes de *software* prontos que incluem desde estruturas para manipulação de arquivos até a construção de aplicativos gráficos. O Java API é organizado como um grupo de bibliotecas com classes e interfaces. Essas bibliotecas são chamadas de pacotes.

### Instalando o ambiente java

Para o desenvolvimento de aplicativos utilizando o Java, é necessária a instalação do compilador Java, do Java API e do JVM. A instalação do ambiente segue o mesmo esquema da instalação de qualquer produto para Windows. Devemos baixar a versão mais apropriada e fazer a instalação. Informações podem ser adquiridas nos seguintes sites: **java.sun.com**, **java.net**, **java.com**, entre outros.

### Comentários em java

O uso de comentários em programas é muito importante, seja na linguagem Java ou em qualquer outra linguagem de programação. Os comentários permitem explicar o conteúdo do código, para que serve, além de facilitar a manutenção posterior desse código. Em Java, podemos comentar o programa de duas formas: usando duas barras (//), onde todo o conteúdo à direita estará comentado; ou barra asterisco, asterisco barra (/\* \*/), onde todo o conteúdo entre os dois asteriscos estará comentado.

Por exemplo:

// Isso é um comentário usando duas barras.

/\* Isso também é um comentário.

Mas, usando barra asterisco e asterisco barra.

\*/

### O método main

No código fonte de qualquer programa Java, temos uma classe principal. Dentro dessa classe principal, devemos ter o método *main* que é o método principal de qualquer programa Java e sem esse método o programa não funciona. Isso por que é pelo método *main* que o programa inicia sua execução.

A sintaxe do método *main* é a seguinte: **public static void main(String args [])** onde: **public** indica que o método *main* pode ser chamado por outro objeto, **static** indica que o método *main* é um método de classe e **void** indica que o método *main* não retorna nenhum valor. O método *main* aceita um único argumento: um vetor de caracteres (do tipo **String)** que armazena os valores digitados pelo usuário após o nome do programa.

## 1.6 Exemplo

/\* Primeiro programa em Java

Este código fonte será salvo no arquivo primeiro\_programa.java

Este programa serve para exemplificar um programa Java para os alunos de Lógica de Programação

\*/

public class primeiro\_programa

{

public static void main(**String** argumentos [])

{

System.out.print("Primeiro Programa de Lógica de   
 Programação");

System.exit(0);

} // fim do método main

} // fim da classe principal primeiro\_programa

## 1.7 Exercícios de lógica

1. Três homens estão em um restaurante. A conta é de R$ 30,00, então cada homem dá uma nota de R$ 10,00 para o garçom. O garçom pega a conta e entrega o pagamento para o gerente que decide dar um desconto de R$ 5,00 na conta. No meio do caminho de volta à mesa, o garçom desonestamente pega R$ 2,00 dos R$ 5,00 e devolve R$ 1,00 para cada um dos homens à mesa. Então cada homem pagou R$9,00, dando um total de R$ 27,00. O garçom ficou com R$ 2,00, os homens pagaram R$ 27,00, ou seja, somando esses dois valores, temos R$ 29,00. O que aconteceu com o outro R$ 1,00?

2. Qual número que quando adicionado de 5/4 dá o mesmo resultado de quando ele é multiplicado por 5/4?

3. Descubra e escreva o próximo número da série abaixo:

18, 46, 94, 63, 52, ?

4. Há um número de três dígitos a que, se você adicionar 7, ele divide exatamente por 7; se você adicionar 8, ele divide exatamente por 8; se você adicionar 9, ele divide exatamente por 9. Qual é esse número?

5. Três garotos juntos têm três balas. João tem 3 balas a mais do que José, mas 5 a menos que Pedro. Quantas balas tem cada um deles?

6. Você tem exatamente R$ 100,00 para comprar os seguintes alimentos cujos preços são para um quilograma: arroz R$ 0,59, feijão R$ 1,99, batata R$ 2,87 e carne R$ 3,44. Você deve gastar exatamente os R$ 100,00. Quantos quilos de cada alimento você deve comprar?

7. Toda soma de cubos consecutivos, começando de 1, é sempre um quadrado perfeito. Por exemplo:

13 + 23 = 9 = 32

13 + 23 + 33 = 36 = 62

13 + 23 + 33 + 43 = 100 = 102

e assim por diante. Entretanto, no problema proposto aqui, começar de 1 não é permitido. Você deve encontrar o menor quadrado perfeito que é igual à soma de no mínimo quatro cubos perfeitos consecutivos, excluída a unidade.

8. Um certo número, maior que 5000, mas menor que 50000, tem as seguintes propriedades: quando dividido por 4, tem um resto igual a 3; quando dividido por 5, tem resto igual a 1; quando dividido por 7, tem resto igual a 2; quando dividido por 11, tem resto igual a 2; e quando dividido por 17, tem resto igual a 12. Qual é esse número?

9. Todo número ímpar exceto dois entre 1 e 8999 ou é um primo ou é a soma de um primo e duas vezes um quadrado, por exemplo:

21 = 19 + 2 \* 12

27 = 19 + 2 \* 22

35 = 5 + 2 \* 52

Quais são as duas exceções?

10. Corredores em uma maratona são numerados começando de 1. Um dos corredores notou que a soma dos números menores que seu próprio número era igual à soma dos números maiores. Se havia mais do que 100 corredores, mas menos do que 1000, que número era do corredor acima e quantos corredores havia na corrida?

11. Em minha casa há muitos gatos: sete não comem peixe; seis não comem fígado; cinco não comem frango; quatro não comem peixe nem fígado; três não comem peixe nem frango; dois não comem fígado nem frango; um não come fígado, peixe nem frango; nenhum deles come todas as três comidas. Quantos gatos eu tenho?

12. Encontre a menor seqüência de números primos consecutivos cuja soma seja igual a 106620.

13. Três casais vivem felizes numa cidade. Com base nas dicas abaixo, tente descobrir o nome de cada marido (Cleber, Lucas, Pedro), a profissão de cada um (Pedreiro, Arquiteto, Escultor) e o nome de suas respectivas esposas (Laura, Priscila, Mariana):

a. O pedreiro é casado com Mariana.

b. Pedro é arquiteto.

c. Priscila não é casada com Pedro.

d. Cleber não é pedreiro.

14. No último verão, três grupos acamparam na floresta. Com base nas dicas abaixo, tente descobrir onome do guia de cada acampamento (Heitor, Paulo, Sergio), o tipo de grupo que cada um liderou (crianças, jovens, adultos) e a cor das barracas que cada grupo de campista usou (amarelo, preto, vermelho).

a. Paulo foi o guia do grupo que acampou nas barracas de cor preto.

b. O animado grupo dos adultos ocupou as barracas de cor vermelho.

c. Heitor foi o grupo das crianças.

# Capítulo 2 – Como construir um algoritmo

## 2.1 Exemplo de como fazer um café

Vamos escrever vários algoritmos de como fazer um café utilizando o algoritmo de descrição narrativa. Vamos perceber que, por mais fácil que seja a tarefa, muitas vezes não nos atentamos para os pequenos detalhes da execução dessa tarefa, mas que podem fazer falta para alcançar o resultado desejado. Vamos ver como fica um algoritmo seqüencial para nosso exemplo:

### Descrição Narrativa Seqüencial

* iniciar algoritmo;
* pegar bule;
* colocar coador de plástico sobre o bule;
* colocar coador de papel sobre o coador de plástico;
* colocar café tostado e moído sobre o coador de papel;
* colocar água sobre o café;
* finalizar algoritmo.

Nessa descrição, seguimos uma seqüência de vários passos, um após o outro. Isso significa que as ações foram executadas linearmente, da direita para a esquerda e de cima para baixo. Esse algoritmo mostra como qualquer pessoa, naturalmente, faria um café, sem se importar com qualquer detalhe.

Agora, analisando o algoritmo, verificamos que o resultado a ser alcançado é bem definido: fazer um café. No entanto, e se a água não estivesse fervente? Como sairia o café? O café seria feito com a água fria porque não foi prevista essa condição para o algoritmo.

Para resolver esse problema, podemos fazer uma seleção para verificar se a água está fervente ou não. Assim, temos o seguinte algoritmo de seleção para o nosso exemplo:

### Descrição Narrativa de Seleção

* iniciar algoritmo;
* pegar bule;
* colocar coador de plástico sobre o bule;
* colocar coador de papel sobre o coador de plástico;
* colocar café tostado e moído sobre o coador de papel;
* **se** a água estiver fervente, **então**
* colocar água sobre o café;
* finalizar algoritmo.

Nessa descrição, fazemos uma seleção para a água, ou seja, verificamos se a água está fervente. Em caso afirmativo, a água é colocada sobre o café. Se a seleção for falsa (a água não estiver fervente), o café não será feito.

Nesse algoritmo, colocamos uma seleção para verificar qual o caminho que o algoritmo deve seguir. Perceba que, dependendo do resultado da seleção, algumas ações do algoritmo podem ser realizadas ou não.

Analisando o algoritmo, percebemos que o café seria feito somente se a água estivesse fervente. Como podemos garantir que o resultado do algoritmo, que é ter o café pronto, seja alcançado? Para resolver esse problema, podemos fazer uma repetição para verificar a condição da água e quando ela estiver fervente ser colocada sobre o café. Assim, temos o seguinte algoritmo de repetição para o nosso exemplo:

### Descrição Narrativa de Repetição sem a estrutura de repetição

* iniciar algoritmo;
* pegar bule;
* colocar coador de plástico sobre o bule;
* colocar coador de papel sobre o coador de plástico;
* colocar café tostado e moído sobre o coador de papel;
* **se** a água não estiver fervente, **então**
* aquecer a água;
* **se** a água não estiver fervente, **então**
* continuar aquecendo a água;
* **se** a água não estiver fervente, **então**
* continuar aquecendo a água;

...

continua até quando?

* colocar água sobre o café;
* finalizar algoritmo.

Percebemos que o algoritmo não está terminado, pois falta saber quando a água estará fervendo para ser colocada sobre o café. Para simplificarmos nosso algoritmo, podemos reescrevê-lo utilizando uma condição de parada para determinar um limite para esse algoritmo:

### Descrição Narrativa de Repetição usando a estrutura de repetição

* iniciar algoritmo;
* pegar bule;
* colocar coador de plástico sobre o bule;
* colocar coador de papel sobre o coador de plástico;
* colocar café tostado e moído sobre o coador de papel;
* **enquanto** a água não estiver fervente, **faça**
* aquecer a água;
* colocar água sobre o café;
* finalizar algoritmo.

Note que as duas descrições narrativas de repetição retornam o mesmo resultado que é aquecer a água até ela ferver e ter o café pronto. A única diferença é que na segunda descrição utilizamos uma forma mais adequada para descrever um conjunto de ações em repetição.

## 2.2 Outro exemplo de construção de algoritmo envolvendo lógica

Considere a Torre de Hanói com três hastes e três discos. Uma das hastes serve de suporte para três discos de tamanhos diferentes de forma que os menores estão sobre os maiores. Desenvolva um algoritmo que mova todos os discos da haste onde se encontram para uma das outras duas hastes da Torre de Hanói, seguindo as seguintes regras: Pode-se mover um disco de cada vez para qualquer haste, contanto que um disco maior nunca seja colocado sobre um disco menor.

Resolução:

Considerando discos 1, 2, 3 e hastes A, B, C, temos o seguinte resultado:

* mover o disco 1 da haste A para a haste B;
* mover o disco 2 da haste A para a haste C;
* mover o disco 1 da haste B para a haste C;
* mover o disco 3 da haste A para a haste B;
* mover o disco 1 da haste C para a haste A;
* mover o disco 2 da haste C para a haste B;
* mover o disco 1 da haste A para a haste B.

## 2.3 Exercícios de construção de algoritmo envolvendo lógica

Utilizando a descrição narrativa, construa algoritmos para os seguintes exercícios:

1. Um homem com suas três cargas – um lobo, um bode e um maço de alfafa – precisa atravessar um rio usando um barco que possui capacidade para atravessar somente o homem e mais uma de suas três cargas. Considerando que o lobo come o bode e o bode come o maço de alfafa, quais os percursos que o homem deve fazer para atravessar o rio sem perder nenhuma de suas cargas? Desenvolva um algoritmo indicando todos os percursos efetuados pelo homem para atravessar o rio sem perder nenhuma das cargas.

2. A partir do exemplo da Seção 2.2, amplie a solução apresentada, de maneira a completar a operação descrita, de troca dos discos entre hastes, considerando a existência de 3 hastes e 4 discos.

3. Três jesuítas e três canibais precisam atravessar um rio com um barco que possui capacidade para transportar somente duas pessoas. Considerando que em qualquer margem, a quantidade de jesuítas não seja menor que a de canibais, quais os percursos que devem ser feitos para os 3 jesuítas e os 3 canibais atravessarem o rio? Desenvolva um algoritmo indicando todos os percursos efetuados para todos atravessarem o rio com segurança.

4. Tendo como exemplo os algoritmos desenvolvidos para solucionar o problema de fazer um café, elabore três algoritmos que mostrem os passos necessários para trocar um pneu furado. Considere o seguinte conjunto de situações, um para cada algoritmo:

a) trocar o pneu dianteiro direito;

b) verificar se o pneu reserva está em condições de uso; em caso afirmativo, trocar o pneu dianteiro direito;

c) verificar se existe algum pneu furado; em caso afirmativo, verificar se o pneu reserva está em condições de uso e, em caso afirmativo, trocar o pneu correto.

Para cada algoritmo desenvolvido, introduza novas ações e altere o fluxo do algoritmo de forma compatível com as situações apresentadas.

5. Considere uma calculadora comum de quatro operações. Considere que as teclas de divisão e multiplicação estão quebradas. Desenvolva três algoritmos que resolvam expressões matemáticas de multiplicação, divisão e exponenciação usando apenas as operações de adição e subtração.

# Capítulo 3 – Constantes, variáveis e tipos de dados

A partir deste capítulo, estaremos preocupados em conhecer as partes necessárias para se escrever um algoritmo completo em pseudocódigo. Para isso, neste capítulo, vamos estudar como as constantes, os tipos de dados e as variáveis são utilizados em pseudocódigo.

## 3.1 Tipos de dados

Os tipos de dados são as características comuns dos dados a serem manipulados. Podemos considerar quatro classificações para os tipos de dados: inteiro, real, caracter e lógico.

O **tipo inteiro** caracteriza qualquer dado numérico que pertença ao conjunto dos números inteiros. Por exemplo: -5, 0, 32. O tipo inteiro quando armazenado no computador ocupa 2 bytes, o que corresponde a 65536 possibilidades, ou seja, de –32767 até 32767.

O **tipo real** caracteriza qualquer dado numérico que pertença ao conjunto dos números reais. Por exemplo: -9.0, 0, 29.45. O tipo real quando armazenado no computador ocupa 4 bytes, o que corresponde a 232 possibilidades, podendo ter de 6 a 11 dígitos significativos com sinal.

O **tipo caracter** caracteriza qualquer dado que pertença a um conjunto de caracteres alfanuméricos e são simbolizados por entre aspas duplas (""). Por exemplo: "15", "Eu", "Pare!", "?%@". O tipo caracter quando armazenado no computador ocupa 1 byte para cada caracter.

O **tipo lógico** caracteriza qualquer dado que possa assumir somente duas situações: verdadeiro ou falso. Por exemplo: feminino ou masculino, loja aberta ou fechada, brasileiro ou estrangeiro. O tipo lógico quando armazenado no computador ocupa 1 byte, pois possui apenas duas formas de representação.

### Pseudocódigo

No pseudocódigo, há nomes especiais para representar cada tipo de dados, como se segue:

* para o tipo inteiro, usaremos **numérico\_inteiro**;
* para o tipo real, usaremos **numérico\_real**;
* para o tipo caracter, usaremos **alfanumérico**;
* para o tipo lógico usaremos **lógico**.

### Java

No Java também há nomes especiais, chamados palavras reservadas, para representar cada tipo de dado, como segue:

|  |  |  |  |
| --- | --- | --- | --- |
| * para o tipo lógico, temos: | | | |
|  | **boolean** | tipo lógico que só pode assumir **true** ou **false**  intervalo compreendido: ocupa 1 bit |  |
|  | | | |
| **Figura 4** – O tipo lógico em Java | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| * para o tipo real, temos dois tipos diferentes, dependendo da capacidade de armazenamento: | | | |
|  | **float** | números reais de 32 bits  intervalo compreendido: 3.4E-038 ... 3.4E+038 |  |
| **double** | números reais com precisão dupla  intervalo compreendido: 1.7E-308 ... 1.7E-308 |
|  | | | |
| **Figura 5** – O tipo real em Java | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| * para o tipo caracter que representa um único caracter, temos: | | | |
|  | **char** | caracter que ocupa 16 bits (sem sinal)  intervalo compreendido: 0 ... 65536  inclui unicode (alfabetos latinos, grego, cirílico, etc.) |  |
|  | | | |
| **Figura 6** – O tipo caracter em Java | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| * para o tipo inteiro, temos quatro tipos diferentes, dependendo da capacidade de armazenamento | | | |
|  | **byte** | números inteiros de 8 bits  intervalo compreendido: -128 ... 127 |  |
| **short** | números inteiros de 16 bits  intervalo compreendido: -32.768 ... 32.767 |
| **int** | números inteiros de 32 bits  intervalo compreendido: -2.147.483.648 ... 2.147.483.647 |
| **long** | números inteiros de 64 bits  intervalo compreendido: -9.223.372.036.854.775.808 ...  9.223.372.036.854.775.807 |
|  | | | |
| **Figura 7** – O tipo inteiro em Java | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| * para o tipo caracter que representa uma cadeia de caracteres, temos: | | | |
|  | **String** | não é um tipo primitivo. É uma classe definida dentro de um pacote do Java |  |
|  | | | |
| **Figura 8** – O tipo **String** em Java | | | |

## 3.2 Constantes

### Definição

Dizemos que determinado dado é uma constante quando este dado não sofre nenhuma alteração, ou seja, ele é fixo. Por exemplo, podemos definir uma constante **pi** ← **3.14**, dentro do algoritmo e, nesse algoritmo, esse valor nunca sofrerá alteração.

### Pseudocódigo

A declaração de um dado constante em pseudocódigo terá a seguinte regra sintática:

**declarar**

**constante <nome da constante>** ← **<valor da constante> <tipo de dado da constante> ;**

Por exemplo:

**declarar**

**constante** pi ← 3.14 **numérico\_real**;

Podemos também declarar vários dados constantes em pseudocódigo numa mesma linha, se eles forem todos do mesmo tipo, seguindo a regra sintática abaixo e considerando NC como nome da constante e VC como valor da constante:

**declarar**

**constante <NC1>** ← **<VC1> , <NC2>** ← **<VC2> , ... , <NCn>** ← **<VCn> <tipo de dado> ;**

Por exemplo:

**declarar**

**constante** pi ← 3.14, x ← 9.3, telefone ← -7.23 **numérico\_real**;

### Java

A declaração de um dado constante em java terá a seguinte regra sintática:

**final <tipo de dado da constante> <nome da constante> = <valor da constante> ;**

Por exemplo:

**final double** pi = 3.14;

Podemos também declarar vários dados constantes em Java numa mesma linha, se eles forem todos do mesmo tipo, seguindo a regra sintática abaixo e considerando NC como nome da constante e VC como valor da constante:

**final <tipo de dado constante> <NC1> = <VC1> , <NC2> = <VC2> , ... , <NCn> = <VCn>;**

Por exemplo:

**final double** pi = 3.14, x = 9.3, telefone = -7.23;

## 3.3 Variáveis

### Definição

Dizemos que determinado dado é uma variável quando o mesmo pode sofrer alguma alteração, ou seja, ele é variável. Por exemplo, podemos definir uma constante **x**, dentro do algoritmo e, nesse algoritmo, esse valor poderá sofrer alteração.

### Pseudocódigo

A declaração de um dado variável em pseudocódigo terá a seguinte regra sintática:

**declarar**

**<nome da variável> <tipo de dado da variável> ;**

Por exemplo:

**declarar**

x **numérico\_inteiro**;

Podemos também declarar vários dados variáveis em pseudocódigo numa mesma linha se eles forem todos do mesmo tipo, seguindo a regra sintática abaixo e considerando NV como nome da variável:

**declarar**

**<NV1> , <NV2> , ... , <NVn> <tipo de dado da variável> ;**

Por exemplo:

**declarar**

telefone, idade, valor, cor **alfanumérico**;

No momento da declaração de um dado variável em pseudocódigo, podemos também inicializar a variável, assim a declaração com a inicialização terá a seguinte regra sintática:

**declarar**

**<nome variável>** ← **<valor de inicialização da variável> <tipo de dado da variável> ;**

Por exemplo:

**declarar**

y ← 5 **numérico\_real**;

### Java

A declaração de um dado variável em java terá a seguinte regra sintática:

**<tipo de dado da variável> <nome da variável> ;**

Por exemplo:

**int** x;

Podemos também declarar vários dados variáveis em Java numa mesma linha se eles forem todos do mesmo tipo, seguindo a regra sintática abaixo e considerando NV como nome da variável:

**<tipo de dado da variável> <NV1> , <NV2> , ... , <NVn>;**

Por exemplo:

**char** letra, tom, nome;

No momento da declaração de um dado variável em Java, podemos também inicializar a variável, assim a declaração com a inicialização terá a seguinte regra sintática:

**<tipo de dado da variável> <nome da variável> = <valor de inicialização da variável>;**

Por exemplo:

**double** y = 5;

## 3.4 Identificadores

### Definição

Os identificadores são os nomes dados às informações de um programa, por exemplo, nome da variável, nome da constante, nome do programa etc.

Por exemplo:

pi, x, num\_1, valor5, etc.

Os identificadores são únicos para uma determinada informação, por exemplo, um mesmo identificador não pode ser usado para o nome de uma variável e de uma constante, ou para o nome de duas variáveis de mesmo tipo ou de tipos diferentes. Uma vez que um identificador foi usado para uma determinada informação, ele não pode ser usado novamente para identificar outra informação.

Por exemplo, em pseudocódigo:

**declarar**

x **numérico\_inteiro**;

x **numérico\_real**;

Por exemplo, em Java:

**int** x ;

**double** x;

Esse exemplo mostra o que **não** pode acontecer, ou seja, o identificador **x** declarado como um tipo inteiro e depois como um tipo real. Um identificador pode possuir somente um tipo de informação.

