

Introdução aos Algoritmos e à Programação Básica com a Linguagem Java

Bruno Ferreira



Formação Inicial e
Continuada

+ IFMG

Campus Formiga



Bruno Ferreira

Introdução aos Algoritmos e à Programação Básica com a Linguagem Java

1ª Edição

Belo Horizonte

Instituto Federal de Minas Gerais

2022

© 2022 by Instituto Federal de Minas Gerais

Todos os direitos autorais reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico. Incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização por escrito do Instituto Federal de Minas Gerais.

Pró-reitor de Extensão	Carlos Bernardes Rosa Júnior
Diretor de Programas de Extensão	Niltom Vieira Junior
Coordenação do curso	Bruno Ferreira
Arte gráfica	Ângela Bacon
Diagramação	Eduardo dos Santos Oliveira

FICHA CATALOGRÁFICA

Dados Internacionais de Catalogação na Publicação (CIP)

F383i Ferreira, Bruno.

Introdução aos algoritmos e à programação básica com a linguagem Java / Bruno Ferreira – Belo Horizonte: Instituto Federal de Minas Gerais, 2022.

63 p. : il. color.

E-book, no formato PDF.

Material didático para Formação Inicial e Continuada.

ISBN 978-65-5876-045-0

1. Algoritmo. 2. Programação. 3. Lógica de programação. I. Ferreira, Bruno. II. Título.

...

CDD 005.1

Catalogação: Simoni Júlia da Silveira - CRB-6/2396

Índice para catálogo sistemático:

Linguagens de programação: Processamento de dados:005.13

2022

Direitos exclusivos cedidos ao
Instituto Federal de Minas Gerais
Avenida Mário Werneck, 2590,
CEP: 30575-180, Buritis, Belo Horizonte – MG,
Telefone: (31) 2513-5157

Sobre o material

Este curso é autoexplicativo e não possui tutoria. O material didático, incluindo suas videoaulas, foi projetado para que você consiga evoluir de forma autônoma e suficiente.

Caso opte por imprimir este *e-book*, você não perderá a possibilidade de acessar os materiais multimídia e complementares. Os *links* podem ser acessados usando o seu celular, por meio do glossário de Códigos QR disponível no fim deste livro.

Embora o material passe por revisão, somos gratos em receber suas sugestões para possíveis correções (erros ortográficos, conceituais, *links* inativos etc.). A sua participação é muito importante para a nossa constante melhoria. Acesse, a qualquer momento, o Formulário “Sugestões para Correção do Material Didático” clicando nesse [link](#) ou acessando o QR Code a seguir:



Formulário de
Sugestões

Para saber mais sobre a Plataforma +IFMG acesse

<http://mais.ifmg.edu.br>



Palavra(s) do(s) autor(es)

Seja bem-vindo ao curso de Introdução aos Algoritmos e à Programação Básica com a Linguagem Java.

A área de desenvolvimento de sistemas é extremamente atrativa, seja pelo grande número de vagas de empregos disponíveis ou pelos ótimos salários ofertados por esse campo de trabalho. No entanto, os profissionais dessa área têm que dominar os fundamentos ou a lógica de programação para ingressar nessa carreira.

Assim, esse curso começa apresentando os conceitos básicos sobre algoritmos e como eles se relacionam com a arquitetura dos computadores.

Em seguida, vamos conhecer o conceito de variáveis, e os comandos básicos de entrada e saída de dados. Neste ponto também é apresentado os operadores que compõe as estruturas sequenciais de cálculos.

Por fim, vamos nos concentrar nas instruções que possibilitam mudar o fluxo de execução, isso em conformidade com certas condições e, vamos aprender a repetir instruções de acordo com as necessidades impostas para resolver uma determinada tarefa.

Bons estudos!

Bruno Ferreira.



Apresentação do curso

Este curso está dividido em quatro semanas, cujos objetivos de cada uma são apresentados, sucintamente, a seguir.

SEMANA 1	Conhecer os conceitos iniciais (algoritmos, programas, IDE, linguagens de programação), a linguagem Java, configuração do ambiente de programação, introdução aos comandos de entrada/saída e aos conceitos de variáveis.
SEMANA 2	Entender e praticar os operadores e comandos que compõem as instruções sequencias.
SEMANA 3	Entender e praticar as estruturas condicionais.
SEMANA 4	Entender e praticar as estruturas de repetição.

Carga horária: 40 horas.

Estudo proposto: 2h por dia em cinco dias por semana (10 horas semanais).