### Formato

Esses identificadores devem seguir as seguintes regras para serem válidos:

* todos os caracteres devem ser letras, números ou o sublinhado (o único caracter especial permitido);
* o primeiro caracter deve ser uma letra;
* os demais caracteres podem ser letras, números ou sublinhado;
* não são permitidos caracteres especiais (com exceção do sublinhado);
* palavras reservadas (de uma linguagem de programação ou do pseudocódigo) não devem ser usadas como identificadores.

## 3.5 Conversões entre tipos em java

Em Java, as conversões de tipos menos significativos para tipos mais significativos, por exemplo, de inteiro para real, são feitas automaticamente e sem problemas, basta fazer a sua atribuição, por exemplo:

// declaração da variável x do tipo **int** inicializado com o valor 10

**int** x = 10;

// declaração da variável y do tipo **double**

**double** y;

// atribuição do valor de x para y, a conversão de **int** para **double** é feita automaticamente

y = x;

No entanto, às vezes, queremos fazer atribuições de tipos mais significativos para tipos menos significativos, por exemplo, de real para inteiro. Nesse caso, precisamos fazer a conversão do tipo real para inteiro antes de fazer qualquer atribuição, por exemplo:

// declaração da variável i do tipo **int**

**int** i;

// declaração da variável j do tipo **double** inicializado com o valor 5.2

**double** j = 5.2;

// atribuição do valor de j, já convertido para **int**, para i

i = (**int**) j;

Essa operação de conversão entre tipos é chamada de *cast*. As conversões de tipos menos significativos para tipos mais significativos permitidas são:

**byte** → **short** → **int** → **long** → **float** → **double** e **char** → **int**

Todas as demais conversões entre tipos devem ser convertidas com o uso de operações de *cast*.

## 3.6 Exemplos com constantes, variáveis e identificadores em pseudocódigo

1. São identificadores válidos:

num1, item5C, tipo\_a2, A123, idade, telefone

2. São identificadores não válidos:

1num, 5Citem, \_tipoa2, \_2valor, 123A, R$, nota/2, 5\*, x&y

3. Declaração de constantes:

**declarar**

**constante** num1 ← 9 **numérico\_inteiro**;

**constante** x\_2 ← -3.28, cor ← 7.4 **numérico\_real**;

4. Declaração de variáveis

**declarar**

um ← 5, dois **numérico\_inteiro**;

maria, vlr9 **numérico\_real**;

valor\_4A ← "alfabeto" **alfanumérico**;

sexo, X987 **lógico**;

amarelo ← '5', cor **alfanumérico**;

## 3.7 Exemplos com constantes, variáveis e identificadores em Java

1. São identificadores válidos:

num1, item5C, tipo\_a2, A123, idade, telefone

2. São identificadores não válidos:

1num, 5Citem, \_tipoa2, \_2valor, 123A, R$, nota/2, 5\*, x&y

3. Declaração de constantes:

**final int** num1 = 9;

**final double** x\_2 = -3.28, cor = 7.4;

4. Declaração de variáveis

**int** um = 5, dois;

**double** maria, vlr9;

**String** valor\_4A = "alfabeto";

**boolean** sexo, X987;

**char** amarelo = '5', cor;

## 3.8 Exercícios com constantes, variáveis e identificadores

1. Determine qual o tipo de dado que aparece nas seguintes frases:

a) A placa de trânsito "E" significa estacionamento permitido.

b) Hoje a temperatura em graus Celsius atingiu 45 graus.

c) A prova de Lógica de Programação estava fácil e a menor nota foi 6.5.

d) O sexo do meu primeiro filho é feminino.

2. Assinale os identificadores válidos:

a) A={1,2,3} b) M2 c) PARE! d) "MARIA" e) email@email

f) D/YU g) Avenida h) A\*34 i) M(I) j) Teste\_1

l) 5mar m) KJO&U n) LKJHHG o) KhMjh p) x<y

q) US$ r) df\_2\_9\_df s) num-4 t) CEP u) abelha

3. Considere as seguintes variáveis: Nome, RA, Nota e Sexo. Suponha que essas variáveis sejam utilizadas para armazenar o nome do aluno, o registro de aluno, a nota do aluno e o sexo do aluno, respectivamente. Declare as variáveis corretamente, associando o tipo de dado adequado que será armazenado para cada informação.

4. Encontre os erros da seguinte declaração de variáveis:

**Declarar**

Cor, X **lógico**;

XLM, A, tamanho, R$ **numérico\_real**;

X, Idade **alfanumérico**;

Nome, sexo **numérico\_inteiro**;

# Capítulo 4 – Expressões Aritméticas e Lógicas

Uma expressão aritmética é um conjunto de operadores e operandos dispostos numa determinada ordem. Nesse caso, os operadores podem ser aritméticos e os operandos podem ser constantes ou variáveis inteiras ou reais. O resultado de uma expressão aritmética será sempre numérica.

Uma expressão lógica também é um conjunto de operadores e operandos dispostos numa determinada ordem. Nesse caso, os operadores podem ser relacionais ou lógicos e os operandos podem ser relações, constantes ou variáveis inteiras, reais, alfanuméricas ou lógicas. O resultado de uma expressão lógica será sempre do tipo lógico, ou seja, verdadeiro ou falso.

## 4.1 Operadores aritméticos

### Pseudocódigo

Os operadores aritméticos pertencem a um conjunto de símbolos que representam as operações matemáticas básicas, que são:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
|  | **Operador** | **Função** | **Exemplo** |  |
| + | adição | 5 + 3 |
| – | subtração | 2 – a |
| \* | multiplicação | b \* c |
| / | divisão | d / 10 |
|  | | | | |
| **Figura 9** – Operadores aritméticos básicos | | | | |

Além dos operadores aritméticos básicos, consideraremos, para pseudocódigo, operadores aritméticos auxiliares para outros cálculos, que são:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
|  | **Operador** | **Função** | **Exemplo** | **Resultado** |  |
| pot | potenciação | pot(2,3) | 23 = 8 |
| rad | radiciação | rad(9) | = 3 |
| mod | resto da divisão | 7 mod 3 | resto da divisão = 1 |
| div | quociente da divisão | 7 div 3 | quoc. da divisão = 2 |
|  | | | | | |
| **Figura 10** – Operadores aritméticos auxiliares | | | | | |

### Java

Os operadores aritméticos básicos em Java são os mesmos apresentados na Figura 9. Os operadores aritméticos auxiliares seguem a simbologia própria do java, como se segue:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
|  | **Operador** | **Função** | **Exemplo** |  |
| Math.pow | potenciação | Math.pow(2,3) |
| Math.sqrt | radiciação | Math.sqrt(9) |
| % | resto da divisão | 7 % 3 |
|  | | | | |
| **Figura 11** – Operadores aritméticos auxiliares em Java | | | | |

Para a utilização dos operadores Math.pow e Math.sqrt, necessitamos importar, no código fonte do Java, a seguinte biblioteca: **java.Math.\*;**.

### Prioridades

Para resolver expressões aritméticas em pseudocódigo, devemos obedecer a uma hierarquia de prioridades entre operadores aritméticos que é a seguinte:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| parênteses mais internos |
| pot rad |
| \* / div mod |
| + – |
| **Figura 12** – Prioridades entre operadores aritméticos | | | |

### Operadores unários

No Java, além dos operadores aritméticos básicos e auxiliares, podemos usar também os operadores unários (operadores que suportam apenas um operando), como se segue:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
|  | **Operador** | **Função** | **Exemplo** | **Equivalência** |  |
| + + | pré-incremento | ++ x | x = x + 1  (incrementa e logo depois usa o valor de x) |
| + + | pós-incremento | y ++ | x = x + 1  (usa o valor de x e logo depois incrementa) |
| – – | pré-decremento | – – z | y = y – 1  (decrementa e logo depois usa o valor de y) |
| – – | pós-decremento | k – – | y = y – 1  (usa o valor de y e logo depois decrementa) |
| + = | atribui a adição | a + = 2 | a = a + 2 |
| – = | atribui a subtração | b – = 5 | b = b – 5 |
| \* = | atribui a multiplicação | c \* = 9 | c = c \* 9 |
| / = | atribui a divisão | d / = 7 | d = d / 7 |
| % = | atribui o resto da div. | e % = 8 | e = e % 8 |
|  | | | | | |
| **Figura 13** – Operadores unários em Java | | | | | |

## 4.2 Operadores relacionais

### Pseudocódigo

Os operadores relacionais pertencem a um conjunto de símbolos que representam as comparações possíveis entre dois valores de mesmo tipo de dados. Esses valores podem ser constantes, variáveis ou expressões aritméticas. Os operadores relacionais podem ser:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
|  | **Operador** | **Função** | **Exemplo** | **Resultado** |  |
| > | maior que | 3 > 2 | Verdadeiro |
| < | menor que | 3 < 2 | Falso |
| > = | maior ou igual a | 5 – 2 > = 1 + 2 | Verdadeiro |
| < = | menor ou igual a | 5 – 2 < = 1 + 2 | Verdadeiro |
| = | igual a | 10 = 9 | Falso |
| < > | diferente de | 10 < > 10 | Falso |
|  | | | | | |
| **Figura 14** – Operadores relacionais | | | | | |

### Java

Os operadores relacionais em Java têm uma pequena diferença dos operadores relacionais em pseudocódigo e são representados como segue:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
|  | **Operador** | **Função** | **Exemplo** | **Resultado** |  |
| > | maior que | 1 > 8 | Falso |
| < | menor que | 1 < 8 | Verdadeiro |
| > = | maior ou igual a | – 2 > = 1 + 5 | Falso |
| < = | menor ou igual a | – 2 < = 7 – 2 | Verdadeiro |
| = = | igual a | 10 = = 9 | Falso |
| ! = | diferente de | 10 ! = 9 | Verdadeiro |
|  | | | | | |
| **Figura 15** – Operadores relacionais em Java | | | | | |

### Prioridades

Para resolver expressões lógicas em pseudocódigo, não há uma hierarquia de prioridades entre operadores relacionais que devemos obedecer.

## 4.3 Operadores lógicos

### Pseudocódigo

Os operadores lógicos pertencem a um conjunto de símbolos que representam os conectivos básicos. Esses conectivos formam novas proposições compostas a partir de outras proposições simples. Os operadores lógicos podem ser:

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | **Operador** | **Função** |  |
| não | negação |
| e | conjunção |
| ou | Disjunção |
|  | | | |
| **Figura 16** – Operadores lógicos | | | |

### Tabela Verdade

A tabela verdade determina todos os resultados possíveis de uma combinação entre valores de variáveis lógicas. Esses resultados possuem somente dois valores: verdadeiro ou falso. Para exemplificar o uso dos operadores lógicos, construiremos uma tabela verdade, que se segue:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | **A** | **B** | **não A** | **A e B** | **A ou B** |  |
| V | V | F | V | V |
| V | F | F | F | V |
| F | V | V | F | V |
| F | F | V | F | F |
|  | | | | | | |
| **Figura 17** – Tabela verdade | | | | | | |

### Java

Os operadores lógicos em Java são representados como segue:

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | **Operador** | **Função** |  |
| ! | Negação |
| && | Conjunção |
| || | Disjunção |
|  | | | |
| **Figura 18** – Operadores lógicos em Java | | | |

### Prioridades

Para resolver expressões lógicas em pseudocódigo, devemos obedecer a uma hierarquia de prioridades entre operadores lógicos que é a seguinte:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| não |
| e ou |
| **Figura 19** – Prioridades entre operadores lógicos | | | |

## 4.4 Prioridades entre Operadores

Para resolver expressões que contenham expressões lógicas e aritméticas, com operandos, operadores aritméticos, lógicos e relacionais, existe uma prioridade entre os operadores que é a seguinte:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| parênteses mais internos |
| operadores aritméticos |
| operadores relacionais |
| operadores lógicos |
| **Figura 20** – Prioridades entre operadores lógicos | | | |

## 4.5 Exemplo com operadores

1. Expressões aritméticas

a) 5 \* 2 – 3 + 14 / 2 + 9

10 – 3 + 7 + 9

9

b) 5 – pot(2 , 3) + 4 – 2 \* rad(4)

5 – 8 + 4 – 2 \* 2

5 – 8 + 4 – 4

– 3

c) pot(3 – 1 , 2) – 7 + rad(8 + 1) \* 2

pot(2 , 2) – 7 + rad(9) \* 2

4 – 7 + 3 \* 2

4 – 7 + 6

3

2. Expressões lógicas com operadores relacionais e aritméticos

a) 5 \* 2 = 4 + 10 / 2

10 = 4 + 5

10 = 9

F

b) 5 mod 2 + 3 < pot(3 , 2) \* 10

1 + 3 < 9 \* 10

4 < 90

V

c) 5 div 2 – 1 >= 4 / 2 + 7

2 – 1 >= 2 + 7

1 >= 9

F

3. Expressões lógicas com operadores lógicos, relacionais e aritméticos

a) 3 < 7 **e** 5 \* 2 = 2 + 1

3 < 7 **e** 10 = 3

V **e** F

F

b) pot(2 , 3) <= rad(9) **ou** 2 \* 3 – 6 / 3 = 4

8 <= 3 **ou** 6 – 2 = 4

F **ou** 4 = 4

F **ou** V

V

c) **não** V **ou** 2 \* 3 mod 4 / 2 – pot(2 , 1) + 7 = 3 **e** 4 \* 7 > 12

**não** V **ou** 6 mod 2 – 2 + 7 = 3 **e** 28 > 12

**não** V **ou** 0 – 2 + 7 = 3 **e** V

**não** V **ou** 5 = 3 **e** V

**não** V **ou** F **e** V

F **ou** F **e** V

F **ou** F

F

## 4.6 Exercícios com operadores

1. Calcule o resultado das seguintes expressões aritméticas:

a) 5 \* 9 mod 2 – 4 / 2

b) 4 – 7 div 2 + pot(2 , 4) / rad(4)

c) rad(pot(3 , 2)) + pot(2 , 3) mod rad(4) – 8 / 2

d) rad(16) div 9 / 3 – 4 \* 3 – pot(5 + 3 , rad(9))

e) rad(16 mod 4) + 5 \* 2 / 10 – pot(8 / 2 , 2 div 2)

f) rad(16) / 2 + 2 \* 5 mod 3 / 1 \* pot(3 , 2 + 4)

g) rad(rad(16)) + pot(pot(2 , 2) , pot(2 , 3)) + 5 \* 4 / 2

h) rad(rad(pot(2 , 4))) + pot(rad(4) , pot(rad(4) , 3)) + 9 – 5 \* 4

i) 5 \* 3 mod 5 + 4 div 8 / 2 + 5 mod (4 + 2)

j) rad(100) mod pot(4 , 3) – pot(2 , 4) div 8 / 2 + 5 – 3 \* 2

2. Calcule o resultado das seguintes expressões lógicas com operadores relacionais e aritméticos:

a) 3 \* 8 mod 2 = – 20 / 4

b) 5 – 10 div 3 > pot(3 , 3) / rad(25)

c) rad(pot(3 , 2)) < pot(5 , 2) mod rad(49) – 81 / 9

d) rad(81) div 8 / 3 – 4 >= 5 – pot(10 – 3 , rad(4))

e) rad(15 mod 3) + 15 \* 12 / 6 <= pot(10 / 5 , 12 div 10)

f) rad(100) / 10 + 12 = 10 mod 25 / 5 \* pot(5 , 2 + 1)

g) rad(rad(81)) + pot(pot(2 , 3) , pot(2 , 2)) <= 15 \* 10 / 5

h) rad(rad(pot(3 , 4))) + pot(rad(16) , pot(rad(4) , 2)) >= 19 – 15 \* 2

i) 27 \* 25 mod 13 + 51 div 3 > 12 + 35 mod (17 – 12)

j) rad(36) mod pot(2 , 3) – pot(3 , 4) div 18 / 6 < 25 – 13 \* 2

3. Calcule o resultado das seguintes expressões lógicas com operadores lógicos, relacionais e aritméticos:

a) 2 = 10 mod 2 **e** – 16 > 4

b) 5 – 10 div 3 > pot(3 , 3) **ou** rad(25) = 5

c) **não** F **e** rad(pot(3 , 2)) < pot(5 , 2) **ou** 5 mod rad(49) >= 81 / 9

d) **não** V **ou** rad(81) div 8 / 3 – 4 >= 5 – pot(10 – 3 , rad(4))

e) rad(15 mod 3) <= 15 **ou** 12 / 6 <= pot(10 / 5 , 12 div 10)

f) rad(100) / 10 + 12 = 10 mod 25 **ou** 5 \* pot(5 , 2 + 1) **e** **não** V

g) rad(rad(81)) > 5 **e** pot(pot(2 , 3) , pot(2 , 2)) <= 15 **e** 10 / 5 = 32

h) rad(rad(pot(3 , 4))) > 3 **ou** pot(rad(16), pot(rad(4) , 2)) >= 19 **ou** – 15 \* 2 = 7

i) **não** F **ou** 27 \* 25 mod 13 < 91 **e** 51 div 3 > 12 **ou** 35 mod (17 – 12) >= 13

j) rad(36) mod pot(2 , 3) >= 10 **ou** **não** V **e** – pot(3 , 4) div 18 / 6 < 25 – 13 \* 2

# Capítulo 5 – Estrutura Seqüencial

Uma estrutura seqüencial é aquela em que as ações de um algoritmo são executadas numa ordem seqüencial, ou seja, de cima para baixo e da esquerda para a direita. Para o pseudocódigo adotado nesta apostila, convencionaremos que o uso do ponto-e-vírgula (;) determina o final de uma ação.

Em qualquer estrutura de algoritmos, para expressar a entrada de dados, o processamento desses dados e a saída de resultados, usamos os comandos de atribuição, comandos de entrada e comandos de saída, os quais veremos a seguir:

## 5.2 Operador de atribuição (←) e comandos de atribuição

O operador de atribuição, no nosso pseudocódigo representado pelo operador de atribuição ← , serve para atribuir um determinado valor para uma variável, tendo o cuidado de verificar se o valor que está sendo atribuído à variável tem o tipo de dado compatível, ou seja, se uma variável **x** foi declarada como inteiro, só é permitido atribuir valores inteiros à variável **x**.

### Pseudocódigo

O comando de atribuição em pseudocódigo terá a seguinte regra sintática:

**<nome da variável>** ← **<expressão> ;**

onde **expressão** pode ser uma expressão lógica, aritmética ou lógica.

Por exemplo, vamos considerar que as variáveis **x**, **y** e **z** foram declaradas como do tipo **numérico\_inteiro**:

x ← 25;

y ← x + 15 – 3;

z ← y – x + rad(x) – pot(y , 2);

### Java

O comando de atribuição em Java será o igual (=) e sua regra sintática fica praticamente inalterada em relação ao pseudocódigo:

**<nome da variável>** = **<expressão> ;**

onde **expressão** pode ser uma expressão lógica, aritmética ou lógica.

Por exemplo, vamos considerar que as variáveis **x**, **y** e **z** foram declaradas como do tipo **int**:

x = 25;

y = x + 15 – 3;

z = y – x + Math.sqrt(x) – Math.pow(y , 2);

## 5.3 Comandos de entrada e saída

Os comandos de entrada e saída são utilizados para que haja uma interação entre o algoritmo e o usuário do algoritmo. Nesse caso, com o comando de entrada, é possível que o usuário forneça dados para serem processados pelo algoritmo e, com o comando de saída, é possível que o usuário veja o resultado do processamento dos dados.

### Pseudocódigo – comandos de entrada

Para que o usuário possa entrar com dados num algoritmo, utilizaremos a seguinte regra sintática para o comando de entrada:

**ler (<nome da variável>);**

Por exemplo, vamos considerar que uma variável **num** foi declarada como do tipo **numérico\_real**:

**ler (num)**;

Quando esse comando for executado, o algoritmo espera que o usuário digite algum valor real e o valor digitado será atribuído à variável **num**. Por exemplo, se o usuário digitar 5, então a variável **num** passará a ter o valor 5.0.

Podemos também receber valores para várias variáveis com um mesmo comando de entrada, seguindo a seguinte regra sintática:

**ler (<nome da variável>, <nome da variável> , ... , <nome da variável>);**

Por exemplo, vamos considerar que as variáveis **num1**, **num2** e **num3** foram declaradas como do tipo **numérico\_real**:

**ler (num1, num2, num3)**;

Quando esse comando for executado, o algoritmo espera que o usuário digite três valores reais separados pela tecla <enter> e os valores digitados serão atribuídos às variáveis **num1**, **num2** e **num3**. Por exemplo, se ele digitar: 5.2 <enter> -2.34 <enter> 0.0 <enter>, então as variáveis **num1**, **num2** e **num3** passarão a ter os valores 5.2, -2.34 e 0.0, respectivamente.

### Java – comandos de entrada

Para que o usuário possa entrar com dados num programa Java, utilizaremos o método **JOptionPane.showInputDialog** definido dentro do pacote **javax.swing**. Para isso, o programa deve fazer o **import** dessa biblioteca no início do programa da seguinte forma:

**import javax.swing.\*;**

Utilizaremos a seguinte regra sintática para o comando de entrada:

**<nome da variável> = <conversão tipos> (JOptionPane.showInputDialog("<msg>"));**

Por exemplo, vamos considerar que uma variável **num** foi declarada como do tipo **double**:

**num = Double.parseDouble(JOptionPane.showInputDialog("Digite um   
 número"))**;

Quando esse comando for executado, o programa espera que o usuário digite algum valor que é recebido pelo método como do tipo **String**. Esse valor é convertido para o tipo **double** e atribuído à variável **num**. Por exemplo, se o usuário digitar 5, então o método recebe a **String** "5", converte para o **double** 5.0 e a variável **num** passará a ter o valor 5.0.

Observação: note que a conversão de tipos só é necessária se a variável que receber o retorno do método **showInputDialog** não for do tipo **String**.

Os métodos acima explicados utilizam janelas para a comunicação com o usuário, ou seja, as mensagens de coleta de dados são feitas com o uso de janelas com uma interface gráfica mais amigável.

Outra forma utilizada para receber dados do usuário é usando o vetor[[2]](#footnote-2) de argumentos. Essa forma não possui uma interface gráfica e a coleta de dados é feita pela tela do MS-DOS. Quando executamos um determinado arquivo, por exemplo, **arquivo.class**, usamos o seguinte comando no *prompt* do DOS:

c:\ java arquivo

Podemos também passar argumentos como parâmetro para o programa que está sendo executado, por exemplo, da seguinte forma:

c:\ java arquivo 1 Logica

O dado **String** "1" e o dado **String** "Logica" estão sendo passados como parâmetros para o método principal. Mas onde?

O cabeçalho do método principal de qualquer programa Java é:

**public static void main(String arg [])**

**arg** é o nome dado para um vetor de **String** que recebe os argumentos passados como parâmetros quando o programa é executado. No nosso exemplo, dois valores ("1" e "Logica") foram passados como parâmetros, nesse caso, nosso vetor **arg** possui esses dois valores, um na sua posição 0 e um na sua posição 1.

Utilizaremos a seguinte regra sintática para pegar os dados desse vetor **arg**:

**<nome da variável> = <conversão de tipos>(arg[<posição>]);**

Por exemplo:

**int** num = **Integer.parseInt**(arg[0]);

**String** n = arg[1];

No primeiro exemplo, pega-se o conteúdo do vetor **arg** na posição 0, converte-se esse dado para o tipo **int** e armazena-se na variável **num**. No segundo exemplo, pega-se o conteúdo do vetor **arg** na posição 1 e armazena-se na variável **n**.

Note que podemos passar tantos argumentos como parâmetros quantos forem necessários e a ordem dos argumentos digitados é sempre a mesma que aparece no vetor **arg**.

Observação: note que a conversão de tipos só é necessária se a variável que receber o conteúdo do vetor **arg** não for do tipo **String**.

### Pseudocódigo – comandos de saída

Para que o usuário possa ver o resultado do processamento dos dados ou alguma mensagem enviada pelo algoritmo, utilizaremos a seguinte regra sintática para o comando de saída:

**escrever (<nome da variável>);**

Por exemplo, vamos considerar que uma variável **num** foi declarada como do tipo **numérico\_inteiro** e o valor 5 lhe foi atribuído:

**num ← 5;**

**escrever (num);**

Quando o comando **escrever** for executado, o algoritmo mostra na tela do usuário o conteúdo da variável **num**, nesse caso, o valor 5, já convertido para o tipo **alfanumérico**.

Ao invés de mostrar para o usuário o conteúdo de uma variável, podemos mostrar um dado **alfanumérico**, conforme a seguinte regra sintática abaixo:

**escrever ("<mensagem>");**

Por exemplo:

**escrever** **("Disciplina de Lógica de Programação");**

Quando o comando **escrever** for executado, o algoritmo mostra na tela do usuário a seguinte mensagem: *Disciplina de Lógica de Programação*.

Com o mesmo comando de saída, podemos enviar uma mensagem cujo conteúdo seja a concatenação de vários dados do tipo **alfanuméricos**, sejam eles do tipo **alfanuméricos** ou da conversão de outros tipos de dados para o tipo **alfanumérico**, seguindo a seguinte regra sintática:

**escrever ("<msg>", <nome var> , ... , "<msg>", <nome var>, ... , "<msg>");**

Por exemplo, vamos considerar que as variáveis **num1** e **num2** foram declaradas como do tipo **numérico\_inteiro** e os valores 10 e 5 lhes foram, respectivamente, atribuídos:

**num1 ← 10;**

**num2 ← 5;**

**escrever ("Maria tem " , num1 , " anos e João tem " , num2 , " anos");**

Quando o comando **escrever** for executado, o algoritmo mostra na tela do usuário a seguinte mensagem: *Maria tem 10 anos e João tem 5 anos*. Perceba que os conteúdos que estão entre aspas duplas ("Maria tem ", " anos e João tem " , " anos") são dados do tipo **alfanuméricos** e sua impressão na tela não se altera, enquanto que **num1** e **num2** são dados do tipo **numérico\_inteiro** e o que é mostrado na tela não são os nomes das variáveis mas sim os seus conteúdos (10 , 5) já convertidos para o tipo **alfanumérico**. Note ainda que a vírgula dentro do comando **escrever** tem a função de concatenar as informações, mostrando ao final uma única mensagem.

### Java – comandos de saída

Para que o usuário possa ver o resultado do processamento dos dados ou alguma mensagem enviada pelo programa Java, utilizaremos o método **JOptionPane.showMessageDialog** definido dentro do pacote **javax.swing**. Para isso, o programa deve fazer o **import** dessa biblioteca no início do programa da seguinte forma:

**import javax.swing.\*;**

Utilizaremos a seguinte regra sintática para o comando de saída:

**JOptionPane.showMessageDialog(null , "<mensagem>");**

Por exemplo, vamos considerar que uma variável **num** foi declarada como do tipo **numérico\_inteiro** e o valor 5 lhe foi atribuído:

**num = 5;**

**JOptionPane.showMessageDialog(null, num)**;

Quando o comando **JOptionPane.showMessageDialog** for executado, o programa mostra na tela do usuário o conteúdo da variável **num**, nesse caso, o valor 5, já convertido para o tipo **String**.

Ao invés de mostrar para o usuário o conteúdo de uma variável, podemos mostrar um dado **alfanumérico**, conforme a seguinte regra sintática abaixo:

**JOptionPane.showMessageDialog(null, "<mensagem>")**;

Por exemplo:

**JOptionPane.showMessageDialog(null, "Disciplina de Lógica de   
 Programação")**;

Quando o comando **JOptionPane.showMessageDialog** for executado, o programa mostra na tela do usuário a seguinte mensagem: *Disciplina de Lógica de Programação*.

Com o mesmo comando de saída, podemos enviar uma mensagem cujo conteúdo seja a concatenação de vários dados do tipo **String**, sejam eles do tipo **String** ou da conversão de outros tipos de dados para o tipo **String**, seguindo a seguinte regra sintática:

**JOptionPane.showMessageDialog(null, "<msg>" + <nome var> + ... + "<msg>" +   
 <nome var> + ... + "<msg>");**

Por exemplo, vamos considerar que as variáveis **num1** e **num2** foram declaradas como do tipo **int** e os valores 10 e 5 lhes foram, respectivamente, atribuídos:

**num1 = 10;**

**num2 = 5;**

**JOptionPane.showMessageDialog(null, "Maria tem " + num1 + " anos e   
 João tem " + num2 + " anos");**

Quando o comando **JOptionPane.showMessageDialog** for executado, o programa mostra na tela do usuário a seguinte mensagem: *Maria tem 10 anos e João tem 5 anos*. Perceba que os conteúdos que estão entre aspas duplas são dados do tipo **String** e sua impressão na tela não se altera, enquanto que **num1** e **num2** são dados do tipo **int** e o que é mostrado na tela não são os nomes das variáveis mas sim os seus conteúdos já convertidos para o tipo **String**. Note ainda que o sinal de mais (+) dentro do comando **JOptionPane.showMessageDialog** tem a função de concatenar as informações, mostrando ao final uma única mensagem.

Os métodos acima explicados utilizam janelas para a comunicação com o usuário, ou seja, as mensagens de aviso são feitas com o uso de janelas com uma interface gráfica mais amigável.

Outro método utilizado para mostrar o resultado de um processamento de dados ou alguma mensagem para o usuário é o método **System.out.print** ou **System.out.println**. A única diferença entre esses dois métodos é que o primeiro somente mostra a mensagem passada como parâmetro e o segundo, além de mostrar a mensagem, pula o cursor para a linha de baixo. Esses métodos não possuem uma interface gráfica e a mensagem é mostrada na tela do MS-DOS. Utilizaremos as seguintes regras sintáticas para este comando de saída:

**System.out.print ("<mensagem>");**

**System.out.println ("<mensagem>");**

Todas as propriedades de quantidade de parâmetros e concatenação de **Strings** vistos para o método **JOptionPane.showMessageDialog** também valem para os métodos **System.out.println** e **System.out.print**.

## 5.4 Blocos

Um bloco é representado por um conjunto de ações cujo início e fim são bem definidos. Um algoritmo pode ser considerado um bloco. As variáveis declaradas dentro de um bloco são reconhecidas somente dentro desse bloco.

### Pseudocódigo

Em pseudocódigo, um bloco tem seu início com a palavra **início** e seu término com a palavra **fim;**. Um algoritmo pode ser considerado um bloco, no entanto, seu início é determinado com a palavra **início\_algoritmo** e seu término com a palavra **fim\_algoritmo**.

Por exemplo:

**Algoritmo Exemplo**

**início\_algoritmo // início do bloco ou algoritmo**

**Declarar**

**// declaração das variáveis e/ou constantes**

**// seqüência dos comandos**

**fim\_algoritmo. // fim do bloco ou algoritmo**

Nesse exemplo, o bloco inicia-se com a palavra **início\_algoritmo** e termina com a palavra **fim\_algoritmo**.

Note que um algoritmo deve estar bem identado, isto é, deve estar numa formação que facilite o seu entendimento. No exemplo acima, veja que as palavras **início\_algoritmo** e **fim\_algoritmo** aparecem ligeiramente à direita do cabeçalho do algoritmo **Algoritmo Exemplo**; a palavra **Declarar** aparece ligeiramente à direita da palavra **início\_algoritmo** porque as declarações estão dentro do bloco; as declarações de variáveis e/ou constantes aparecem ligeiramente à direita da palavra **Declarar**, pois as declarações fazem parte do bloco de declaração; e, por fim, a seqüência de comandos aparece ligeiramente à direita da palavra **início\_algoritmo**, mas alinhado com a palavra **Declarar**, pois a seqüência de comandos também faz parte do bloco do algoritmo.

### Java

Em programas Java, um bloco tem seu início com a chave aberta, **{**, e seu término com a chave fechada, **}**. Um programa Java também pode ser considerado um bloco com início e término representados pela chave aberta e chave fechada, respectivamente.

Por exemplo:

**// nome da classe**

**class Exemplo**

**{ // início do bloco que limita a classe**

**// cabeçalho do método principal**

**public static void main(String argumentos[ ])**

**{ // início do bloco que limita o método principal**

**// declaração das variáveis e/ou constantes**

**// seqüência dos comandos**

**} // fim do bloco que limita o método principal**

**} // fim do bloco que limita a classe**

Nesse exemplo, o bloco inicia-se com a chave aberta e termina com a chave fechada. Nesse caso, temos dois blocos, o bloco da classe **exemplo** e o bloco do método principal **main()**.

Note que um programa Java também deve estar bem identado, isto é, deve estar numa formação que facilite o seu entendimento. No exemplo acima, veja que as chaves aberta e fechada aparecem ligeiramente à direita do nome da classe **class Exemplo** e ligeiramente à direita do método principal **main()**; as declarações de variáveis e/ou constantes aparecem ligeiramente à direita da chave aberta e da chave fechada do método principal **main()**, pois as declarações fazem parte do bloco do método principal; e, por fim, a seqüência de comandos também aparece ligeiramente à direita da chave aberta e da chave fechada do método principal **main()**, mas alinhada com as declarações, pois a seqüência de comandos também faz parte do bloco do método principal **main()**.

## 5.5 Estrutura seqüencial

Uma estrutura seqüencial é aquela em que as ações de um algoritmo são executadas numa ordem seqüencial, ou seja, de cima para baixo e da esquerda para a direita. Para o pseudocódigo adotado nesta apostila, convencionaremos que o uso do ponto-e-vírgula (;) determina o final de uma ação.

### Pseudocódigo

Em pseudocódigo, a estrutura seqüencial representa um conjunto de ações ou comandos que são executados num determinado fluxo. Numa estrutura sequëncial, este fluxo é executado de cima para baixo e da esquerda para a direita. A estrutura seqüencial para pseudocódigo segue a seguinte regra sintática:

**Algoritmo <nome do algoritmo>**

**início\_algoritmo**

**Declarar**

**// declaração das variáveis e/ou constantes**

**<comando\_1>;**

**<comando\_2>;**

**...**

**<comando\_n>;**

**fim\_algoritmo.**

Por exemplo:

**Algoritmo Exemplo\_Sequencial**

**início\_algoritmo**

**Declarar**

**x, y alfanumérico;**

**ler(x, y);**

**escrever("Você digitou primeiro o " , x);**

**escrever("Você digitou logo em seguida o " , y);**

**fim\_algoritmo.**

Esse exemplo caracteriza uma estrutura seqüencial, pois a execução desse algoritmo, desde seu início até o seu término, segue um fluxo: primeiro as variáveis do tipo **alfanumérico** **x** e **y** são declaradas; depois essas variáveis recebem valores digitados pelo usuário; depois uma mensagem com o valor de **x** é apresentada ao usuário; e, por último, uma mensagem com o valor de **y** é apresentada na tela.

Se o usuário digitou 5 e 10, o valor de **x** será 5 e o valor de **y** será 10. O resultado da execução deste exemplo será:

Você digitou primeiro o 5

Você digitou logo em seguida o 10

### Java

Em Java, a estrutura seqüencial não difere do pseudocódigo em relação ao comportamento, pois também representa um conjunto de ações ou comandos que são executados num determinado fluxo. Este fluxo, num programa seqüencial em Java, também é executado de cima para baixo e da esquerda para a direita. A estrutura seqüencial para Java segue a seguinte regra sintática:

**class <nome da classe>**

**{**

**public static void main(String argumentos[ ])**

**{**

**// declaração das variáveis e/ou constantes**

**<comando\_1>;**

**<comando\_2>;**

**...**

**<comando\_n>;**

**}**

**}**

Por exemplo:

**class Exemplo\_Seqüencial**

**{**

**public static void main(String arg[ ])**

**{**

**String x, y;**

**x = JOptionPane.showInputDialog("Digite um valor");**

**y = JOptionPane.showInputDialog("Digite um valor");**

**JOptionPane.showMessageDialog(null, "Você digitou   
 primeiro o " + x);**

**JOptionPane.showMessageDialog(null, "Você digitou logo   
 em seguida o " + y);**

**System.exit(0);**

**}**

**}**

Esse exemplo caracteriza uma estrutura seqüencial, pois a execução desse programa, desde seu início até o seu término, segue um fluxo: primeiro as variáveis do tipo **String** **x** e **y** são declaradas; depois a variável **x** recebe um valor digitado pelo usuário; depois a variável **y** recebe um valor digitado pelo usuário; depois uma mensagem com o valor de **x** é apresentada ao usuário; depois uma mensagem com o valor de **y** é apresentada na tela; e, por fim; o método **System.exit(0)**, que finaliza e sai do programa, é executado.

O resultado da execução desse exemplo é:

Digite um valor

// suponha que o usuário digitou 5 e clicou no botão ok

Digite um valor

// suponha que o usuário digitou 10 e clicou no botão ok

Você digitou primeiro o 5 // e o usuário clicou no botão ok

Você digitou logo em seguida o 10 // e o usuário clicou no botão ok

## 5.6 Funções de tratamento de caracteres em Java

Algumas vezes, precisamos trabalhar com caracteres em Java e para isso podemos utilizar alguns comandos existentes:

* **static boolean isDigit(char)**

Verifica se determinado caracter é digito.

* **static boolean isLowerCase(char)**

Verifica se determinado caracter é minúsculo.

* **static boolean isUpperCase(char)**

Verifica se determinado caracter é maiúsculo.

* **static boolean isSpace(char)**

Verifica se determinado caracter é um espaço.

* **static char toLowerCase(char)**

Retorna a conversão de determinado caracter para minúsculo.

* **static char toUpperCase(char)**

Retorna a conversão de determinado caracter para maiúsculo.

* **boolean equals(Object)**

Verifica se um objeto é igual a outro.

* **boolean equalsIgnoreCase(String)**

Verifica se um **String** é igual a outro sem diferenciar letras maiúsculas de minúsculas.

* **String concat(String)**

Retorna dois **Strings** concatenados em um único **String**.

* **String toString()**

Retorna a conversão de determinado valor num **String**.

* **int length()**

Retorna o tamanho de um **String**.

* **boolean compareTo(String)**

Verifica se dois **Strings** são iguais.

## 5.7 Exemplos de estrutura seqüencial em pseudocódigo

1. Desenvolva um algoritmo que receba três valores numéricos inteiros e mostre a soma desses três números.

**Algoritmo** Somar

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

num1, num2, num3, soma **numérico\_inteiro**;

// mensagem ao usuário

escrever ("Digite três valores inteiros");

// entrada de dados

ler (num1 , num2 , num3);

// processamento de dados

soma ← num1 + num2 + num3;

// saída de resultados

escrever ("A soma dos três valores digitados é: " , soma);

**fim\_algoritmo**.

2. Desenvolva um algoritmo que receba quatro notas bimestrais, calcule e mostre a média aritmética dessas quatro notas.

**Algoritmo** MediaAritmetica

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

n1, n2, n3, n4, media **numérico\_real**;

// mensagem ao usuário

escrever ("Digite as quatro notas bimestrais");

// entrada de dados

ler (n1 , n2 , n3 , n4);

// processamento de dados

media ← (n1 + n2 + n3 + n4) / 4;

// saída de resultados

escrever ("A média é: " , media);

**fim\_algoritmo**.

3. Desenvolva um algoritmo que receba a base e a altura de um triângulo, calcule e mostre a área deste triângulo.

**Algoritmo** AreaTriangulo

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

base, altura, area **numérico\_real**;

// mensagem ao usuário

escrever ("Digite a base e a altura do triângulo");

// entrada de dados

ler (base , altura);

// processamento de dados

area ← (base \* altura) / 2;

// saída de resultados

escrever ("A área do triângulo é: " , area);

**fim\_algoritmo**.

## 5.8 Exemplos de estrutura seqüencial em Java

1. Desenvolva um algoritmo que receba três valores numéricos inteiros e mostre a soma desses três números.

**class** Somar

**{**

**public static void main (String args [ ])**

**{**

// declaração de variáveis e/ou constantes

**int** num1, num2, num3, soma;

// mensagem ao usuário e entrada de dados

num1 = Integer.parseInt(JOptionPane.showInputDialog(   
 "Digite um valor inteiro"));

num2 = Integer.parseInt(JOptionPane.showInputDialog(   
 "Digite um valor inteiro"));

num3 = Integer.parseInt(JOptionPane.showInputDialog(   
 "Digite um valor inteiro"));

// processamento de dados

soma = num1 + num2 + num3;

// saída de resultados

System.out.println ("A soma dos três valores digitados   
 é: " + soma);

System.exit(0);

**}** // fim do void main

**}** // fim da classe Somar

2. Desenvolva um algoritmo que receba quatro notas bimestrais, calcule e mostre a média aritmética dessas quatro notas.

**class** MediaAritmetica

**{**

**public static void main (String args [ ])**

**{**

// declaração de variáveis e/ou constantes

**double** n1, n2, n3, n4, media;

// mensagem ao usuário e entrada de dados

n1 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite uma nota bimestral"));

n2 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite uma nota bimestral"));

n3 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite uma nota bimestral"));

n4 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite uma nota bimestral"));

// processamento de dados

media = (n1 + n2 + n3 + n4) / 4;

// saída de resultados

System.out.println ("A média é: " + media);

System.exit(0);

**}** // fim do void main

**}** // fim da classe

3. Desenvolva um algoritmo que receba a base e a altura de um triângulo, calcule e mostre a área desse triângulo.

**class** AreaTriangulo

**{**

**public static void main (String args [ ])**

**{**

// declaração de variáveis e/ou constantes

**double** base, altura, area;

// mensagem ao usuário e entrada de dados

base = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite a base do triângulo"));

altura = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite altura do triângulo"));

// processamento de dados

area = (base \* altura) / 2;

// saída de resultados

System.out.println ("A área do triângulo é: " + area);

System.exit(0);

**}** // fim do void main

**}** // fim da classe main

## 5.9 Exercícios de algoritmos seqüenciais

1. Desenvolva um algoritmo que receba o salário de um funcionário, calcule e mostre seu novo salário com reajuste de 15%.

2. Desenvolva um algoritmo que receba a base e a altura de um retângulo, calcule e mostre a área desse retângulo. Fórmula da área de um retângulo: A = base . altura.

3. Desenvolva um algoritmo que receba uma temperatura em graus Celsius (C), calcule e mostre a temperatura convertida para graus Fahrenheit (F).

Fórmula de conversão: F = (9.C + 160) / 5.

4. Desenvolva um algoritmo que receba uma temperatura em graus Kelvin (K), calcule e mostre a temperatura convertida para graus Celsius (C).

Fórmula de conversão: C = K - 273.

5. Desenvolva um algoritmo que receba dois valores inteiros para as variáveis **x** e **y**, efetue a troca dos valores, ou seja, **x** passa a ter o valor de **y** e **y** passa a ter o valor de **x** e mostre os valores trocados.

6. Desenvolva um algoritmo que receba os coeficientes a, b e c de uma equação de segundo grau (ax2 + bx + c), calcule e mostre as raízes reais dessa equação, considerando uma equação que possui duas raízes reais.

7. Desenvolva um algoritmo que receba os valores do comprimento (C), da largura (L) e da altura (H) de um paralelepípedo, calcule e mostre o volume desse paralelepípedo. Fórmula do volume de um paralelepípedo: V = C . L . H.

8. Desenvolva um algoritmo que receba o salário bruto de um funcionário, calcule e mostre o salário líquido desse funcionário, sabendo que ele recebe 10% de gratificação calculados sobre o salário bruto, mas paga 20% de imposto de renda também calculados sobre o salário bruto.

9. Desenvolva um algoritmo que receba o valor de um depósito em poupança, calcule e mostre o valor após um mês de aplicação na poupança, sabendo que a poupança rende 5% ao mês.

10. Desenvolva um algoritmo que receba um valor em Real, calcule e mostre o valor convertido para Dólar.

11. Desenvolva um algoritmo que receba um valor em Euro, calcule e mostre o valor convertido para Real.

12. Desenvolva um algoritmo que receba um valor numérico real, calcule e mostre o quadrado desse número.

13. Desenvolva um algoritmo que receba dois valores numérico inteiro, calcule e mostre a soma do quadrado desses dois números.

14. Desenvolva um algoritmo que receba dois valores numérico real, calcule e mostre o quadrado da diferença desses dois números.

15. Desenvolva um algoritmo que receba o nome e o sobrenome de uma pessoa e mostre o nome inteiro dessa pessoa.

16. Desenvolva um algoritmo que receba três valores numérico inteiro, calcule e mostre o cubo da soma desses três números.

17. Desenvolva um algoritmo que receba dois valores numérico real, calcule e mostre a diferença do cubo desses dois números.

18. Desenvolva um algoritmo que receba o nome, a idade e o sexo de uma pessoa e mostre essas informações na tela.

19. Desenvolva um algoritmo que receba o raio (R) de uma circunferência, calcule e mostre o comprimento dessa circunferência.

Fórmula do comprimento da circunferência: C = 2 . π . R.

20. Desenvolva um algoritmo que receba o raio (R) de uma circunferência, calcule e mostre a área dessa circunferência. Fórmula da área: A = π . R2.