Apresentação dos Ícones

Os ícones são elementos gráficos para facilitar os estudos, fique atento quando eles aparecem no texto. Veja aqui o seu significado:



Atenção: indica pontos de maior importância no texto.



Dica do professor: novas informações ou curiosidades relacionadas ao tema em estudo.



Atividade: sugestão de tarefas e atividades para o desenvolvimento da aprendizagem.



Mídia digital: sugestão de recursos audiovisuais para enriquecer a aprendizagem.



Sumário

Semana 1 – Conceitos básicos	15
1.1. Considerações iniciais.....	15
1.2. Algoritmo.....	17
1.3. Programa de Computador.....	18
1.4. Linguagem de programação.....	19
1.5. Compilando um programa.....	21
1.6. Ambientes de Desenvolvimento Integrado	21
1.7. A linguagem <i>Java</i>	23
Semana 2 – Estrutura sequencial.....	25
2.1 Considerações iniciais.....	25
2.2 Variáveis	26
2.3 Comandos de exibição de dados	27
2.4 Comandos de entrada de dados	28
2.5 Operador de atribuição.....	29
2.6 Operadores aritméticos e acumuladores.....	30
2.7 Operadores de incremento e os relacionais	32
2.8 Operadores lógicos	34
Semana 3 – Estruturas condicionais	37
3.1 Conceitos iniciais	37
3.2 Comando condicional simples.....	38
3.3 Comando condicional composto	39
3.4 Comando <i>switch</i>	41
3.5 Operador condicional ternário	42
Semana 4 – Estruturas de repetição	45
4.1 Conceitos iniciais	45
4.2 Estrutura de repetição “For” (para).....	46
4.3 Estrutura de repetição “While” (enquanto).....	47
4.4 Estrutura de repetição “Do while” (repita enquanto)	48
4.5 Modificadores de fluxo de laço (break, continue).....	49

Referências	53
Currículo do autor.....	55
Glossário de códigos QR (<i>Quick Response</i>)	57



Objetivos

Nesta semana você irá conhecer os conceitos básicos relacionados aos algoritmos e ter o primeiro contato com a criação dos mesmos.



Mídia digital: Antes de iniciar os estudos, vá até a sala virtual e assista ao vídeo “Apresentação do curso”.

1.1. Considerações iniciais

Desde o início de sua existência, o homem procurou criar máquinas que o auxiliassem em seu trabalho, diminuindo o esforço e economizando tempo. Por exemplo, o Ábaco foi inventado há aproximadamente 3 mil anos atrás por comerciantes de algumas regiões da Ásia, o objetivo era facilitar as operações aritméticas básicas.



Figura 1 – Ábaco.

Fonte: <http://metodosupera.com.br/conheca-o-abaco/> (Acesso em: 01 fev. 2022).

Dentre as máquinas mais recentes, o computador vem se mostrando uma das mais versáteis, rápidas e seguras. A principal finalidade de um computador é receber, manipular e armazenar dados, hoje, de forma eletroeletrônica.

Os computadores estão inseridos no nosso cotidiano, seja através dos modernos celulares (*smartphones*), nos equipamentos que compõem os veículos, ou mesmos nas operações bancárias ou instrumentos cirúrgicos dos hospitais. Em comum, esses diferentes tipos de equipamentos seguem uma estrutura interna chamada de arquitetura de Von Neumann, a qual é um modelo de computador digital, ele utiliza uma unidade central de processamento (CPU) e uma unidade de armazenamento ("memória") para guardar instruções de execução e os dados.

A Figura 2 apresenta a ideia da arquitetura de Von Neumann, as instruções executadas pelo computador e os dados são canalizadas a partir da memória para a CPU. Depois do processamento, os resultados são reconduzidos para a memória afim de serem salvos.