21. Desenvolva um algoritmo que receba o ano de nascimento de uma pessoa e o ano atual, calcule e mostre a idade dessa pessoa e quantos anos essa pessoa terá daqui a 10 anos.

22. Desenvolva um algoritmo que receba o número de horas trabalhadas por um funcionário e quanto esse funcionário recebe por hora trabalhada, calcule e mostre o valor que deve ser recebido por esse funcionário.

23. Desenvolva um algoritmo que receba uma quantidade de um alimento em quilos, calcule e mostre quantos dias durará esse alimento para uma pessoa que consome 50 gramas desse alimento por dia.

24. Desenvolva um algoritmo que receba a altura de um degrau, calcule e mostre quantos degraus uma pessoa precisa subir se deseja estar a uma altura de 5 quilômetros.

25. Desenvolva um algoritmo que receba um valor numérico inteiro, calcule e mostre qual o quociente e o resto da divisão desse número por 3.

26. Desenvolva um algoritmo que receba os valores dos catetos (C1, C2) de um triângulo retângulo, calcule e mostre o valor da hipotenusa (H) desse triângulo. Fórmula do cálculo da hipotenusa de um triângulo retângulo. H2 = C12 + C22.

27. Desenvolva um algoritmo que receba os valores de dois ângulos de um triângulo qualquer, calcule e mostre o valor do terceiro ângulo desse triângulo, sabendo que a soma dos três ângulos de um triângulo é igual a 180.

28. Desenvolva um algoritmo que receba um horário em horas, minutos e segundos, calcule e mostre este mesmo horário convertido em minutos.

29. Desenvolva um algoritmo que receba a data de nascimento de uma pessoa e a data atual, calcule e mostre a idade desta pessoa em anos, meses, dias e semanas.

30. Desenvolva um algoritmo que receba o valor de um produto, calcule e mostre o valor desse produto após um desconto de 15%.

# Capítulo 6 – Estrutura de Decisão

Quando desenvolvemos um algoritmo ou programa estruturado, muitas vezes, precisamos interferir no fluxo natural ou seqüencial que o algoritmo deve seguir. No entanto, precisamos saber controlar qual o fluxo que o algoritmo percorre do início até o fim de sua execução. Para isso, utilizaremos os fluxos de controle, por exemplo, estrutura de decisão e estrutura de repetição.

Uma estrutura de decisão é um fluxo de controle utilizado para decidir qual o fluxo que o algoritmo seguirá. Uma estrutura de decisão determina qual conjunto de comandos ou bloco será executado após uma condição ser avaliada. Essa condição é representada por expressões lógicas e relacionais que podem ou não serem satisfeitas, isto é, podem retornar ao valor verdadeiro ou falso.

Uma estrutura de decisão pode ser simples, composta ou encadeada. Veremos a seguir como e quando trabalhar com cada uma delas.

## 6.1 Estrutura de Decisão Simples (se/então)

Uma estrutura de decisão simples pode ser utilizada quando o algoritmo precisa testar determinada condição antes de executar um conjunto de comandos. Nesse caso, uma condição é avaliada e, se seu resultado for verdadeiro, o conjunto de comandos dentro da estrutura de decisão simples **se/então** é executado. Por outro lado, se o resultado da avaliação for falso, esse conjunto de comandos não fará parte do fluxo de execução do algoritmo.

### Pseudocódigo

Na estrutura de decisão simples, utilizaremos as palavras **se** e **então** que representam as palavras principais dessa estrutura e a palavra **fimse;** para determinar o fim do bloco de execução dessa estrutura. A estrutura de decisão simples para pseudocódigo segue a seguinte regra sintática:

**se (<condição>)**

**então**

**<comandos>;**

**fimse;**

Por exemplo:

**se** (var < 0)

**então**

escrever("O número " , var , " é negativo");

**fimse**;

Nesse exemplo, a variável **var** possui um valor numérico e vai passar por uma avaliação, ou seja, a expressão **var<0** será avaliada. Se o resultado de retorno dessa avaliação for verdadeira, então o comando **escrever(...);** será executado. Por outro lado, se o resultado da avaliação for falso, então o mesmo comando não será executado.

O resultado da execução desse exemplo é:

// se o valor de **var** for -5

O número –5 é negativo

### Java

Na estrutura de decisão simples, utilizaremos a palavra **if** para representar a palavra principal dessa estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte da estrutura. A estrutura de decisão simples para Java segue a seguinte regra sintática:

**if (<condição>)**

**{**

**<comandos>;**

**}**

Por exemplo:

**if** (var < 0)

**{**

System.out.print("O número " + var + " é negativo");

**}**

Nesse exemplo, assim como no pseudocódigo, a variável **var** possui um valor numérico e vai passar por uma avaliação. Se o resultado de retorno dessa avaliação for verdadeiro, então o comando **System.out.print(...);** será executado. Por outro lado, se o resultado da avaliação for falso, então o comando não será executado.

O resultado da execução desse exemplo é:

// se o valor de **var** for -5

O número –5 é negativo

## 6.2 Estrutura de Decisão Composta (se/então/senão)

Uma estrutura de decisão composta é parecida com a estrutura de decisão simples. A diferença entre as duas estruturas é que a primeira pode ser utilizada quando o algoritmo precisa testar determinada condição, cujos resultados podem ser verdadeiro ou falso, antes de executar um conjunto de comandos. Neste caso, uma condição é avaliada e, se seu resultado for verdadeiro, o conjunto de comandos dentro da estrutura **se/então** é executado. Por outro lado, se o resultado da avaliação for falso, o conjunto de comandos dentro da estrutura **senão** é executado.

### Pseudocódigo

Na estrutura de decisão composta, utilizaremos as palavras **se**, **então** e **senão** que representam as palavras principais dessa estrutura e a palavra **fimse;** para determinar o fim do bloco de execução dessa estrutura. A estrutura de decisão composta para pseudocódigo segue a seguinte regra sintática:

**se (<condição>)**

**então**

**<comandos1>;**

**senão**

**<comandos2>;**

**fimse;**

Por exemplo:

**se** (var < 0)

**então**

escrever("O número " , var , " é negativo");

**senão**

escrever("O número " , var , " é não negativo");

**fimse**;

Nesse exemplo, a variável **var** possui um valor numérico e vai passar por uma avaliação, ou seja, a expressão **var<0** será avaliada. Se o resultado de retorno dessa avaliação for verdadeiro, então o comando **escrever(...);** dentro da estrutura do **então** será executado. Por outro lado, se o resultado da avaliação for falso, então o comando **escrever(...);** dentro da estrutura do **senão** é executado. Note que somente os comandos de uma das duas estruturas **então** e **senão** são executados, ou seja, nunca as duas estruturas podem ser executadas num mesmo momento.

O resultado da execução desse exemplo é:

// se o valor de **var** for 5

O número 5 é não negativo

// se o valor de **var** for -5

O número –5 é negativo

### Java

Na estrutura de decisão composta, utilizaremos as palavras **if** e **else** para representar as palavras principais dessa estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte da estrutura. A estrutura de decisão composta para Java segue a seguinte regra sintática:

**if (<condição>)**

**{**

**<comandos1>;**

**}**

**else**

**{**

**<comandos2>;**

**}**

Por exemplo:

**if** (var < 0)

**{**

System.out.print("O número " + var + " é negativo");

**}**

**else**

**{**

System.out.print("O número " + var + " é não negativo");

**}**

Nesse exemplo, assim como no pseudocódigo, a variável **var** possui um valor numérico e vai passar por uma avaliação. Se o resultado de retorno dessa avaliação for verdadeiro, então o comando **System.out.print(...);** dentro da estrutura do **if** será executado. Por outro lado, se o resultado da avaliação for falso, então o comando **System.out.print(...);** dentro da estrutura do **else** é executado. Assim, como no pseudocódigo, somente os comandos de uma das duas estruturas **if** e **else** são executados, ou seja, nunca as duas estruturas podem ser executadas num mesmo momento.

O resultado da execução desse exemplo é:

// se o valor de **var** for 5

O número 5 é não negativo

// se o valor de **var** for -5

O número –5 é negativo

## 6.3 Estrutura de Decisão Encadeada (se/ então ... / senão ...)

Uma estrutura de decisão encadeada pode ser utilizada quando o algoritmo precisa testar um conjunto de condições, cujos resultados podem ser verdadeiro ou falso. Dizemos que essa estrutura é encadeada, pois há estruturas de decisões dentro de outras estruturas de decisões. Para qualquer caso, cada condição é avaliada e, se seu resultado for verdadeiro, o conjunto de comandos dentro da estrutura **se/então** é executado. Por outro lado, se o resultado da avaliação for falso, o conjunto de comandos dentro da estrutura **senão** é executado. Não existe limite para a estrutura de decisão encadeada, podemos ter quantas estruturas de decisão encadeadas forem necessárias. Note que, para qualquer estrutura de decisão, no máximo, um único conjunto de comando será executado, aquele cuja avaliação de condição for sempre verdadeira.

### Pseudocódigo

Na estrutura de decisão encadeada, utilizaremos as palavras **se**, **então** e **senão** que representam as palavras principais dessa estrutura e a palavra **fimse;** para determinar o fim do bloco de execução dessa estrutura. Note que, para cada estrutura **se** que aparece, existe um **fimse** adequadamente localizado dentro da estrutura. A estrutura de decisão encadeada para pseudocódigo segue a seguinte regra sintática:

**se (<condição1>)**

**então**

**se (<condição2>)**

**então**

**se (<condição3>)**

**então**

**... // continua com outra estrutura de seleção ou coloca os comandos**

**senão**

**... // continua com outra estrutura de seleção ou coloca os comandos**

**fimse; // da condição 3**

**fimse; // da condição 2**

**senão**

**se (<condição4>)**

**então**

**se (<condição5>)**

**então**

**... // continua com outra estrutura de seleção ou coloca os comandos**

**senão**

**... // continua com outra estrutura de seleção ou coloca os comandos**

**fimse; // da condição 5**

**fimse; // da condição 4**

**fimse; // da condição 1**

Por exemplo:

**se** (var < 0)

**então**

escrever("O número " , var , " é negativo");

**senão**

**se** (var = 0)

**então**

escrever("O número " , var , " é nulo");

**senão**

escrever("O número " , var , " é positivo");

**fimse**; // do var = 0

**fimse**; // do var < 0

Nesse exemplo, a variável **var** possui um valor numérico e vai passar por uma avaliação, ou seja, a expressão **var<0** será avaliada. Se o resultado de retorno dessa avaliação for verdadeiro, então o comando **escrever(...);** dentro da estrutura do primeiro **então** será executado. Por outro lado, se o resultado da avaliação for falso, então uma nova estrutura de decisão será avaliada, ou seja, a expressão **var=0** será avaliada. Se o resultado de retorno dessa avaliação for verdadeiro, então o comando **escrever(...);** dentro da estrutura do segundo **então** será executado. Por outro lado, se o resultado da segunda avaliação for falso, então o comando **escrever(...);** dentro da estrutura do segundo **senão** é executado. Note que somente os comandos de uma única das estruturas de toda a estrutura de decisão encadeada são executados.

O resultado da execução desse exemplo é:

// se o valor de **var** for 5

O número 5 é positivo

// se o valor de **var** for -5

O número –5 é negativo

// se o valor de **var** for 0

O número 0 é nulo

### Java

Na estrutura de decisão encadeada, utilizaremos as palavras **if** e **else** para representar as palavras principais desta estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte desta estrutura. A estrutura de decisão encadeada para Java segue a seguinte regra sintática:

**if (<condição1>)**

**{**

**if (<condição2>)**

**{**

**if (<condição3>)**

**{**

**... // continua com outra estrutura de seleção ou coloca os comandos**

**} // fim da condição 3 quando verdadeira**

**else**

**{**

**... // continua com outra estrutura de seleção ou coloca os comandos**

**} // fim do else da condição 3**

**} // fim da condição 2 quando verdadeira**

**} // fim do else da condição 1**

**else**

**{**

**if (<condição4>)**

**{**

**if (<condição5>)**

**{**

**... // continua com outra estrutura de seleção ou coloca os comandos**

**} // fim da condição 5 quando verdadeira**

**else**

**{**

**... // continua com outra estrutura de seleção ou coloca os comandos**

**} // fim do else da condição 5**

**} // fim da condição 4 quando veradeira**

**} // fim do else da condição 1**

Por exemplo:

**if** (var < 0)

**{**

System.out.print("O número " + var + " é negativo");

**}**

**else**

**{**

**if** (var = 0)

**{**

System.out.print("O número " + var + " é nulo");

**}**

**else**

**{**

System.out.print("O número " + var + " é positivo");

**}** // do var = 0

**}** // do var < 0

Nesse exemplo, assim como no pseudocódigo, a variável **var** possui um valor numérico e vai passar por uma avaliação, ou seja, a expressão **var<0** será avaliada. Se o resultado de retorno dessa avaliação for verdadeiro, então o comando **System.out.print(...);** dentro da estrutura do primeiro **if** será executado. Por outro lado, se o resultado da avaliação for falso, então uma nova estrutura de decisão será avaliada, ou seja, a expressão **var=0** será avaliada. Se o resultado de retorno dessa avaliação for verdadeiro, então o comando **System.out.print(...);** dentro da estrutura do segundo **if** é executado. Por outro lado, se o resultado da segunda avaliação for falso, então o comando **System.out.print(...);** dentro da estrutura do segundo **else** é executado. Assim, como no pseudocódigo, somente os comandos de uma única das estruturas de toda a estrutura de decisão encadeada são executados. Nesse caso, somente uma das impressões é efetuada.

O resultado da execução desse exemplo é:

// se o valor de **var** for 5

O número 5 é positivo

// se o valor de **var** for -5

O número –5 é negativo

// se o valor de **var** for 0

O número 0 é nulo

## 6.4 Exemplos de estrutura de decisão simples, composta e encadeada em pseudocódigo

1. Desenvolva um algoritmo que receba um valor numérico real, verifique e mostre se esse número é positivo.

**Algoritmo** Positivo

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

x **numérico\_real**;

// mensagem ao usuário

escrever ("Digite um número real");

// entrada de dados

ler (x);

// processamento de dados

**se** (x > 0)

**então**

// saída de resultados

escrever (x , " é positivo");

**fimse**;

**fim\_algoritmo**.

2. Desenvolva um algoritmo que receba três valores numéricos inteiros, mostra a soma desses três números, verifique e mostre se a soma é maior que 100.

**Algoritmo** Somar

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

num1, num2, num3, soma **numérico\_inteiro**;

// mensagem ao usuário

escrever ("Digite três valores inteiros");

// entrada de dados

ler (num1 , num2 , num3);

// processamento de dados

soma ← num1 + num2 + num3;

// saída de resultados

escrever ("A soma dos três valores digitados é: " , soma);

**se** (soma > 100)

**então**

// saída de resultados

escrever("A soma é maior que 100");

**senão**

// saída de resultados

escrever("A soma é menor ou igual a 100");

**fimse**;

**fim\_algoritmo**.

3. Desenvolva um algoritmo que receba quatro notas bimestrais, calcule e mostre a média aritmética dessas quatro notas, bem como, se o aluno foi aprovado (média >= 7), reprovado (média < 3) ou em exame (média >= 3 ou média <7).

**Algoritmo** MediaAritmetica

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

n1, n2, n3, n4, media **numérico\_real**;

// mensagem ao usuário

escrever ("Digite as quatro notas bimestrais");

// entrada de dados

ler (n1 , n2 , n3 , n4);

// processamento de dados

media ← (n1 + n2 + n3 + n4) / 4;

// saída de resultados

escrever ("A média é: " , media);

// processamento de dados

**se** (media >= 7)

**então**

// saída de resultados

escrever ("Aluno aprovado");

**senão**

**se** (media < 3)

**então**

// saída de resultados

escrever ("Aluno reprovado");

**senão**

// saída de resultados

escrever ("Aluno em exame");

**fimse**; // do media < 3

**fimse**; // do media >= 7

**fim\_algoritmo**.

4. Desenvolva um algoritmo que receba a base e a altura de um triângulo e verifique se os dados recebidos são válidos, ou seja, todos maiores que zero. Se os dados forem válidos, calcule e mostre a área desse triângulo, caso contrário, mostre uma mensagem de dados inválidos ao usuário.

**Algoritmo** AreaTriangulo

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

base, altura, area **numérico\_real**;

// mensagem ao usuário

escrever ("Digite a base e a altura do triângulo");

// entrada de dados

ler (base , altura);

// processamento de dados

**se** (base <= 0 **ou** altura <= 0)

**então**

// mensagem ao usuário

escrever ("Dados digitados inválidos. As medidas de base e altura de um triângulo   
 devem ser maiores que zero");

**senão**

// processamento de dados

area ← (base \* altura) / 2;

// saída de resultados

escrever ("A área do triângulo é: " , area);

**fimse**;

**fim\_algoritmo**.

5. Desenvolva um algoritmo que receba três valores numéricos reais, verifique e mostre se esses valores podem ser o comprimento dos lados de um triângulo. Se falso, mostrar essa informação ao usuário, caso contrário, verificar se formam um triângulo equilátero, isósceles ou escaleno.

**Algoritmo** Triangulo

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

lado1, lado2, lado3 **numérico\_real**;

// mensagem ao usuário

escrever ("Digite três valores reais");

// entrada de dados

ler (lado1, lado2, lado3);

// processamento de dados

**se** ((lado1 < lado2 + lado3) **e** (lado2 < lado1 + lado3 ) **e** (lado3 < lado1 + lado2))

**então**

// saída de resultados

escrever ("Com os valores digitados, podemos construir um triângulo ");

**se** ((lado1 = lado2) **e** (lado2 = lado3))

**então**

// saída de resultados

escrever ("equilátero, com três lados iguais.");

**senão**

**se** ((lado1 = lado2) **ou** (lado1 = lado3) **ou** (lado2 = lado3))

**então**

// saída de resultados

escrever ("isósceles, com dois lados iguais.");

**senão**

// saída de resultados

escrever ("escaleno, com três lados diferentes");

**fimse**;

**fimse**;

**senão**

// saída de resultados

escrever ("Com os valores digitados, não podemos construir um triângulo ");

**fimse**;

**fim\_algoritmo**.

## 6.5 Exemplos de estrutura de decisão simples, composta e encadeada em Java

1. Desenvolva um algoritmo que receba um valor numérico real, verifique e mostre se esse número é positivo.

**class** Positivo

**{**

**public static void main (String args [ ])**

**{**

// declaração de variáveis e/ou constantes

**double** x;

// mensagem ao usuário e entrada de dados

x = Double.parseDouble(JOptionPane.showInputDialog("Digite  
 um número real"));

// processamento de dados

**if** (x > 0)

**{**

// saída de resultados

System.out.println (x + " é positivo");

**}** // fim do if

System.exit(0);

**}** // fim do void main

**}** // fim da classe Positivo

2. Desenvolva um algoritmo que receba três valores numéricos inteiros, mostre a soma desses três números, verifique e mostre se a soma é maior que 100.

**class** Somar

**{**

**public static void main (String args [ ])**

**{**

// declaração de variáveis e/ou constantes

**int** num1, num2, num3, soma;

// mensagem ao usuário e entrada de dados

num1 = Integer.parseInt(JOptionPane.showInputDialog(   
 "Digite um valor inteiro"));

num2 = Integer.parseInt(JOptionPane.showInputDialog(   
 "Digite um valor inteiro"));

num3 = Integer.parseInt(JOptionPane.showInputDialog(   
 "Digite um valor inteiro"));

// processamento de dados

soma = num1 + num2 + num3;

// saída de resultados

System.out.println ("A soma dos três valores digitados   
 é: " + soma);

**if** (soma > 100)

**{**

// saída de resultados

System.out.println("A soma é maior que 100");

**}** // fim do if

**else**

**{**

// saída de resultados

System.out.println("A soma é menor ou igual a 100");

**}**// fim do else

System.exit(0);

**}** // fim do void main

**}** //fim da classe

3. Desenvolva um algoritmo que receba quatro notas bimestrais, calcula e mostra a média aritmética dessas quatro notas, bem como, se o aluno foi aprovado (média >= 7), reprovado (média < 3) ou em exame (média >= 3 ou média <7).

**class** MediaAritmetica

**{**

**public static void main (String args [ ])**

**{**

// declaração de variáveis e/ou constantes

**double** n1, n2, n3, n4, media;

// mensagem ao usuário e entrada de dados

n1 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite uma nota bimestral"));

n2 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite uma nota bimestral"));

n3 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite uma nota bimestral"));

n4 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite uma nota bimestral"));

// processamento de dados

media = (n1 + n2 + n3 + n4) / 4;

// saída de resultados

System.out.println ("A média é: " + media);

// processamento de dados

**if** (media >= 7)

**{**

// saída de resultados

System.out.println ("Aluno aprovado");

**}**

**else**

**{**

**if** (media < 3)

**{**

// saída de resultados

System.out.println ("Aluno reprovado");

**}**

**else**

**{**

// saída de resultados

System.out.println ("Aluno em exame");

**}** // do media < 3

**}** // do media >= 7

System.exit(0);

**}** // fim do void main

**}** //fim da classe

4. Desenvolva um algoritmo que receba a base e a altura de um triângulo, verifique se os dados recebidos são válidos, ou seja, todos maiores que zero. Se os dados forem válidos, calcuque e mostre a área desse triângulo, caso contrário, mostre mensagem de dados inválidos ao usuário.

**class** AreaTriangulo

**{**

**public static void main (String args [ ])**

**{**

// declaração de variáveis e/ou constantes

**double** base, altura, area;

// mensagem ao usuário e entrada de dados

base = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite a base do triângulo"));

altura = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite altura do triângulo"));

// processamento de dados

**if** (base <= 0 **||** altura <= 0)

**{**

// mensagem ao usuário

System.out.println ("As medidas de um triângulo devem   
 ser maiores que zero");

**}**

**else**

**{**

// processamento de dados

area = (base \* altura) / 2;

// saída de resultados

System.out.print ("A área do triângulo é: " + area);

**}**

System.exit(0);

**}** // fim do void main

**}** //fim da classe

5. Desenvolva um algoritmo que receba três valores numéricos reais, verifique e mostre se esses valores podem ser o comprimento dos lados de um triângulo. Se falso, mostrar essa informação ao usuário, caso contrário, verificar se formam um triângulo equilátero, isósceles ou escaleno.

**class** Triangulo

**{**

**public static void main (String args [ ])**

**{**

// declaração de variáveis e/ou constantes

**double** lado1, lado2, lado3;

// mensagem ao usuário e entrada de dados

lado1 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite o lado do triângulo"));

lado2 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite o lado do triângulo"));

lado3 = Double.parseDouble(JOptionPane.showInputDialog(   
 "Digite o lado do triângulo"));

// processamento de dados

**if** ((lado1 < lado2 + lado3) **&&** (lado2 < lado1 + lado3 )   
 **&&** (lado3 < lado1 + lado2))

**{**

// saída de resultados

System.out.print ("Com os valores digitados,   
 podemos construir um triângulo");

**if** ((lado1 == lado2) **&&** (lado2 == lado3))

**{**

// saída de resultados

System.out.print ("equilátero, com três lados  
 iguais.");

**}**

**else**

**{**

**if** ((lado1 == lado2) **||** (lado1 == lado3)  
 **||** (lado2 == lado3))

**{**

// saída de resultados

System.out.print ("isósceles, com dois   
 lados iguais.");

**}**

**else**

**{**

// saída de resultados

System.out.print ("escaleno, com três lados   
 diferentes");

**}**

**}**

**}**

**else**

**{**

// saída de resultados

System.out.print ("Com os valores digitados, não   
 podemos construir um triângulo ");

**}**

System.exit(0);

**}** // fim do void main

**}** //fim da classe

## 6.6 Exercícios de estrutura de decisão simples, composta e encadeada

1. Desenvolva um algoritmo que receba dois valores numérico inteiro, calcule e mostre o resultado da diferença do maior pelo menor valor.

2. Desenvolva um algoritmo que receba três valores numérico real e mostre-os em ordem crescente. Utilizar a estrutura de decisão encadeada.

3. Desenvolva um algoritmo que receba um valor inteiro, calcule e mostre o módulo desse número. O módulo de um número qualquer é esse número positivo, ou seja, o módulo de –5 é 5 e o módulo de 5 é 5.

4. Desenvolva um algoritmo que receba dois valores numérico real, calcule e mostre o maior entre esses dois números.

5. Desenvolva um algoritmo que receba os três coeficientes a, b e c de uma equação de segundo grau da forma ax2 + bx + c = 0, verifique e mostre a existência de raízes reais e caso exista raíz(es) real(is), calcule e mostre essa(s) raíz(es).

6. Desenvolva um algoritmo que receba os valores referentes a quatro notas bimestrais de um aluno, calcule a média desse aluno e mostre a mensagem de aprovação (média >= 5) ou reprovação (média < 5) do aluno, juntamente com a média calculada.

7. Desenvolva um algoritmo que receba três valores obrigatoriamente em ordem crescente e um quarto valor não necessariamente nessa ordem. Esse algoritmo deve mostrar os quatro números em ordem decrescente.

8. Desenvolva um algoritmo que receba um valor numérico inteiro, verifique e mostre se esse valor é divisível por 2 e por 3.

9. Desenvolva um algoritmo que receba um valor numérico inteiro, verifique e mostre se esse valor é divisível por 2 ou por 3.

10. Desenvolva um algoritmo que receba um valor numérico real, verifique e mostre se esse valor é par ou ímpar.

11. Desenvolva um algoritmo que receba os valores da altura e do sexo de uma pessoa, calcule e mostre o peso ideal dessa pessoa, utilizando as seguintes regras:

- para homens: (72.7 \* altura) – 58;

- para mulheres: (62.1 \* altura) – 44.7.

12. Desenvolva um algoritmo que receba a data de nascimento de uma pessoa e a data atual, calcule e mostre se essa pessoa é maior de idade.

13. Desenvolva um algoritmo que receba o ano de nascimento de uma pessoa, calcule e mostre a sua idade, e também verifique e mostre se essa pessoa já tem idade para votar (16 anos ou mais) e se tem idade para obter a Carteira Nacional de Habilitação (18 anos ou mais).

14. Desenvolva um algoritmo que receba a hora de início de um jogo e a hora final desse jogo (cada hora é composta por duas variáveis inteiras: hora e minuto). Esse algoritmo deve calcular e mostrar a duração do jogo (horas e minutos) sabendo-se que o tempo máximo de duração do jogo é de 24 horas e que o jogo pode iniciar em um dia e terminar no dia seguinte.

15. Desenvolva um algoritmo que receba cinco valores numérico inteiro, identifique e mostre o maior e o menor dentre esses números. Proibida em qualquer hipótese a ordenação dos valores.

16. Desenvolva um algoritmo que receba um caracter alfanumérico, verifique e mostre se o caracter digitado é uma vogal, uma consoante, um número, ou qualquer outro caracter.

17. Desenvolva um algoritmo que receba um valor numérico real, verifique e mostre se esse valor é positivo ou negativo.

18. Desenvolva um algoritmo que receba dois valores numérico real, verifique qual o maior desses valores e calcule e mostre a soma do cubo do maior número com o cubo do menor número.

19. Desenvolva um algoritmo que receba três valores numérico real, calcule e mostre a média ponderada desses números, considerando que o primeiro número tem peso 2, o segundo número tem peso 3 e o terceiro número tem peso 5.

20. Desenvolva um algoritmo que receba o preço atual e a venda mensal média de um produto, calcule e mostre o novo preço, sabendo que:

- se a venda média mensal for menor que 500 e o preço atual menor que 30,00, então o produto sofre um aumento de 10%;

- se a venda média mensal for maior ou igual a 500 e menor que 1000 e o preço atual maior ou igual a 30,00 e menor que 80,00, então o produto sofre um aumento de 15%;

- se a venda mensal média for maior ou igual a 1000 e o preço atual maior ou igual a 80,00, então o produto sofre uma diminuição de 5%.

21. Desenvolva um algoritmo que receba dois valores numérico inteiro, verifique e mostre se o maior número é múltiplo do menor número.

22. Desenvolva um algoritmo que receba dois valores número inteiro, verifique e mostre se o maior número é divisível pelo menor número.

23. Desenvolva um algoritmo que receba o número de voltas dadas, a extensão do circuito, em metros, e o tempo de duração, em minutos, de uma corrida de fórmula 1. O algoritmo deve calcular e mostrar a velocidade média do primeiro colocado, em quilômetros por hora.

24. Desenvolva um algoritmo que receba o salário de um funcionário, calcule e mostre o valor do aumento salarial e o salário final a ser recebido pelo funcionário, considerando que se o funcionário:

- recebe menos que 500,00, terá reajuste de 100%;

- recebe mais ou igual a 500,00 e menos que 1.000,00, terá reajuste de 75%;

- recebe mais ou igual a 1.000,00 e menos que 1.500,00, terá reajuste de 50%;

- recebe mais ou igual a 1.500,00 e menos que 2.000,00, terá reajuste de 25%;

- recebe mais ou igual a 2.000,00 e menos que 3.000,00, terá reajuste de 10%;

- recebe mais ou igual a 3.000,00 e menos que 5.000,00, terá reajuste de 5%;

- recebe mais ou igual a 5.000,00, terá reajuste de 2%.

25. Desenvolva um algoritmo que receba a medida de um ângulo em graus, verifique se é um ângulo válido (entre 0 grau e 360 graus) calcule e mostre o quadrante em que esse ângulo se localiza. Considere os quadrantes da trigonometria, primeiro quadrante graus entre 0 e 90, segundo quadrante graus entre 90 e 180, terceiro quadrante graus entre 180 e 270 e quarto quadrante graus entre 270 e 360.

26. Desenvolva um algoritmo que receba a medida de um ângulo em graus, calcule e mostre o quadrante em que esse ângulo se localiza. Considere os quadrantes da trigonometria e para ângulos maiores que 360 graus ou menores que –360 graus, reduzi-los, mostrando também o número de voltas e o sentido da volta (horário ou anti-horário).

27. Desenvolva um algoritmo que receba uma senha de quatro números, verifique a validade desta senha sabendo que a senha correta é 1234 e mostre ao usuário uma mensagem dizendo se a senha digitada é válida ou não.

28. Desenvolva um algoritmo que receba o tipo de investimento (poupança ou fundo de renda fixa) e o valor do investimento, calcule e mostre o valor corrigido do investimento após o período de 30 dias, considerando que o rendimento mensal da poupança é de 3% e o do fundo de renda fixa é de 5%.

29. Desenvolva um algoritmo que receba a data de nascimento de uma pessoa, verifique e mostre em qual estação do ano (primavera, verão, outono ou inverno) essa pessoa nasceu.

30. Desenvolva um algoritmo que receba a data de nascimento de uma pessoa, verifique e mostre qual o signo dessa pessoa. Usar estrutura de decisão encadeada.

# Capítulo 7 – Estrutura de Múltipla Escolha

Uma estrutura de múltipla escolha também é uma estrutura de decisão em que um fluxo de controle é utilizado para decidir qual o fluxo que o algoritmo seguirá. Uma estrutura de múltipla escolha determina qual conjunto de comandos ou bloco será executado após uma opção ser avaliada e um caso para essa opção for detectado. Essa opção é representada por valores de quaisquer tipos de dados conhecidos. Note que somente um caso é executado, dentre todos os casos da estrutura de múltipla escolha.

Uma estrutura de múltipla escolha pode ser construída dentro de uma estrutura de decisão, seja ela simples, composta ou encadeada, assim como qualquer estrutura de decisão pode ser construída dentro de uma estrutura de múltipla escolha.

Uma estrutura de múltipla escolha pode ser simples ou encadeada. Veremos a seguir como e quando trabalhar com cada uma delas.

## 7.1 Estrutura de Múltipla Escolha Simples

Uma estrutura de decisão de múltipla escolha pode ser utilizada quando o algoritmo precisa testar um conjunto de valores diferentes antes de executar um conjunto de comandos associado a esses valores. Nesse caso, uma opção é avaliada, seu resultado é verificado dentro de um conjunto de casos e o conjunto de comandos dentro do caso relacionado à opção avaliada é executado como parte do fluxo de execução do algoritmo. Numa estrutura de múltipla escolha, o tipo de dado da opção precisa, necessariamente, ser do mesmo tipo dos casos relacionados.

### Pseudocódigo

Na estrutura de múltipla escolha simples, utilizaremos as palavras **escolha**, **caso** e **caso contrário** que representam as palavras principais dessa estrutura e a palavra **fimescolha;** para determinar o fim do bloco de execução dessa estrutura. A estrutura de múltipla escolha simples para pseudocódigo segue a seguinte regra sintática:

**escolha (<opção>)**

**caso <caso\_1> : <comandos\_1>;**

**caso <caso\_2> : <comandos\_2>;**

**...**

**caso <caso\_n> : <comandos\_n>;**

**caso contrário : <mensagem>;**

**fimescolha;**

Por exemplo:

**escolha** (var)

**caso** 1 **:** escrever ("O valor da variável var é 1")**;**

**caso** 2 **:** escrever ("O valor da variável var é 2")**;**

**caso** 3 **:** escrever ("O valor da variável var é 3")**;**

**caso contrário :** escrever ("O valor da variável var não é nem 1, nem 2, nem 3")**;**

**fimescolha;**

Nesse exemplo, a variável **var** possui um valor **numérico\_inteiro** e vai passar por uma avaliação, ou seja, o valor da variável **var** será avaliada. Se o resultado de retorno dessa avaliação for o número inteiro 1, então o comando **escrever(...);** do **caso 1** será executado, se for o número inteiro 2, então o comando **escrever(...);** do **caso 2** será executado, se for o número inteiro 3, então o comando **escrever(...);** do **caso 3** será executado e se for diferente de 1,2 ou 3, então o **escrever(...);** do **caso contrário** será executado. Note que somente um dos casos é executado e que o tipo de dado da variável **var** é **numérico\_inteiro** assim como os valores de cada um dos casos.

Note que o uso do comando **caso contrário** serve para os valores não previstos que a variável **var** possa assumir.

O resultado da execução desse exemplo é:

// se o valor de **var** for 1

O valor da variável var é 1

// se o valor de **var** for 2

O valor da variável var é 2

// se o valor de **var** for 3

O valor da variável var é 3

// se o valor de **var** for 4

O valor da variável var não é nem 1, nem 2, nem 3

Numa estrutura de múltipla escolha, os valores escolhidos para cada caso não precisam ser únicos, isto é, cada caso da estrutura pode ser representado por um conjunto de valores específicos. Nesse caso, a estrutura de múltipla escolha simples para pseudocódigo segue a seguinte regra sintática:

**escolha (<opção>)**

**caso <caso\_1> : <comandos\_1>;**

**caso <caso\_2>, <caso\_3>, <caso\_4> : <comandos\_2>;**

**caso <caso\_5> .. <caso\_10> : <comandos\_3>;**

**...**

**caso <caso\_11>, <caso\_15> .. <caso\_20> : <comandos\_4>;**

**caso contrário : <mensagem>;**

**fimescolha;**

Por exemplo, um caso pode ter o valor 1, outro caso, os valores 2, 3 e 4, outro caso, os valores 7, 15 e 25 e outro caso, os valores 8, 11, 12, 13 e 14:

**escolha** (var)

**caso** 1 **:** escrever ("O valor da variável var é 1");

**caso** 2 **. .** 4 **:** escrever ("O valor da variável var pode ser 2, 3 ou 4");

**caso** 7, 15, 25 **:** escrever ("O valor da variável var pode ser 7, 15 ou 25");

**caso** 8, 11 **. .** 14 **:** escrever ("O valor da variável var pode ser 8 , 11, 12, 13 ou 14");

**caso contrário :** escrever ("opção inválida.");

**fimescolha;**

Nesse exemplo, a variável **var** possui um valor **numérico\_inteiro** e vai passar por uma avaliação, ou seja, o valor da variável **var** será avaliado. Se o resultado de retorno dessa avaliação for o número inteiro 1, então o comando **escrever(...);** do **caso 1** será executado, se for o número inteiro 2, 3 ou 4, então o comando **escrever(...);** do **caso 2 .. 4** será executado, se for o número inteiro 7, 15, ou 25, então o comando **escrever(...);** do **caso 7, 15, 25** será executado, se for 8, 11, 12, 13 ou 14, então o comando **escrever(...);** do **caso 8, 11 .. 14** será executado e se for diferente de qualquer um dos valores anteriores, então o **escrever(...);** do **caso contrário** será executado. Note que somente um dos casos é executado e que o tipo de dado da variável **var** é **numérico\_inteiro** assim como os valores de cada um dos casos.

O resultado da execução desse exemplo é:

// se o valor de **var** for 1

O valor da variável var é 1

// se o valor de **var** for 2

O valor da variável var pode ser 2, 3 ou 4

// se o valor de **var** for 7

O valor da variável var pode ser 7, 15 ou 25

// se o valor de **var** for 8

O valor da variável var pode ser 8 , 11, 12, 13 ou 14

// se o valor de **var** for 100

Opção Inválida

### Java

Na estrutura de múltipla escolha simples, utilizaremos as palavras **switch**, **case**, **default** e **break** que representam as palavras principais dessa estrutura e a chave aberta, **{**, e a chave fechada, **}**, para representar o conjunto de comandos que fazem parte dessa estrutura. A estrutura de múltipla escolha simples para Java segue a seguinte regra sintática:

**switch (<opção>)**

**case <caso\_1> : <comandos\_1>;**

**break;**

**case <caso\_2> : <comandos\_2>;**

**break;**

**...**

**case <caso\_n> : <comandos\_n>;**

**break;**

**default : <mensagem>;**

**fimescolha;**

Por exemplo:

**switch** (var)

**{**

**case** 1 **:** System.out.print ("O valor da variável var é 1")**;**

**break;**

**case** 2 **:** System.out.print ("O valor da variável var é 2")**;**

**break;**

**case** 3 **:** System.out.print ("O valor da variável var é 3")**;**

**break;**

**default :** System.out.print ("O valor da variável var não é  
 nem 1, nem 2, nem 3")**;**

**}**

Nesse exemplo, a variável **var** possui um valor **numérico\_inteiro** e vai passar por uma avaliação. Se o resultado de retorno dessa avaliação for o número inteiro 1, então o comando **System.out.print(...);** do **case 1** será executado, se for o número inteiro 2, então o comando **System.out.print (...);** do **case 2** será executado, se for o número inteiro 3, então o comando **System.out.print (...);** do **case 3** será executado e se for diferente de 1,2 ou 3, então o **System.out.print (...);** do **default** será executado. Note que somente um dos casos é executado e que o tipo de dado da variável **var** é **numérico\_inteiro** assim como os valores de cada um dos casos.

Note que o uso do comando **break** serve para sair de determinado bloco. Nesse caso, o **break** é utilizado para sair da estrutura **switch/case** após executar os comandos do **case** selecionado. Isso é necessário, pois sem o uso do **break**, todos os comandos após o **case** selecionado seriam executados.

Note que o uso do comando **default** serve para os valores não previstos que a variável **var** possa assumir.

O resultado da execução desse exemplo é:

// se o valor de **var** for 1

O valor da variável var é 1

// se o valor de **var** for 2

O valor da variável var é 2

// se o valor de **var** for 7

O valor da variável var é 3

// se o valor de **var** for 4

O valor da variável var não é nem 1, nem 2, nem 3

Numa estrutura de **switch case**, os valores escolhidos para cada caso não precisam ser únicos, isto é, cada caso da estrutura pode ser representado por um conjunto de valores específicos. Nesse caso, a estrutura de múltipla escolha simples para pseudocódigo segue a seguinte regra sintática:

**switch (<opção>)**

**{**

**case <caso\_1> : <comandos\_1>;**

**break;**

**case <caso\_2> : case <caso\_3> : case <caso\_4> : <comandos\_2>;**

**break;**

**case <caso\_5> : case <caso\_6> : case <caso\_7> : case <caso\_8> :   
 case <caso\_9> : case <caso\_10> : <comandos\_3>;**

**break;**

**...**

**case <caso\_11> : case <caso\_15> : case <caso\_16> : case <caso\_17> :   
 case <caso\_18> : <comandos\_4>;**

**break;**

**default : <mensagem>;**

**}**

Por exemplo, um caso pode ter o valor 1, outro caso, os valores 2, 3 e 4, outro caso, os valores 7, 15 e 25 e outro caso, os valores 8, 11, 12, 13 e 14:

**switch** (var)

{

**case** 1 **:** System.out.println ("O valor da variável var é 1");

**break;**

**case** 2 **:**

**case** 3 **:**

**case** 4 **:** System.out.println ("O valor da variável var pode ser  
 2, 3 ou 4");

**break;**

**case** 7 **:**

**case** 15**:**

**case** 25**:** System.out.println ("O valor da variável var pode ser  
 7, 15 ou 25");

**break;**

**case** 8 **:**

**case** 11**:**

**case** 12**:**

**case** 13**:**

**case** 14**:** System.out.println ("O valor da variável var pode ser  
 8 , 11, 12, 13 ou 14");

**break;**

**default :** System.out.println ("opção inválida.");

**}**

Nesse exemplo, a variável **var** possui um valor **int** e vai passar por uma avaliação, ou seja, o valor da variável **var** será avaliado. Se o resultado de retorno dessa avaliação for o número inteiro 1, então o comando **System.out.println(...);** do **case 1 :** será executado, se for o número inteiro 2, 3 ou 4, então o comando **System.out.println(...);** do **case 2 : case 3: case 4 :** será executado, se for o número inteiro 7, 15, ou 25, então o comando **System.out.printon(...);** do **case 7 : case 15 : case 25 :** será executado, se for 8, 11, 12, 13 ou 14, então o comando **System.out.println(...);** do **case 8 : case 11 : case 12 : case 13 : case 14 :** será executado e se for diferente de qualquer um dos valores anteriores, então o **System.out.println(...);** do **default** será executado. Note que somente um dos casos é executado e que o tipo de dado da variável **var** é **int** assim como os valores de cada um dos casos.

O resultado da execução desse exemplo é:

// se o valor de **var** for 1

O valor da variável var é 1

// se o valor de **var** for 2

O valor da variável var pode ser 2, 3 ou 4

// se o valor de **var** for 7

O valor da variável var pode ser 7, 15 ou 25

// se o valor de **var** for 8

O valor da variável var pode ser 8 , 11, 12, 13 ou 14

// se o valor de **var** for 100

Opção Inválida

## 7.2 Estrutura de Múltipla Escolha Encadeada

Uma estrutura de múltipla escolha encadeada pode ser utilizada quando o algoritmo precisa testar um conjunto de valores diferentes antes de executar um conjunto de comandos associados a esses valores. Dizemos que essa estrutura é encadeado, pois há estruturas de múltipla escolha dentro de outras estruturas de múltipla escolha. Para qualquer caso, cada opção é avaliada e para cada resultado, um conjunto de comandos dentro da estrutura **escolha/caso** pode ser executado. Não existe limite para a estrutura de múltipla escolha encadeada, podemos ter quantas estruturas de múltipla escolha encadeadas forem necessárias. Note que, para qualquer estrutura de múltipla escolha, no máximo, um único conjunto de comando será executado, aquele cuja avaliação de opção for escolhida.