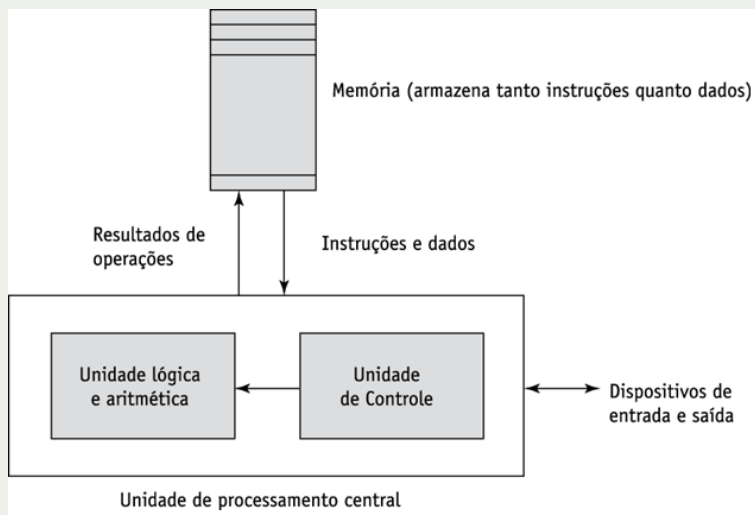


Figura 2 – Arquitetura de Von Neumann.
Fonte: (SEBESTA, 2011).

Essa arquitetura segue um ciclo básico de obtenção e execução de instruções que é mostrado no Quadro 1. No caso, o “contador de programa”, é um local ou informação que indica qual será a próxima instrução que será executada pelo computador. Assim, essa instrução é trazida da memória para a CPU e depois devolvida ao local de origem.

```

inicialize o contador de programa
repita para sempre
    obtenha a instrução apontada pelo contador de programa
    incremente o contador de programa
    decodifique a instrução
    execute a instrução
fim repita
  
```

Quadro 1 – Ciclo de obtenção e execução da arquitetura de Von Neumann.
Fonte: (SEBESTA, 2011).



Atenção: As instruções armazenadas na memória e apontadas pelo “contador de programa” torna possível comandar as máquinas construídas utilizando a arquitetura de Von Neumann.

Ao conjunto de instruções para resolver um determinado problema, dá-se o nome de **algoritmo**.

1.2. Algoritmo

O termo algoritmo não é novo e foi sendo formulado desde o surgimento do sistema de notação numérico hindu-arábico e da escrita dos cálculos algébricos pelo árabe Al-Kharazmi, contudo, foi a partir de 1250 d.c. que o matemático e astrônomo John Halifax apresentou conceitos mais concretos. Hoje esses conceitos e sua lógica são as bases para a Ciência da Computação (FILHO, 2007, p. 33).

Cormen et al. (2002, p. 22) e Manzano e Oliveira (1997) descrevem que um algoritmo é uma sequência finita de instruções bem definidas e não ambíguas. Cada uma dessas instruções deve ser executada num período de tempo específico e com uma quantidade de esforço limitada, visando um fim. Contudo, existem diversas variações dessa definição, mas para nosso curso essa descrição é suficiente.



Atenção: Algoritmo é a descrição de uma sequência de passos/instruções que devem ser seguidos para a realização de uma tarefa.

Veja abaixo dois exemplos de algoritmos cotidianos que executamos:

Algoritmo 1	— Somar três números
Passo 1	— Receber os três números.
Passo 2	— Somar os três números.
Passo 3	— Mostrar o resultado obtido.
Algoritmo 2	— Fazer um sanduíche
Passo 1	— Pegar o pão.
Passo 2	— Cortar o pão ao meio.
Passo 3	— Pegar a maionese.
Passo 4	— Passar a maionese no pão.
Passo 5	— Pegar e cortar alface e tomate.
Passo 6	— Colocar alface e tomate no pão.
Passo 7	— Pegar o hambúrguer.
Passo 8	— Fritar o hambúrguer.
Passo 9	— Colocar o hambúrguer no pão.

É importante ressaltar que a maioria dos algoritmos não são únicos, ou seja, existem diversas formas de executar ações afim de chegar em um mesmo resultado. Por exemplo, no Algoritmo 2 (citado acima), o primeiro passo poderia ser “Pegar a maionese”, assim, trocar essa ação não influenciaria no resultado final. Contudo, alguns passos devem ser prioritários, por exemplo, como você conseguiria fritar o hambúrguer sem antes pegá-lo da geladeira?



Atividade: Para exercitarmos um pouco sobre a criação de algoritmos, escreva ou digite os passos de um algoritmo para “trocar uma lâmpada” e para “ir à escola”. Obs.: essa atividade não requer entrega e não é avaliativa.

Para ajudar a organizar esses passos/ações durante a criação de um algoritmo, nós devemos pensar em três etapas básicas, as quais são mostrados na Figura 3.



Figura 3 – Etapas de implementação de um algoritmo.
Fonte: próprio autor.