### Pseudocódigo

Na estrutura de múltipla escolha encadeada, utilizaremos as palavras **escolha**, **caso** e **caso contrário** que representam as palavras principais dessa estrutura e a palavra **fimescolha;** para determinar o fim do bloco de execução dessa estrutura. Note que, para cada estrutura **escolha/caso** que aparece, existe um **fimescolha** adequadamente localizado dentro da estrutura. A estrutura de múltipla escolha encadeada para pseudocódigo segue a seguinte regra sintática:

**escolha (<opção\_1>)**

**caso <caso\_1> : escolha (<opção\_2>)**

**caso <caso\_11> : escolha (<opção\_3>)**

**caso <caso\_111> : <comandos\_111>;**

**caso <caso\_112> : <comandos\_112>;**

**...**

**caso <caso\_11n> : <comandos\_11n>;**

**caso contrário : <mensagem>;**

**fimescolha; // opção 3**

**caso <caso\_12> : <comandos\_12>;**

**...**

**caso <caso\_1n> : <comandos\_1n>;**

**caso contrário : <mensagem>;**

**fimescolha; // opção 2**

**caso <caso\_2> : escolha (<opção\_4>)**

**caso <caso\_21> : <comandos\_21>;**

**caso <caso\_22> : <comandos\_22>;**

**...**

**caso <caso\_2n> : <comandos\_2n>;**

**caso contrário : <mensagem>;**

**fimescolha; // opção 4**

**caso <caso\_3> : <comandos\_3>;**

**...**

**caso <caso\_n> : <comandos\_n>;**

**caso contrário : <mensagem>;**

**fimescolha; // opção 1**

Por exemplo:

**escolha** (var)

**caso** 1 **. .** 3 **:** **escolha** (var)

**caso** 1 , 2 **:** **escolha** (var)

**caso** 1 **:** escrever ("o valor da variável var é 1");

**caso** 2 **:** escrever ("o valor da variável var é 2");

**caso contrário :** escrever ("opção inválida");

**fimescolha**;

**caso** 3 **:** escrever ("o valor da variável var é 3");

**caso contrário** **:** escrever ("opção inválida");

**fimescolha**;

**caso** 4 , 5 **:** **escolha** (var)

**caso** 4 **:** escrever ("o valor da variável var é 4");

**caso** 5 **:** escrever ("o valor da variável var é 5");

**caso contrário** **:** escrever("opção inválida");

**fimescolha**;

**caso** 6 **:** escrever ("o valor da variável var é 6");

**caso contrário :** escrever ("opção inválida");

**fimescolha**;

Nesse exemplo, a variável **var** possui um valor **numérico\_inteiro** e vai passar por uma avaliação, ou seja, o valor da variável **var** será avaliado. Se o resultado de retorno dessa avaliação for o número inteiro 6, então o comando **escrever(...);** do **caso 6** será executado, se for o número inteiro 4 ou 5, então uma nova estrutura de múltipla escolha é avaliado. Se nessa avaliação, o valor de **var** for o número 4, o comando **escrever(...);** do **caso 4** será executado, se for o número 5, o comando **escrever(...);** do **caso 5** será executado. Se o resultado da avaliação do primeiro **var** for o número inteiro 1, 2, ou 3, então uma nova estrutura de múltipla escolha é avaliada e assim por diante. Note que somente um dos casos é executado, ou seja, uma única mensagem **escrever(...);** será apresentada ao usuário e que o tipo de dado da variável **var** é **numérico\_inteiro** assim como os valores de cada um dos casos.

O resultado da execução desse exemplo é:

// se o valor de **var** for 1

O valor da variável var é 1

// se o valor de **var** for 2

O valor da variável var é 2

// se o valor de **var** for 3

O valor da variável var é 3

// se o valor de **var** for 4

O valor da variável var é 4

// se o valor de **var** for 5

O valor da variável var é 5

// se o valor de **var** for 6

O valor da variável var é 6

// se o valor de **var** for 7

Opção Inválida

### Java

Na estrutura de múltipla escolha encadeada, utilizaremos as palavras **switch**, **case**, **default** e **break** que representam as palavras principais dessa estrutura e a chave aberta, **{**, e a chave fechada, **}**, para representar o conjunto de comandos que fazem parte dessa estrutura. A estrutura de múltipla escolha encadeada para Java segue a seguinte regra sintática:

**switch (<opção\_1>)**

**{**

**case <caso\_1> : switch (<opção\_2>)**

**{**

**case <caso\_11> : switch (<opção\_3>)**

**{**

**case <caso\_111> : <comandos\_111>;**

**break;**

**case <caso\_112> : <comandos\_112>;**

**break;**

**...**

**case <caso\_11n> : <comandos\_11n>;**

**break;**

**default : <mensagem>;**

**} // opção 3**

**break;**

**case <caso\_12> : <comandos\_12>;**

**break;**

**...**

**case <caso\_1n> : <comandos\_1n>;**

**break;**

**default : <mensagem>;**

**} // opção 2**

**break;**

**case <caso\_2> : switch (<opção\_4>)**

**{**

**case <caso\_21> : <comandos\_21>;**

**break;**

**case <caso\_22> : <comandos\_22>;**

**break;**

**...**

**case <caso\_2n> : <comandos\_2n>;**

**break;**

**default : <mensagem>;**

**} // opção 4**

**break;**

**case <caso\_3> : <comandos\_3>;**

**break;**

**...**

**case <caso\_n> : <comandos\_n>;**

**break;**

**default : <mensagem>;**

**} // opção 1**

Por exemplo:

**switch** (var)

**{**

**case** 1 **:**

**case** 2 **:**

**case** 3 **:**

**switch** (var)

**{**

**case** 1 **:**

**case** 2 **:**

**switch** (var)

**{**

**case** 1 **:** System.out.print ("o valor da variável var é  
 1");

**break**;

**case** 2 **:** System.out.print ("o valor da variável var é  
 2");

**break**;

**default :** System.out.print ("opção inválida");

**}**

**break;**

**case** 3 **:** System.out.print ("o valor da variável var é 3");

**break;**

**default** **:** System.out.print ("opção inválida");

**}**

**break;**

**case** 4 **:**

**case** 5 **:**

**switch** (var)

**{**

**case** 4 **:** System.out.print ("o valor da variável var é 4");

**break;**

**case** 5 **:** System.out.print ("o valor da variável var é 5");

**break;**

**default** **:** System.out.print ("opção inválida");

**}**

**break;**

**case** 6 **:** System.out.print ("o valor da variável var é 6");

**break;**

**default :** System.out.print ("opção inválida");

**}**

Nesse exemplo, a variável **var** possui um valor **numérico\_inteiro** e vai passar por uma avaliação, ou seja, o valor da variável **var** será avaliado. Se o resultado de retorno dessa avaliação for o número inteiro 6, então o comando **System.out.print(...);** do **case 6** será executado, se for o número inteiro 4 ou 5, então uma nova estrutura do **switch/case** é avaliada. Se nessa avaliação o valor de **var** for o número 4, o comando **System.out.print(...);** do **case 4** será executado, se for o número 5, o comando **System.out.print(...);** do **case 5** será executado. Se o resultado da avaliação do primeiro **var** for o número inteiro 1, 2, ou 3, então uma nova estrutura do **switch/case** é avaliada e assim por diante. Note que somente um dos casos é executado, ou seja, uma única mensagem **System.out.print(...);** será apresentada ao usuário e que o tipo de dado da variável **var** é **numérico\_inteiro** assim como os valores de cada um dos casos.

O resultado da execução desse exemplo é:

// se o valor de **var** for 1

O valor da variável var é 1

// se o valor de **var** for 2

O valor da variável var é 2

// se o valor de **var** for 3

O valor da variável var é 3

// se o valor de **var** for 4

O valor da variável var é 4

// se o valor de **var** for 5

O valor da variável var é 5

// se o valor de **var** for 6

O valor da variável var é 6

// se o valor de **var** for 7

Opção Inválida

## 7.3 Exemplos de Estrutura de Múltipla Escolha em pseudocódigo

1. Desenvolva um algoritmo que receba quatro notas bimestrais, calcule e mostre a média aritmética dessas quatro notas, bem como, se o aluno foi aprovado (média >= 7), reprovado (média < 3), em exame (média >= 3 ou média < 7) ou aprovado com louvor (média = 10).

**Algoritmo** MediaAritmetica

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

n1, n2, n3, n4, media **numérico\_real**;

// mensagem ao usuário

escrever ("Digite as quatro notas bimestrais");

// entrada de dados

ler (n1 , n2 , n3 , n4);

// processamento de dados

media ← (n1 + n2 + n3 + n4) / 4;

// saída de resultados

escrever ("A média é: " , media);

// processamento de dados

**escolha** (media)

**caso** 7 .. 10 **:** **se** (media = 10)

**então**

// saída de resultados

escrever("Aluno aprovado com louvor");

**senão**

// saída de resultados

escrever ("Aluno aprovado");

**fimse**;

**caso** 0 .. 2.9 **:** // saída de resultados

escrever ("Aluno reprovado");

**caso** 3 .. 6.9 **:** // saída de resultados

escrever ("Aluno em exame");

**caso contrário :** // saída de resultados

escrever ("média inválida");

**fimescolha**;

**fim\_algoritmo**.

2. Desenvolva um algoritmo que receba o preço de um produto e seu código de origem e mostre o preço do produto junto de sua procedência, conforme tabela abaixo:

|  |  |
| --- | --- |
| código de origem | região de procedência |
| 1 | norte |
| 2, 5, 9 | sul |
| 3 , 10 até 15 | leste |
| 7 ou 20 | oeste |
| qualquer outro | importado |

**Algoritmo** Produto

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

preco **numérico\_real**;

codigo **numérico\_inteiro**;

// mensagem ao usuário

escrever ("Digite o preço e o código de origem do produto");

// entrada de dados

ler (preco, codigo);

// processamento de dados

**escolha** (codigo)

**caso** 1 **:** // entrada de dados

escrever (preco , " - Norte");

**caso** 2, 5, 9 **:** // saída de resultados

escrever (preco ," - Sul");

**caso** 3, 10 .. 15 **:** // saída de resultados

escrever (preco , " - Leste");

**caso** 7 , 20 **:** // saída de resultados

escrever (preco , "- Oeste");

**caso contrário :** // saída de resultados

escrever (preco , "- Importado");

**fimescolha**;

**fim\_algoritmo**.

3. Desenvolva um algoritmo que receba dois valores numérico inteiro e o símbolo da operação conforme tabela abaixo, calcule e mostre a operação efetuada:

|  |  |
| --- | --- |
| Símbolo da operação | Nome da operação |
| + | adição |
| - | subtração |
| \* | multiplicação |
| / | divisão |

**Algoritmo** Cálculo

**início\_algoritmo**

// declaração de variáveis e/ou constantes

**Declarar**

d **numérico\_real**;

res, num1, num2 **numérico\_inteiro**;

oper **alfanumérico**;

// mensagem ao usuário

escrever ("Digite dois inteiros e a operação");

// entrada de dados

ler (num1, num2, oper);

// processamento de dados

**escolha** (oper)

**caso** '+' **:** res ← num1 + num2;

// saída de resultados

escrever ("A soma de " , num1 , " com " , num2 , " é " , res);

**caso** '-' **:** res ← num1 - num2;

// saída de resultados

escrever ("A diferença de " , num1 , " com " , num2 , " é " , res);

**caso** '\*' **:** res ← num1 \* num2;

// saída de resultados

escrever ("O produto de " , num1 , " com " , num2 , " é " , res);

**caso** '/' **:** d ← num1 / num2;

// saída de resultados

escrever ("A divisão de " , num1 , " com " , num2 , " é " , d);

**caso contrário :** // saída de resultados

escrever ("operação inválida");

**fimescolha**;

**fim\_algoritmo**.

## 7.4 Exemplos de Estrutura de Múltipla Escolha em Java

1. Desenvolva um algoritmo que receba quatro notas bimestrais, calcule e mostre a média aritmética dessas quatro notas, bem como, se o aluno foi aprovado (média >= 7), reprovado (média < 3), em exame (média >= 3 ou média < 7) ou aprovado com louvor (média = 10).

**class** MediaAritmetica

**{**

**public static void main(String arg [ ])**

**{**

// declaração de variáveis e/ou constantes

**double** n1, n2, n3, n4, media;

**int** op = 0;

// mensagem ao usuário e entrada de dados

n1 = Double.parseDouble(JOptionPane.showInputDialog("Digite  
 uma nota bimestral"));

n2 = Double.parseDouble(JOptionPane.showInputDialog("Digite  
 uma nota bimestral"));

n3 = Double.parseDouble(JOptionPane.showInputDialog("Digite  
 uma nota bimestral"));

n4 = Double.parseDouble(JOptionPane.showInputDialog("Digite  
 uma nota bimestral"));

// processamento de dados

media = (n1 + n2 + n3 + n4) / 4;

// saída de resultados

System.out.println ("A média é: " + media);

// processamento de dados

**if** (media >= 7 && media <= 10)

**{**

op = 1;

**}**

**else**

**{**

**if** (media < 7 && media >= 3)

**{**

op = 3;

**}**

**else**

**{**

**if** (media >= 0 && media < 3)

**{**

op = 2;

**}**

**}**

**}**

**switch** (op)

**{**

**case** 1 **:** **if** (media == 10)

**{**

// saída de resultados

System.out.println("Aluno aprovado com  
 louvor");

**}**

**else**

**{**

// saída de resultados

System.out.println ("Aluno aprovado");

**}**

**break;**

**case** 2 **:** // saída de resultados

System.out.println ("Aluno reprovado");

**break;**

**case** 3 **:** // saída de resultados

System.out.println ("Aluno em exame");

**break;**

**default :** // saída de resultados

System.out.println ("média inválida");

**}** // fim do switch

System.exit(0);

**}** // fim do void main

**}** // fim da classe

2. Desenvolva um algoritmo que receba o preço de um produto e seu código de origem e mostre o preço do produto junto de sua procedência, conforme tabela abaixo:

|  |  |
| --- | --- |
| Código de origem | região de procedência |
| 1 | norte |
| 2, 5, 9 | sul |
| 3 , 10 até 15 | leste |
| 7 ou 20 | oeste |
| qualquer outro | importado |

**class** Produto

**{**

**public static void main(String arg [ ])**

**{**

// declaração de variáveis e/ou constantes

**double** preco;

**int** codigo;

// mensagem ao usuário e entrada de dados

preco = Double.parseDouble(JOptionPane.showInputDialog("Digite  
 o preço do produto"));

codigo = Integer.parseInt(JOptionPane.showInputDialog("Digite  
 o código de origem do produto"));

// processamento de dados

**switch** (codigo)

**{**

**case** 1 **:** // saída de resultados

System.out.println (preco + " - Norte");

**break;**

**case** 2 **: case** 5 **: case** 9 **:**

// saída de resultados

System.out.println (preco + " - Sul");

**break;**

**case** 3 **: case** 10 **: case** 11 **: case** 12 **:**

**case** 13 **: case** 14 **: case** 15 **:**

// saída de resultados

System.out.println (preco + " - Leste");

**break;**

**case** 7 **: case** 20 **:**

// saída de resultados

System.out.println (preco + "- Oeste");

**break;**

**default :** // saída de resultados

System.out.println (preco + "- Importado");

**}** // fim do switch

System.exit(0);

**}** // fim do void main

**}** // fim da classe

3. Desenvolva um algoritmo que receba dois valores numérico inteiro e o símbolo da operação conforme tabela abaixo, calcule e mostre a operação efetuada:

|  |  |
| --- | --- |
| Símbolo da operação | Nome da operação |
| + | adição |
| - | subtração |
| \* | multiplicação |
| / | divisão |

**class** Calculo

**{**

**public static void main(String arg [ ])**

**{**

// declaração de variáveis e/ou constantes

**double** d;

**int** res, num1, num2, oper;

// mensagem ao usuário e entrada de dados

num1 = Integer.parseInt(JOptionPane.showInputDialog("Digite  
 um número inteiro"));

num2 = Integer.parseInt(JOptionPane.showInputDialog("Digite  
 um número inteiro"));

oper = Integer.parseInt(JOptionPane.showInputDialog("Digite  
 a operação: \n 1 para somar \n   
 2 para subtrair \n 3 para   
 multiplicar \n 4 para dividir"));

// processamento de dados

**switch** (oper)

**{**

**case** 1 **:** res = num1 + num2;

// saída de resultados

System.out.println ("A soma de " + num1 +   
 " com " + num2 + " é " + res);

**break;**

**case** 2 **:** res = num1 - num2;

// saída de resultados

System.out.println ("A diferença de " + num1 +  
 " com " + num2 + " é " + res);

**break;**

**case** 3 **:** res = num1 \* num2;

// saída de resultados

System.out.println ("O produto de " + num1 +   
 " com " + num2 + " é " + res);

**break;**

**case** 4 **:** d = num1 / num2;

// saída de resultados

System.out.println ("A divisão de " + num1 +   
 " com " + num2 + " é " + d);

**break;**

**default :** // saída de resultados

System.out.println ("operação inválida");

**}** // fim do switch

System.exit(0);

**}** // fim do void main

**}** // fim da classe

## 7.5 Exercícios de Estrutura de Múltipla Escolha Simples e Encadeada

1. Desenvolva um algoritmo que receba o nome e os dados para cálculo da área de uma figura geométrica conforme tabela abaixo, calcule e mostre a área da figura geométrica:

|  |  |
| --- | --- |
| Figura geométrica | Fórmula |
| quadrado | lado \* lado |
| triângulo | (base \* altura) / 2 |
| retângulo | base \* altura |
| trapézio | ((Base maior + base menor) \* altura) / 2 |

2. Desenvolva um algoritmo que receba dois valores reais e o código do produto notável conforme tabela abaixo, calcule e mostre o valor do produto notável:

|  |  |  |
| --- | --- | --- |
| Código | Produto Notável | Fórmula |
| 1 | Quadrado da diferença de dois números | (a – b) \* (a – b) |
| 2 | Quadrado da soma de dois números | (a + b) \* (a + b) |
| 3 | soma do quadrado de dois números | a \* a + b \* b |
| 4 | Diferença do quadrado de dois números | a \* a – b \* b |
| 5 | produto da soma com a diferença de dois números | (a – b) \* (a + b) |

3. Desenvolva um algoritmo que receba o nome de um lugar e mostre para o usuário o que ele deve fazer nesse lugar, conforme tabela abaixo:

|  |  |
| --- | --- |
| Lugar | O que fazer |
| Escola | Estudar |
| Banco | Pagar contar |
| Farmácia | Comprar remédios |
| Casa | Descansar |
| Correio | Remeter cartas |

4. Desenvolva um algoritmo que receba um dia da semana e mostre qual(is) a(s) disciplina(s) você tem naquele dia da semana.

5. Desenvolva um algoritmo que receba um mês do ano e mostre qual(is) o(s) feriado(s) daquele mês.

6. Desenvolva um algoritmo que receba a data de nascimento de uma pessoa e mostre qual o signo dessa pessoa.

7. Desenvolva um algoritmo que receba o código de determinado produto e mostre a sua classificação, conforme a tabela abaixo:

|  |  |
| --- | --- |
| Código | Classificação |
| 10 ou 11 | alimento não perecível |
| 12 | alimento perecível |
| 13, 14 ou 15 | vestuário |
| 19, 26 até 30 | higiene pessoal |
| 17 até 25 | limpeza e utensílios domésticos |
| Qualquer outro código | inválido |

8. Desenvolva um algoritmo que receba a idade de um atleta e mostre a sua classificação em categorias, conforme a tabela abaixo:

|  |  |
| --- | --- |
| Idade | Categoria |
| 3 até 4 anos | Mirim |
| 5 até 7 anos | Infantil A |
| 8 até 10 anos | Infantil B |
| 11 até 13 anos | Juvenil A |
| 14 até 17 anos | Juvenil B |
| 18 até 50 anos | Adulto |
| mais que 50 | Veterano |

9. Desenvolva um algoritmo que receba o código da condição de pagamento e o preço de um produto, calcule o que deve ser pago pelo produto e mostre a condição de pagamento e o preço a ser pago, conforme a tabela abaixo:

|  |  |
| --- | --- |
| Código | Condição de pagamento |
| A | à vista em dinheiro ou cheque tem 20% de desconto |
| B | à vista em cartão de crédito tem 10% de desconto |
| C | em 2 vezes, preço normal de etiqueta sem juros |
| D | em 3 vezes, preço normal de etiqueta mais juros de 15% |

10. Desenvolva um algoritmo que receba o valor de dois números inteiros, o símbolo da operação aritmética desejada, calcule e mostre o resultado da operação aritmética, conforme a tabela abaixo:

|  |  |
| --- | --- |
| símbolo | Operação aritmética |
| + | adição |
| - | subtração |
| \* | Multiplicação |
| / | divisão |
| ^ | Potenciação |
| m | Resto da divisão |
| q | Quociente da divisão |

11. Desenvolva um algoritmo que receba o número de horas extras de um funcionário e o número de horas que um funcionário faltou ao trabalho, calcule e mostre a gratificação de natal desse funcionário sabendo que o valor da gratificação é calculada conforme fórmula e tabela abaixo:

G = (número de horas extras) – 2/3 \* (número de horas que faltou)

|  |  |
| --- | --- |
| G | gratificação |
| maior ou igual a 5.000,00 | 100,00 |
| menor que 5.000,00 e maior ou igual que 2.500,00 | 200,00 |
| menor que 2.500,00 e maior ou igual que 1.000,00 | 300,00 |
| menor que 1.000,00 e maior ou igual que 500 | 400,00 |
| menor que 500,00 | 500,00 |

12. Desenvolva um algoritmo que receba a idade e o peso de uma pessoa, verifique e mostre em qual grupo de risco essa pessoa se encaixa, conforme a tabela abaixo:

|  |  |  |  |
| --- | --- | --- | --- |
| Idade | Peso | | |
| Até 60 (inclusive) | Entre 60 e 90 (inclusive) | acima de 90 |
| menores de 20 | 9 | 8 | 7 |
| de 20 a 50 | 6 | 5 | 4 |
| maiores de 50 | 3 | 2 | 1 |

13. Desenvolva um algoritmo que receba o código do produto comprado e a quantidade comprada de um produto, calcule e mostre o preço unitário do produto, conforme a tabela abaixo, e o preço total da nota:

|  |  |
| --- | --- |
| Código do produto | Preço do produto |
| 1 a 5 | 10,00 |
| 6, 8 a 15 | 50,00 |
| 7 | 15,00 |
| 16, 17 | 25,00 |

14. Desenvolva um algoritmo que receba o salário de um funcionário, calcule e mostre o valor do aumento salarial e o salário final a ser recebido pelo funcionário, considerando que:

- se o funcionário recebe menos que 500,00, terá reajuste de 100%;

- se o funcionário recebe mais ou igual a 500,00 e menos que 1.000,00, terá reajuste de 75%;

- se o funcionário recebe mais ou igual a 1.000,00 e menos que 1.500,00, terá reajuste de 50%;

- se o funcionário recebe mais ou igual a 1.500,00 e menos que 2.000,00, terá reajuste de 25%;

- se o funcionário recebe mais ou igual a 2.000,00 e menos que 3.000,00, terá reajuste de 10%;

- se o funcionário recebe mais ou igual a 3.000,00 e menos que 5.000,00, terá reajuste de 5%;

- se o funcionário recebe mais que 5.000,00, terá reajuste de 2%.

15. Desenvolva um algoritmo que receba o tipo de investimento e o valor do investimento, calcule e mostre o valor corrigido do investimento após o período de 30 dias, considerando que o rendimento mensal para cada tipo de investimento varia conforme a tabela abaixo:

|  |  |
| --- | --- |
| Tipo de investimento | Rendimento |
| Poupança | 3% |
| Fundo de renda fixa | 5% |
| CDB | 6% |
| Ações | 10% |

16. Desenvolva um algoritmo que receba a data de nascimento de uma pessoa, verifique e mostre em qual estação do ano (primavera, verão, outono ou inverno) esta pessoa nasceu.