A maioria dos problemas requerem uma **entrada de dados**, o **processamento** deles e finalmente a **apresentação ou armazenamento** dos resultados. A entrada pode ser, por exemplo, a data de nascimento de uma pessoa. Todo problema requerer algum tipo de processamento, por exemplo, o cálculo da idade da pessoa de acordo com o ano atual e, a saída seria a apresentação dessa idade na tela do computador.

1.3. Programa de Computador

Um programa de computador ou *software* (termo em inglês) é criado por um, ou mais algoritmos a serem usados direta ou indiretamente por um computador, o objetivo é apresentar um determinado resultado. Ele é composto por um código fonte, desenvolvido em alguma linguagem de programação.

As etapas para o desenvolvimento de um programa podem ser:

- **Análise:** estuda-se o enunciado do problema para definir os dados de entrada, o processamento e os dados de saída.
- **Algoritmo:** ferramentas do tipo descrição narrativa, fluxograma ou português estruturado são utilizadas para descrever o problema com suas soluções.

- **Codificação:** o algoritmo é transformado em códigos da linguagem de programação escolhida para se trabalhar.



Atenção: Um programa de computador (*software*) é a codificação de um ou mais algoritmos em uma linguagem de programação.

1.4. Linguagem de programação

Na Seção 1.2, foram apresentados os algoritmos para solucionar o problema de “somar dois números” e de “fazer um sanduíche”, nessa ocasião utilizamos a linguagem natural para descrever os passos/ações. Contudo, os computadores e sua arquitetura de Von Neumann têm uma linguagem própria, ou seja, a instrução apontada pelo “contador de programa” é uma instrução de máquina, esse tipo de instrução não é facilmente manipulado pelo homem, então para facilitar essa tarefa de comandar a máquina, foram criadas linguagens de mais alto nível de abstração, às quais são muito parecidas com as nossas linguagens naturais (português, inglês, espanhol, etc.).

Uma **linguagem de programação** é um método padronizado, formado por um conjunto de regras **sintáticas** e **semânticas** para descrever os algoritmos que compõem o programa de computador (SEBESTA, 2011). Toda linguagem formal tem sua sintaxe e semântica:

- **sintaxe** refere-se às regras que regem a composição de textos com significado em uma linguagem formal, tal como uma linguagem de programação;
- **semântica** é o significado, ela fornece uma interpretação ao texto escrito.

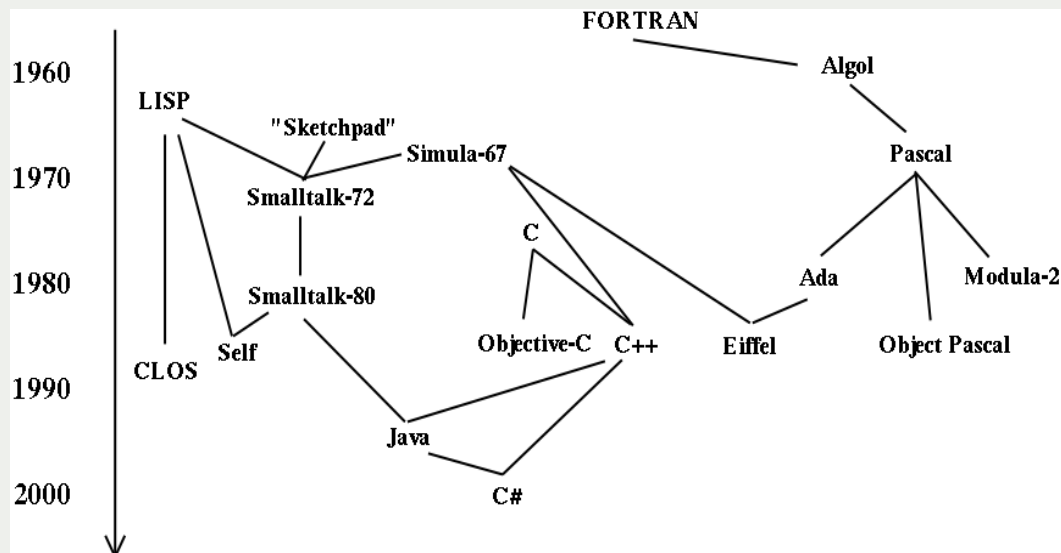
Uma linguagem de programação pode ser entendida também como um conjunto de símbolos e códigos usados para criar as estruturas de dados e o fluxo de execução do programa. Ou seja, um conjunto de palavras, organizadas de acordo com regras que ditam a gramática e o significado das frases.