17. Desenvolva um algoritmo que receba três valores numérico real e um código e mostre-os conforme a tabela abaixo:

|  |  |
| --- | --- |
| código | Operação |
| 1 | apresentar os três valores em ordem crescente |
| 2 | apresentar os três valores em ordem decrescente |
| 3 | apresentar somente os valores pares |
| 4 | apresentar somente os valores ímpares |
| 5 | apresentar somente os valores positivos |
| 6 | apresentar somente os valores negativos |

18. Desenvolva um algoritmo que receba o preço atual e a venda mensal média de um produto, calcule e mostre o novo preço, sabendo que:

- se a venda média mensal for menor que 500 e o preço atual menor que 30,00, então o produto sofre um aumento de 10%;

- se a venda média mensal for maior ou igual a 500 e menor que 1000 e o preço atual maior ou igual a 30,00 e menor que 80,00, então o produto sofre um aumento de 15%;

- se a venda mensal média for maior ou igual a 1000 e o preço atual maior ou igual a 80,00, então o produto sofre uma diminuição de 5%.

18. Desenvolva um algoritmo que receba o raio de uma circunferência e um código, calcule e mostre os dados da circunferência, conforme a tabela abaixo:

|  |  |
| --- | --- |
| Código | Operação |
| x | calcula o comprimento: C = 2 . π . raio |
| y | calcula a área: A = π . raio2 |
| z | calculo o volume: V = 4/3 . π . raio3 |

20. Desenvolva um algoritmo que receba um caracter, verifique e mostre se o caracter digitado é uma vogal, uma consoante, ou qualquer outro caracter.

21. Desenvolva um algoritmo que recebe o nome de uma pessoa, verifique e mostre qual a sua cor e fruta preferida, conforme a tabela abaixo:

|  |  |  |
| --- | --- | --- |
| Nome | Preferência | |
| Cor | fruta |
| Maria | vermelho | morango |
| José | azul | figo |
| João | verde | uva |

22. Desenvolva um algoritmo que recebe o nome de um País, verifique e mostre como características a sua capital, continente a que pertence e idioma oficial, conforme tabela abaixo:

|  |  |  |  |
| --- | --- | --- | --- |
| País | Características | | |
| capital | continente | idioma |
| Brasil | Brasilia | América | Português |
| Japão | Tóquio | Ásia | Japonês |
| Itália | Roma | Europa | Italiano |
| Argélia | Argel | África | Francês |
| Austrália | Canberra | Oceania | Inglês |

23. Desenvolva um algoritmo que recebe o nome de uma bebida, verifique e mostre como características dessa bebida uma fruta, um ingrediente e que dia da semana essa bebida é oferecida, conforme tabela abaixo:

|  |  |  |  |
| --- | --- | --- | --- |
| Bebida | Características | | |
| fruta | ingrediente | dia da semana |
| Magro | Abacaxi | Ginseng | Segunda-feira |
| Forte | Pêra | Geléia | Terça-feira |
| Vitaminado | Pêssego | Guaraná | Quarta-feira |
| Poderoso | Acerola | Germe de Trigo | Quinta-feira |
| Saudável | Laranja | Algas | Sexta-feira |

24. Desenvolva um algoritmo que recebe o nome de uma família, verifique e mostre o nome da fonte e o dia de instalação dessa fonte, conforme tabela abaixo:

|  |  |  |
| --- | --- | --- |
| Família | Fonte | |
| Nome | Instalação |
| Tompson | Oásis | Sábado |
| Dorneles | Floresta Tropical | Domingo |
| Aguiar | Gruta | Sábado |
| Santos | Pão de Açúcar | Domingo |

25. Desenvolva um algoritmo que recebe o número da etapa de uma viagem de barco, verifique e mostre o nome do barco, a distância percorrida e a vista dessa viagem, conforme tabela abaixo:

|  |  |  |  |
| --- | --- | --- | --- |
| Etapa | Viagem | | |
| barco | Distância | Vista |
| primeira | Netuno | 3 quilômetros | Canhão |
| segunda | Plutão | 4 quilômetros | Índios |
| terceira | Saturno | 5 quilômetros | Ruínas |
| quarta | Marte | 2 quilômetros | Igreja |
| quinta | Urano | 8 quilômetros | Gruta |

26. Desenvolva um algoritmo que recebe o nome de um museu, o horário de abertura, o horário de fechamento e o valor da entrada, mostre os dados do museu, quantas horas o museu fica aberto e o troco que uma pessoa deve receber se possui R$ 50,00 para pagar a entrada, conforme tabela abaixo:

|  |  |  |  |
| --- | --- | --- | --- |
| Museu | Dados | | |
| horário abertura | horário de fechamento | valor |
| Arte Moderna | 8 horas | 16 horas e 30 minutos | Canhão |
| Cartas e Manuscritos | 10 horas | 17 horas e 15 minutos | Índios |
| Instrumentos Musicais | 12 horas | 17 horas e 30 minutos | Ruínas |
| Pedras Preciosas | 14 horas | 18 horas e 10 minutos | Igreja |
| Relíquias | 16 horas | 18 horas e 45 minutos | Gruta |

27. Desenvolva um algoritmo que recebe o nome de uma vila, verifique e mostre o nome do rio que passa nessa vila e a extensão desse rio, conforme a tabela abaixo:

|  |  |  |
| --- | --- | --- |
| Vila | Rio | |
| Nome | Extensão |
| Vila Áurea | Rio Coral | 21 km |
| Vila do Retorno | Rio Grande | 24 km |
| Vila da Luz | Rio Mesquita | 28 km |
| Vila dos Pinheiros | Rio Ronco | 32 km |
| Vila Santana | Rio da Viúva | 33 km |

28. Desenvolva um algoritmo que recebe uma data, verifique e mostre qual atividade foi desenvolvida nesta data, como estava o tempo e a temperatura, conforme tabela abaixo:

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Dados | | |
| Atividade | tempo | temperatura |
| 17 de maio | Caminhada | Frente Fria | 16 graus |
| 18 de maio | Leitura | Nublado | 22 graus |
| 19 de maio | Computador | Chuva Fina | 18 graus |
| 20 de maio | Pesca | Sol | 25 graus |
| 21 de maio | Bicicleta | Vento | 20 graus |

29. Desenvolva um algoritmo que recebe um dia da semana, verifique e mostre qual a avaliação realizada nesta data, qual o horário e o grau de dificuldade, conforme tabela abaixo:

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Dados | | |
| Disciplina | horário | dificuldade |
| segunda-feira | Lógica de Programação | 19h30 as 21h | médio |
| terça-feira | Lógica Matemática | 21h30 as 23h | difícil |
| quarta-feira | Fundamentos Software | 19h30 as 21h | fácil |
| quinta-feira | Fundamentos Hardware | 21h30 as 23h | fácil |
| sexta-feira | Inglês | 19h30 as 21h | médio |

30. Desenvolva um algoritmo que recebe o nome de cinco alunos de sua sala, verifique e mostre qual a idade, o sexo e o estado civil de cada um deles.

# Capítulo 8 – Estrutura de Repetição

Quando desenvolvemos um algoritmo ou programa estruturado, muitas vezes, precisamos repetir parte do algoritmo, um comando ou um conjunto de comandos. Podemos fazer isso repetindo esse bloco quantas vezes forem necessárias, mas isso é inviável. Para solucionar esse problema, podemos usar as estruturas de repetição. No entanto, precisamos saber controlar quantas vezes o algoritmo deve repetir determinada parte do algoritmo. Para isso, estudaremos o fluxo de controle chamado estrutura de repetição.

Uma estrutura de repetição é um fluxo de controle utilizado para decidir quantas vezes determinado conjunto de comandos se repetirá dentro do algoritmo. Uma estrutura de repetição determina qual conjunto de comandos ou bloco será executado após uma condição ser avaliada. Essa condição é representada por expressões lógicas e relacionais que podem ou não ser satisfeitas, isto é, podem retornar o valor verdadeiro ou falso. Enquanto essa condição estiver retornando verdadeiro, o conjunto de comandos será executado, parando somente quando o resultado da avaliação da condição for falso. Numa estrutura de repetição, o número de repetições pode ser indeterminado, mas necessariamente finito.

As estruturas de repetição também podem estar encadeadas, ou seja, uma estrutura de repetição dentro de outra estrutura de repetição. Às vezes, precisamos repetir várias vezes uma mesma estrutura de repetição, é nesses casos que utilizamos as estruturas de repetições encadeadas.

Uma estrutura de repetição pode ser construída dentro de uma estrutura de decisão, seja ela simples, composta ou encadeada, bem como dentro de uma estrutura de múltipla escolha. Da mesma forma, qualquer estrutura de decisão ou de múltipla escolha pode ser construída dentro de uma estrutura de repetição.

Temos três tipos de estrutura de repetição: **enquanto**, **faça/enquanto** e **para**. Cada uma dessas estruturas de repetição pode ser simples ou encadeada. Veremos a seguir como e quando trabalhar com cada uma delas.

## 8.1 Estrutura de Repetição Enquanto

Uma estrutura de repetição **enquanto** pode ser utilizada quando o algoritmo precisa testar determinada condição antes de executar um conjunto de comandos repetidas vezes. Nesse caso, uma condição é avaliada e se seu resultado for verdadeiro, o conjunto de comandos dentro da estrutura de repetição **enquanto** é executado e, após essa execução, a condição é novamente avaliada. Por outro lado, se o resultado da avaliação for falso, esse conjunto de comandos não será executado e o fluxo do algoritmo segue normalmente. Note que nessa estrutura de repetição, pode ocorrer que o conjunto de comando não seja executado nenhuma vez.

### Pseudocódigo

Na estrutura de repetição **enquanto**, utilizaremos as palavras **enquanto** e **faça** que representam as palavras principais dessa estrutura e a palavra **fimenquanto;** para determinar o fim do bloco de execução da estrutura. A estrutura de repetição **enquanto** para pseudocódigo segue a seguinte regra sintática:

**<inicialização da variável de controle>;**

**enquanto (<condição>) faça**

**<comandos>;**

**<atualização da variável de controle>;**

**fimenquanto;**

Por exemplo:

x ← 0;

**enquanto** (x < 3) **faça**

escrever ("O valor de x é: " , x);

x ← x + 1;

**fimenquanto**;

Nesse exemplo, o valor inicial da variável **x** é 0. Na primeira vez que a condição **x<3** é avaliada, seu resultado é verdadeiro, pois 0 é menor que 3, então o comando **escrever(...);** e a atualização de **x** (**x** passa a ser 1) são executados. Na segunda vez que a condição **x<3** é avaliada, seu resultado é verdadeiro, pois 1 é menor que 3, então o comando **escrever(...);** e a atualização de **x** (**x** passa a ser 2) são executados. Na terceira vez que a condição **x<3** é avaliada, seu resultado é verdadeiro, pois 2 é menor que 3, então o comando **escrever(...);** e a atualização de **x** (**x** passa a ser 3) são executados. Na quarta vez que a condição **x<3** é avaliada, seu resultado é falso, pois 3 não é menor que 3, então os comandos dentro da estrutura de repetição **enquanto** não são executados e essa estrutura é finalizada, seguindo com o fluxo do algoritmo após esta estrutura.

O resultado da execução deste exemplo é:

O valor de x é 0

O valor de x é 1

O valor de x é 2

### Java

No Java, para a estrutura de repetição **enquanto**, utilizaremos a palavra **while** que representa a palavra principal dessa estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte dessa estrutura. A estrutura de repetição **enquanto** para Java segue a seguinte regra sintática:

**<inicialização da variável de controle>;**

**while (<condição>)**

**{**

**<comandos>;**

**<atualização da variável de controle>;**

**}**

Por exemplo:

x ← 0;

**while** (x < 3)

**{**

System.out.println ("O valor de x é: " + x);

x ← x + 1;

**}**

Nesse exemplo, o valor inicial da variável **x** é 0. Na primeira vez que a condição **x<3** é avaliada, seu resultado é verdadeiro, pois 0 é menor que 3, então o comando **System.out.println(...);** e a atualização de **x** (**x** passa a ser 1) são executados. Na segunda vez que a condição **x<3** é avaliada, seu resultado é verdadeiro, pois 1 é menor que 3, então o comando **System.out.println (...);** e a atualização de **x** (**x** passa a ser 2) são executados. Na terceira vez que a condição **x<3** é avaliada, seu resultado é verdadeiro, pois 2 é menor que 3, então o comando **System.out.println (...);** e a atualização de **x** (**x** passa a ser 3) são executados. Na quarta vez que a condição **x<3** é avaliada, seu resultado é falso, pois 3 não é menor que 3, então os comandos dentro da estrutura de repetição **while** não são executados e essa estrutura é finalizada, seguindo com o fluxo do algoritmo após essa estrutura.

O resultado da execução desse exemplo é:

O valor de x é 0

O valor de x é 1

O valor de x é 2

### Encadeamento

Na estrutura de repetição **enquanto**, utilizaremos as palavras **enquanto** e **faça** que representam as palavras principais dessa estrutura e a palavra **fimenquanto;** para determinar o fim do bloco de execução dessa estrutura. A estrutura de repetição **enquanto** encadeada para pseudocódigo segue a seguinte regra sintática:

**<inicialização da variável de controle 1>;**

**enquanto (<condição1>) faça**

**<inicialização da variável de controle 2>;**

**enquanto (<condição2>) faça**

**<comandos que podem ser outra estrutura de repetição enquanto>;**

**<atualização da variável de controle 2>;**

**fimenquanto;**

**<atualização da variável de controle 1>;**

**fimenquanto;**

No Java, para a estrutura de repetição **enquanto**, utilizaremos a palavra **while** que representa a palavra principal dessa estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte dessa estrutura. A estrutura de repetição **while** encadeada para Java segue a seguinte regra sintática:

**<inicialização da variável de controle 1>;**

**while (<condição1>)**

**{**

**<inicialização da variável de controle 2>;**

**while (<condição2>)**

**{**

**<comandos que podem ter outra estrutura de repetição while>;**

**<atualização da variável de controle 2>;**

**}**

**<atualização da variável de controle 1>;**

**}**

Para um melhor entendimento do encadeamento nas estruturas de repetição **enquanto**, resolveremos o seguinte exemplo:

Desenvolva um algoritmo que calcula o fatorial dos números inteiros de 5 até 15.

Nesse exemplo, precisamos usar uma estrutura de repetição **enquanto** para calcular o fatorial de um número e uma outra estrutura de repetição **enquanto** para passar por todos os números de 5 até 15. Note que a primeira estrutura de repetição está dentro da segunda estrutura de repetição, ou seja, elas estão encadeadas. A resolução desse exercício em pseudocódigo é a seguinte:

**Algoritmo ExemploEnquanto**

**início\_algoritmo**

**Declarar**

i, num, fat **numérico\_inteiro**;

// entrada de dados

num ← 5;

// processamento de dados

**enquanto** (num <= 15) **faça**

fat ← num;

i ← num – 1;

**enquanto** (i > 1) **faça**

fat ← fat \* i;

i ← i – 1;

**fimenquanto**;

// saída de dados

escrever ("O valor do fatorial de " , num , " é " , fat);

num ← num + 1;

**fimenquanto**;

**fim\_algoritmo.**

A solução desse exercício em Java é a seguinte:

**class ExemploEnquanto**

**{**

**public static void main(String args [])**

**{**

**int** i, num, fat;

num = 5;

**while** (num <= 15)

**{**

fat = num;

i = num – 1;

**while** (i > 1)

**{**

fat = fat \* i;

i = i – 1;

**} // segundo while**

System.out.print ("O valor do fatorial de " + num +   
 " é " + fat + "\n");

num = num + 1;

**} // primeiro while**

**} // main**

**} // classe**

## 8.2 Faça/Enquanto

Uma estrutura de repetição **faça/enquanto** pode ser utilizada quando o algoritmo precisa testar determinada condição depois de executar um conjunto de comandos, passando por esses passos repetidas vezes. Nesse caso, um conjunto de comandos dentro da estrutura de repetição **faça/enquanto** é executado e, após esta execução, uma condição é avaliada e, se seu resultado for verdadeiro, o conjunto de comandos é novamente executado. Por outro lado, se o resultado da avaliação for falso, esse conjunto de comandos não volta a ser executado e o fluxo do algoritmo segue normalmente. Note que nessa estrutura de repetição, pelo menos uma vez, o conjunto de comandos dentro da estrutura de repetição **faça/enquanto** é executado.

### Pseudocódigo

Na estrutura de repetição **faça/enquanto**, utilizaremos as palavras **faça** e **enquanto** que representam as palavras principais dessa estrutura e a própria palavra **enquanto** determinará o fim do bloco de execução da estrutura. A estrutura de repetição **faça/enquanto** para pseudocódigo segue a seguinte regra sintática:

**<inicialização da variável de controle>;**

**faça**

**<comandos>;**

**<atualização da variável de controle>;**

**enquanto(<condição>);**

Por exemplo:

x ← 0;

**faça**

escrever ("O valor de x é: " , x);

x ← x + 1;

**enquanto** (x < 3);

Neste exemplo, o valor inicial da variável **x** é 0. Na primeira vez que a estrutura de repetição **faça/enquanto** é solicitada, o comando **escrever(...);** e a atualização de **x** (**x** passa a ser 1) são executados e a condição **x<3** é avaliada. O resultado dessa avaliação é verdadeiro, pois 1 é menor que 3, então a estrutura de repetição **faça/enquanto** é novamente solicitada, o comando **escrever(...);** e a atualização de **x** (**x** passa a ser 2) são executados e a condição **x<3** é avaliada. O resultado dessa avaliação é verdadeiro, pois 2 é menor que 3, então a estrutura de repetição **faça/enquanto** é novamente solicitada, o comando **escrever(...);** e a atualização de **x** (**x** passa a ser 3) são executados e a condição **x<3** é avaliada. O resultado dessa avaliação é falso, pois 3 não é menor que 3, então essa estrutura é finalizada, seguindo com o fluxo do algoritmo após essa estrutura.

Resultado da execução deste exemplo:

O valor de x é 0

O valor de x é 1

O valor de x é 2

### Java

No Java, para a estrutura de repetição **faça/enquanto**, utilizaremos as palavras **do** e **while** que representam as palavras principais dessa estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte dessa estrutura. A estrutura de repetição **faça/enquanto** para Java segue a seguinte regra sintática:

**<inicialização da variável de controle>;**

**do**

**{**

**<comandos>;**

**<atualização da variável de controle>;**

**} while (<condição>);**

Por exemplo:

x = 0;

**do**

**{**

System.out.println ("O valor de x é: " + x);

x = x + 1;

**} while** (x < 3);

Nesse exemplo, o valor inicial da variável **x** é 0. Na primeira vez que a estrutura de repetição **do/while** é solicitada, o comando **System.out.println(...);** e a atualização de **x** (**x** passa a ser 1) são executados e a condição **x<3** é avaliada. O resultado dessa avaliação é verdadeiro, pois 1 é menor que 3, então a estrutura de repetição **do/while** é novamente solicitada, o comando **System.out.println (...);** e a atualização de **x** (**x** passa a ser 2) são executados e a condição **x<3** é avaliada. O resultado dessa avaliação é verdadeiro, pois 2 é menor que 3, então a estrutura de repetição **do/while** é novamente solicitada, o comando **System.out.println (...);** e a atualização de **x** (**x** passa a ser 3) são executados e a condição **x<3** é avaliado. O resultado dessa avaliação é falso, pois 3 não é menor que 3, então essa estrutura é finalizada, seguindo com o fluxo do algoritmo após essa estrutura.

Resultado da execução desse exemplo:

O valor de x é 0

O valor de x é 1

O valor de x é 2

### Encadeamento

Na estrutura de repetição **faça/enquanto**, utilizaremos as palavras **faça** e **enquanto** que representam as palavras principais dessa estrutura e a própria palavra **enquanto** determinará o fim do bloco de execução dessa estrutura. A estrutura de repetição **faça/enquanto** encadeada para pseudocódigo segue a seguinte regra sintática:

**<inicialização da variável de controle 1>;**

**faça**

**<inicialização da variável de controle 2>;**

**faça**

**<comandos que podem ter outra estrutura de repetição faça/enquanto>;**

**<atualização da variável de controle 2>;**

**enquanto(<condição1>);**

**<atualização da variável de controle 1>;**

**enquanto(<condição2>);**

No Java, para a estrutura de repetição **faça/enquanto**, utilizaremos as palavras **do** e **while** que representam as palavras principais dessa estrutura e chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte dessa estrutura. A estrutura de repetição **faça/enquanto** encadeado para Java segue a seguinte regra sintática:

**<inicialização da variável de controle 1>;**

**do**

**{**

**<inicialização da variável de controle 2>;**

**do**

**{**

**<comandos que podem ter outra estrutura de repetição do/while>;**

**<atualização da variável de controle 2>;**

**} while (<condição1>);**

**<atualização da variável de controle 2>;**

**} while (<condição2>);**

Para um melhor entendimento do encadeamento nas estruturas de repetição **faça/enquanto**, resolveremos o seguinte exemplo:

Desenvolva um algoritmo que calcula o fatorial dos números inteiros de 5 até 15.

Nesse exemplo, precisamos usar uma estrutura de repetição **faça/enquanto** para calcular o fatorial de um número e uma outra estrutura de repetição **faça/enquanto** para passar por todos os números de 5 até 15. Note que a primeira estrutura de repetição está dentro da segunda estrutura de repetição, ou seja, elas estão encadeadas. A resolução desse exercício em pseudocódigo é a seguinte:

**Algoritmo ExemploFacaEnquanto**

**início\_algoritmo**

**Declarar**

i, num, fat **numérico\_inteiro**;

num ← 5; // entrada de dados

// processamento de dados

**faça**

fat ← num;

i ← num – 1;

**faça**

fat ← fat \* i;

i ← i – 1;

**enquanto** (i > 1);

escrever ("O valor do fatorial de " , num , " é " , fat); // saída de dados

num ← num + 1;

**enquanto** (num <= 15);

**fim\_algoritmo.**

A solução desse exercício em Java é a seguinte:

**class ExemploFacaEnquanto**

**{**

**public static void main(String args [])**

**{**

**int** i, num, fat;

num = 5;

**do**

**{**

fat = num;

i = num – 1;

**do**

**{**

fat = fat \* i;

i = i – 1;

**} while** (i > 1);

System.out.print ("O valor do fatorial de " + num +   
 " é " + fat);

num = num + 1;

**} while** (num <= 15);

**} // main**

**} // classe**

## 8.3 Para

Uma estrutura de repetição **para** pode ser utilizada quando o algoritmo precisa ter definida a quantidade de vezes que um conjunto de comandos deve ser executado. Nesse caso, a variável de controle, sua inicialização e finalização, bem como sua atualização, fazem parte do cabeçalho da estrutura de repetição **para** e o conjunto de comandos dentro da estrutura de repetição **para** é executado a quantidade de vezes determinada no cabeçalho dessa estrutura. Assim que o número de repetições dessa estrutura for finalizado, o fluxo do algoritmo segue normalmente. Note que nessa estrutura de repetição, pode ocorrer que o conjunto de comandos não seja executado nenhuma vez.

### Pseudocódigo

Na estrutura de repetição **para**, utilizaremos as palavras **para**, **de** , **até**, **passo** e **faça** que representam as palavras principais dessa estrutura e a palavra **fimpara;** para determinar o fim do bloco de execução dessa estrutura. A estrutura de repetição **para** para pseudocódigo segue a seguinte regra sintática:

**para <variável de controle> de <valor inicial> até <valor final> passo <atualização> faça**

**<comandos>;**

**fimpara;**

Por exemplo:

**para** x **de** 0 **até** 2 **passo** +1 **faça**

escrever ("O valor de x é: " , x);

**fimpara**;

Nesse exemplo, na primeira vez que a estrutura de repetição do **para** for solicitada, o valor inicial da variável **x** é 0, então o comando **escrever(...);** e a atualização de **x** (**x** passa a ser 1) são executados. Na segunda vez que a estrutura de repetição do **para** for solicitado, o valor de **x** é 1, então o comando **escrever(...);** e a atualização de **x** (**x** passa a ser 2) são executados. Na terceira vez que a estrutura de repetição do **para** for solicitado, o valor de **x** é 2, então o comando **escrever(...);** e a atualização de **x** (**x** passa a ser 3) são executados. Na quarta vez que a estrutura de repetição do **para** for solicitado, o valor de **x** é 3, então os comandos dentro da estrutura de repetição **para** não são executados e essa estrutura é finalizada, seguindo com o fluxo do algoritmo após essa estrutura.

O resultado da execução desse exemplo é:

O valor de x é 0

O valor de x é 1

O valor de x é 2

### Java

No Java, para a estrutura de repetição **para**, utilizaremos a palavra **for** que representa a palavra principal dessa estrutura e a chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte dessa estrutura. A estrutura de repetição **para** para Java segue a seguinte regra sintática:

**for (<var de controle> = <vlr inicial> ; <condição> ; <atualização>)**

**{**

**<comandos>;**

**}**

Por exemplo:

**for** (x=0 **;** x<3 **;** x++)

**{**

System.out.println ("O valor de x é: " + x);

**}**

Nesse exemplo, na primeira vez que a estrutura de repetição do **for** for solicitada, o valor inicial da variável **x** é 0, então o comando **System.out.println(...);** e a atualização de **x** (**x** passa a ser 1) são executados. Na segunda vez que a estrutura de repetição do **for** for solicitada, o valor de **x** é 1, então o comando **System.out.println (...);** e a atualização de **x** (**x** passa a ser 2) são executados. Na terceira vez que a estrutura de repetição do **for** for solicitada, o valor de **x** é 2, então o comando **System.out.println (...);** e a atualização de **x** (**x** passa a ser 3) são executados. Na quarta vez que a estrutura de repetição do **for** for solicitada, o valor de **x** é 3, então os comandos dentro da estrutura de repetição **for** não são executados e essa estrutura é finalizada, seguindo com o fluxo do algoritmo após esta estrutura.

O resultado da execução desse exemplo é:

O valor de x é 0

O valor de x é 1

O valor de x é 2

### Encadeamento

Na estrutura de repetição **para**, utilizaremos as palavras **para**, **de**, **até**, **passo** e **faça** que representam as palavras principais dessa estrutura e a palavra **fimpara;** para determinar o fim do bloco de execução dessa estrutura. A estrutura de repetição **para** encadeado para pseudocódigo segue a seguinte regra sintática:

**para <var de controle 1> de <valor inicial 1> até <vlr final> passo <atualização 1> faça**

**para <var de controle 2> de <vlr inicial 2> até <vlr final> passo <atualização 2> faça**

**<comandos que podem ter outra estrutura de repetição para>;**

**fimpara;**

**fimpara;**

No Java, para a estrutura de repetição **para**, utilizaremos a palavra **for** que representa a palavra principal dessa estrutura e a chave aberta, **{**, e chave fechada, **}**, para representar o conjunto de comandos que fazem parte dessa estrutura. A estrutura de repetição **para** encadeado para Java segue a seguinte regra sintática:

**for (<var de controle 1> = <vlr inicial 1> ; <condição 1> ; <atualização 1>)**

**{**

**for (<var de controle 2> = <vlr inicial 2> ; <condição 2> ; <atualização 2>)**

**{**

**<comandos que podem ter outra estrutura de repetição for>;**

**}**

**}**

Para um melhor entendimento do encadeamento nas estruturas de repetição **para**, resolveremos o seguinte exemplo:

Desenvolva um algoritmo que calcule o fatorial dos números inteiros de 5 até 15.

Nesse exemplo, precisamos usar uma estrutura de repetição **para** para calcular o fatorial de um número e uma outra estrutura de repetição **para** para passar por todos os números de 5 até 15. Note que a primeira estrutura de repetição está dentro da segunda estrutura de repetição, ou seja, elas estão encadeadas. A resolução deste exercício em pseudocódigo é a seguinte:

**Algoritmo ExemploPara**

**início\_algoritmo**

**Declarar**

i, num, fat **numérico\_inteiro**;

// processamento de dados

**para** num **de** 5 **até** 15 **passo** +1 **faça**

fat ← num;

**para** i **de** (num – 1) **até** 2 **passo** –1 **faça**

fat ← fat \* i;

**fimpara**;

escrever ("O valor do fatorial de " , num , " é " , fat); // saída de dados

**fimpara**;

**fim\_algoritmo.**

A solução desse exercício em Java é a seguinte:

**class ExemploPara**

**{**

**public static void main(String args [])**

**{**

**int** i, num, fat;

**for** (num = 5 ; num <= 15 ; num++)

**{**

fat = num;

**for** (i = (num – 1) ; i > 1 ; i––)

**{**

fat = fat \* i;

**} // segundo for**

System.out.print ("O valor do fatorial de " + num +   
 " é " + fat);

**} // primeiro for**

**} // main**

**} // classe**

## 8.4 Exemplo de Estrutura de Repetição Enquanto, Faça/Eqto. e Para em pseudocódigo

1. Desenvolva um algoritmo que calcula a soma de todos os números compreendidos entre 1 e 100.

// na estrutura de repetição **enquanto**

**Algoritmo Soma**

**Declarar**

i, s ← 0 **numérico\_inteiro**;

i ← 1;

**enquanto** (i <= 100) **faça**

s ← s + i;

i ← i + 1;

**fimenquanto;**

escreva ("A somatória de 1 até 100 é " , s);

**fim\_algoritmo.**

// na estrutura de repetição **faça/enquanto**

**Algoritmo Soma**

**Declarar**

i, s ← 0 **numérico\_inteiro**;

i ← 1;

**faça**

s ← s + i;

i ← i + 1;

**enquanto** (i <= 100)**;**

escreva ("A somatória de 1 até 100 é " , s);

**fim\_algoritmo.**

// na estrutura de repetição **para**

**Algoritmo Soma**

**Declarar**

i, s ← 0 **numérico\_inteiro**;

**para** i **de** 1 **até** 100 **passo** +1 **faça**

s ← s + i;

**fimpara;**

escreva ("A somatória de 1 até 100 é " , s);

**fim\_algoritmo.**

2. Desenvolva um algoritmo que calcula a soma de todos os números pares compreendidos entre 1 e 100.

// na estrutura de repetição **enquanto**

**Algoritmo Soma**

**Declarar**

i, s ← 0 **numérico\_inteiro**;

i ← 1;

**enquanto** (i <= 100) **faça**

**se** ((i mod 2) = 0)

**então**

s ← s + i;

**fimse**;

i ← i + 1;

**fimenquanto;**

escreva ("A somatória dos números pares de 1 até 100 é " , s);

**fim\_algoritmo.**

// na estrutura de repetição **faça/enquanto**

**Algoritmo Soma**

**Declarar**

i, s ← 0 **numérico\_inteiro**;

i ← 1;

**faça**

**se** ((i mod 2) = 0)

**então**

s ← s + i;

**fimse**;

i ← i + 1;

**enquanto** (i <= 100)**;**

escreva ("A somatória dos números pares de 1 até 100 é " , s);

**fim\_algoritmo.**

// na estrutura de repetição **para**

**Algoritmo Soma**

**Declarar**

i, s ← 0 **numérico\_inteiro**;

**para** i **de** 1 **até** 100 **passo** +1 **faça**

**se** ((i mod 2) = 0)

**então**

s ← s + i;

**fimse**;

**fimpara;**

escreva ("A somatória dos números pares de 1 até 100 é " , s);

**fim\_algoritmo.**

3. Desenvolva um algoritmo que mostra um menu de opções para: calcular a soma de todos os números compreendidos entre 1 e 100; calcular a soma de todos os números pares compreendidos entre 1 e 100; e calcular a soma de todos os números divisíveis por 3.

**Algoritmo Soma**

**Declarar**

op, i, s ← 0 **numérico\_inteiro**;

**faça**

escreva ("Digite 1 para soma de todos os números entre 1 e 100\n");

escreva ("Digite 2 para soma de todos os números pares entre 1 e 100\n");

escreva ("Digite 3 para soma de todos os números divisíveis por 3 entre 1 e 100\n");

escreva ("Digite 0 para sair do programa\n");

leia (op);

**escolha** (op)

**caso** 1 : **início**

i ← 1;

**enquanto** (i <= 100) **faça**

s ← s + i;

i ← i + 1;

**fimenquanto;**

escreva ("A somatória de 1 até 100 é " , s);

**fim;**

**caso** 2 : **início**

i ← 1;

**faça**

**se** ((i mod 2) = 0)

**então**

s ← s + i;

**fimse**;

i ← i + 1;

**enquanto** (i <= 100)**;**

escreva ("A somatória dos números pares de 1 até 100 é " , s);

**fim;**

**caso** 3 : **início**

**para** i **de** 1 **até** 100 **passo** +1 **faça**

**se** ((i mod 3) = 0)

**então**

s ← s + i;

**fimse**;

**fimpara;**

escreva ("A somatória dos números divisíveis por 3 de 1 até 100 é " , s);

**fim;**

**caso** 0 : **início**

escreva ("Saindo do programa!");

**fim;**

**caso contrário** : escreva ("opção inválida, tente novamente");

**fimescolha**;

**enquanto** (op <> 0);

**fim\_algoritmo.**

## 8.5 Exemplo de Estrutura de Repetição Enquanto, Faça/Enquanto e Para em Java

1. Desenvolva um algoritmo que calcula a soma de todos os números compreendidos entre 1 e 100.

// na estrutura de repetição **enquanto**

**class Soma**

**{**

**public static void main (String args [ ])**

**{**

**int** i, s = 0;

i = 1;

**while** (i <= 100)

**{**

s = s + i;

i = i + 1;

**}** // fim do while

System.out.println ("A somatória de 1 até 100 é " + s);

System.exit(0);

**}** // fim do void main

**}** // fim da classe

// na estrutura de repetição **faça/enquanto**

**class Soma**

**{**

**public static void main (String args [ ])**

**{**

**int** i, s = 0;

i = 1;

**do**

**{**

s = s + i;

i = i + 1;

**} while** (i <= 100)**;**

System.out.println ("A somatória de 1 até 100 é " + s);

System.exit(0);

**}** // fim do void main

**}** // fim da classe

// na estrutura de repetição **para**

**class Soma**

**{**

**public static void main (String args [ ])**

**{**

**int** i, s = 0;

**for (** i = 1 **;** i <= 100 **;** i++ **)**

**{**

s = s + i;

**}**

System.out.println ("A somatória de 1 até 100 é " + s);

System.exit(0);

**}** // fim do void main

**}** // fim da classe

2. Desenvolva um algoritmo que calcula a soma de todos os números pares compreendidos entre 1 e 100.

// na estrutura de repetição **enquanto**

**class Soma**

**{**

**public static void main (String args [ ])**

**{**

**int** i, s = 0;

i = 1;

**while** (i <= 100)

**{**

**if** ((i % 2) == 0)

**{**

s = s + i;

**}** // fim do if

i = i + 1;

**}** // fim do while

System.out.println ("A somatória dos números pares de 1 até 100 é " + s);

System.exit(0);

**}** // fim do void main

**}** // fim da classe

// na estrutura de repetição **faça/enquanto**

**class Soma**

**{**

**public static void main (String args [ ])**

**{**

**int** i, s = 0;

i = 1;

**do**

**{**

**if** ((i % 2) == 0)

**{**

s = s + i;

**}** // fim do se

i = i + 1;

**} while** (i <= 100)**;**

System.out.println ("A somatória dos números pares de 1 até 100 é " + s);

System.exit(0);

**}** // fim do void main

**}** // fim da classe

// na estrutura de repetição **para**

**class Soma**

**{**

**public static void main (String args [ ])**

**{**

**int** i, s = 0;

**for (** i = 1 **;** i <= 100 **;** i++ **)**

**{**

**if** ((i % 2) == 0)

**{**

s = s + i;

**}** // fim do if

**}** // fim do for

System.out.println ("A somatória dos números pares de 1 até 100 é " + s);

System.exit(0);

**}** // fim do void main

**}** // fim da classe

3. Desenvolva um algoritmo que mostra um menu de opções para: calcular a soma de todos os números compreendidos entre 1 e 100; calcular a soma de todos os números pares compreendidos entre 1 e 100; e calcular a soma de todos os números divisíveis por 3.

**class Soma**

**{**

**public static void main (String args [ ])**

**{**

**int** op, i, s;

**do**

**{**

s = 0;

op = Integer.parseInt(JOptionPane.showInputDialog("Digite 1 para soma de todos os números entre 1 e 100 \nDigite 2 para soma de todos os números pares entre 1 e 100 \nDigite 3 para soma de todos os números divisíveis por 3 entre 1 e 100 \nDigite 0 para sair do programa"));

**switch** (op)

**{**

**case** 1 : **{**

i = 1;

**while** (i <= 100)

**{**

s = s + i;

i = i + 1;

**}** // fim do while

System.out.println("A somatória de 1 até 100 é " + s);

**}**

break;

**case** 2 : **{**

i = 1;

**do**

**{**

**if** ((i % 2) == 0)

**{**

s = s + i;

**}** // fim do if

i = i + 1;

**} while** (i <= 100)**;**

System.out.println ("A somatória dos números pares de 1 até 100   
 é " + s);

**}**

break;

**case** 3 : **{**

**for (** i = 1 **;** i <= 100 **;** i++ **)**

**{**

**if** ((i % 3) == 0)

**{**

s = s + i;

**}** // fim do if

**}** // fim do for

System.out.println ("A somatória dos números divisíveis por 3 de 1   
 até 100 é " + s);

**}**

break;

**case** 0 : **{**

System.out.println("Saindo do programa!");

System.exit(0);

**}**

break;

**default** : System.out.println ("opção inválida, tente novamente");

**}** // fim do switch

**}** **while** (op != 0);

**}** // fim do void main

**}** // fim da classe

## 8.6 Exercícios de Estrutura de Repetição Enquanto, Faça/Enquanto e Para

Para todos os exercícios abaixo, construir um algoritmo para cada estrutura de repetição, ou seja, um usando o **Enquanto**, um usando o **Faça/Enquanto** e outro usando o **Para**.

1. Desenvolva um algoritmo que calcule e mostre o quadrado dos números inteiros compreendidos entre 10 e 150.

2. Desenvolva um algoritmo que receba um número inteiro, calcule e mostre o seu fatorial.

3. Desenvolva um algoritmo que receba um número **N**, calcule e mostre o valor da seguinte série:

Série = 1 + 1/2 + 1/3 + ... + 1/N.

4. Desenvolva um algoritmo que receba um número, calcule e mostre os resultados da tabuada desse número.

5. Desenvolva um algoritmo que receba dois números inteiros, verifique qual é o maior entre eles, calcule e mostre o resultado da somatória dos números ímpares compreendidos entre esses dois números.

6. Desenvolva um algoritmo que receba um número **N**, calcule e mostre o valor da seguinte série:

Série = 1 + 1/1! + 1/2! + 1/3! + ... + 1/N!.

7. Desenvolva um algoritmo que receba um número inteiro **N**, calcule e mostre a série de Fibonacci até o seu **N**-ésimo termo. A série de Fibonacci é dada pela seguinte seqüência: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ... etc.

8. Desenvolva um algoritmo que receba 100 números reais, verifique e mostre o maior e o menor número recebido.

9. Desenvolva um algoritmo que calcule o número de grãos de trigo dispostos num tabuleiro de xadrez e que segue a seguinte regra: no primeiro quadro, colocar um grão de trigo; no segundo quadro, colocar o dobro de grãos de trigo do primeiro quadro; e, para todos os quadros subseqüentes, colocar o dobro de grãos de trigo do quadro anterior. Um tabuleiro de xadrez tem 64 quadros.

10. Desenvolva um algoritmo que receba um número **x** calcule e mostre um número inteiro que mais se aproxima da raiz quadrada desse número **x**.

11. Desenvolva um algoritmo que receba um número inteiro, verifique e mostre se esse número é primo ou não.

12. Desenvolva um algoritmo que receba dois números inteiros, verifique e mostre todos os números primos existentes entre os dois números recebidos.

13. Desenvolva um algoritmo que mostre todas as possibilidades de lançamento de dois dados, de forma que tenhamos o valor 7 como resultado da soma dos valores de cada dado.

14. Desenvolva um algoritmo que calcule e mostre o valor da seguinte série:

Série = 1 + 2/3 + 3/5 + 4/7 + ... + 50/99.

15. Desenvolva um algoritmo que receba 15 números inteiros, calcule e mostre a somatória do fatorial de cada número recebido.

16. Desenvolva um algoritmo que recebe 30 números reais, calcula e mostra a somatória e a média dos números recebidos.

17. Desenvolva um algoritmo que receba dois números inteiros, verifique qual o maior entre eles, calcule e mostre o quociente e o resto da divisão do maior número pelo menor. Em hipótese nenhuma utilizar os operadores **div** e **mod**.

18. Desenvolva um algoritmo que receba o nome e a idade de 50 pessoas, verifique e mostre o nome e a idade da pessoa mais velha e da pessoa mais nova.

19. Desenvolva um algoritmo que calcule e mostre quantos anos serão necessários para que Florentina seja maior que Clarisbela, considerando que Clarisbela tem 1,50 metros e cresce 2 centímetros por ano e Florentina tem 1,10 metros e cresce 3 centímetros por ano.

20. Desenvolva um algoritmo que calcule e mostre o valor da somatória dos trinta primeiros termos da seguinte série:

Série = 5/1000 – 10/980 + 5/960 – 10/940 + ...

21. Desenvolva um algoritmo que receba o salário de Clarisbela, calcule e mostre o salário de Florentina que é um terço do salário de Clarisbela. Além disso, calcule e mostre quantos meses são necessários para que o dinheiro aplicado de Florentina seja igual ou maior que o dinheiro aplicado de Clarisbela, considerando que Clarisbela aplicou todo o seu salário na poupança, que rende 2% ao mês, e que Florentina aplicou todo o seu salário no fundo de renda fixa, que rende 5% ao mês.

22. Desenvolva um algoritmo que receba dois valores inteiros, **x** e **y**, calcule e mostre a potência **xy**, sem utilizar o operador **pot**.

23. Desenvolva um algoritmo que receba o sexo, a idade e a experiência no trabalho (s/n) de 50 candidatos a uma vaga de uma empresa, calcule e mostre: o número de candidatos do sexo masculino; o número de candidatos do sexo feminino; a idade média dos candidatos que já têm experiência no trabalho; a porcentagem de homens com mais de 50 anos do total de homens; o número de mulheres com idade inferior a 30 e com experiência no trabalho; e a menor e a maior idade entre as mulheres que já têm experiência no trabalho.

24. Desenvolva um algoritmo que receba 50 números reais, calcule e mostre: a soma dos números digitados; a média dos números digitados; o maior número digitado; o menor número digitado; o dobro dos números digitados; o cubo dos números digitados; a porcentagem dos números ímpares digitados.

25. Desenvolva um algoritmo que calcule e mostre o valor da seguinte série:

Série = 1/2 – 2/4 + 3/6 – 4/8 + 5/10 – 6/12 + ... – 50/100.

26. Desenvolva um algoritmo que receba um número inteiro **N**, mostre um menu de opções: calcule e mostre a soma do fatorial dos números entre 1 e **N**; calcule e mostre a série de Fibonacci até o seu **N**-ésimo termo; calcule e mostre o produto dos números entre 1 e **N**; e calcule e mostre a soma dos números pares entre 1 e **N**.

27. Desenvolva um algoritmo que mostre um menu de opções: receba 20 números reais, calcule e mostre a somatória desses números; receba 15 números reais, calcule e mostre o produto dos números ímpares desses números; e receba 10 números inteiros, calcule e mostre quais desses números são divisíveis por 5 e 7.

28. Desenvolva um algoritmo que mostre um menu de opções: receba a altura e a base de 10 triângulos, calcule e mostre a área de cada triângulo; receba a altura e a base de 15 retângulos, calcule e mostre a área de cada retângulo; receba a altura, a base maior e a base menor de 20 trapézios, calcule e mostre a área de cada trapézio; e receba o raio de 15 circunferências, calcule e mostre a área de cada circunferência.

29. Desenvolva um algoritmo que receba dois valores inteiros, **a** e **b**, **a** > **b**, mostre um menu de opções que: calcule e mostre a potência **ab**, sem utilizar o operador **pot**; calcule e mostre o quociente e o resto da divisão de **a** por **b**, sem usar os operadores **div** e **mod**; e calcule e mostre se os valores **a** e **b** são primos ou não.

30. Desenvolva um algoritmo que calcule e mostre o valor da seguinte série:

Série = 1/1 – 2/4 + 3/9 – 4/16 + 5/25 – ... + 15/225.

1. Ferreira, Aurélio Buarque de Holanda. Novo Aurélio século XXI: O Dicionário da Língua Portuguesa. 3a edição. Nova Fronteira. Rio de Janeiro. 1999. [↑](#footnote-ref-1)
2. O conceito de vetor será visto no capítulo 10. [↑](#footnote-ref-2)