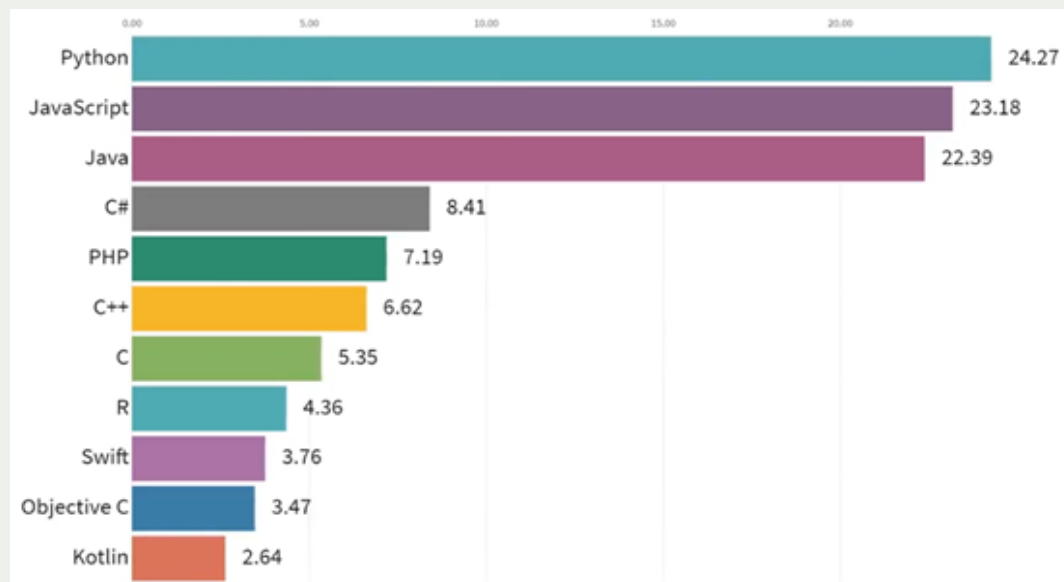
Atenção: Os algoritmos inscritos em uma linguagem de programação e usados para criar um programa de computador, também são chamados de código fonte do programa.

Assim como existem diversas linguagens faladas pelos diversos países existentes no planeta (português, inglês, etc.), nós podemos fazer uso de várias opções de linguagens de programação para escrever/digitar os algoritmos que compõem um programa de computador. A Figura 4 mostra dois tipos de gráficos. Na primeira imagem (A) é exibida uma linha do tempo, ela indica em qual ano uma linguagem foi criada e em quais outras ela foi inspirada. Por exemplo, em meados dos anos de 1990, a linguagem *Java* foi criada e ela foi inspirada nas linguagens chamadas *Smalltalk* e *C++*.

O segundo gráfico (B), mostra um levantamento das linguagens mais utilizadas globalmente no ano de 2019. Esses dados foram levantados pelo canal do Youtube chamado "Data Is Beautiful"¹ e leva em considerações as interações em sites de dúvidas e repositórios de código fonte. Nesse gráfico é fácil perceber que as linguagens mais populares atualmente são Python, Javascript e Java.



(A) Linha do tempo de criações das linguagens.
Fonte: (SEBESTA, 2011)



(B) Popularidade das linguagens por ano.
Fonte: Canal Data Is Beautiful¹.
Figura 4 – Evolução das linguagens.

¹ Canal do Youtube chamado “Data Is Beaultful”: <http://www.youtube.com/watch?v=8anOprjALhs> 21/02/2022 acessado em 21/02/2022.

1.5. Compilando um programa

Como explicado na Seção 1.4, os programas de computadores são escritos através de um ou mais algoritmos e esses algoritmos são escritos em uma linguagem de programação de alto nível, como por exemplo, o *Java*. Contudo, os computadores entendem somente uma linguagem de mais baixo nível, chamada também de linguagem de máquina. Neste ponto, um leitor atento já se perguntou: Como é feita a transformação entre essa linguagem de alto nível para a de baixo nível? Será que essa ação é complicada?

A boa notícia é que apensar de ser uma tarefa que demanda um bom processamento de máquina, essa tradução do alto nível de abstração para o baixo nível é feita de forma automática e transparente para o programador. Isso é feito através de um processo chamado de **compilação**² e é executado por um software chamado **compilador**.

O compilador é um software capaz de traduzir o código fonte de um programa, escrito em uma linguagem de alto nível, para uma espécie de programa equivalente, escrito em outra linguagem, que seja semanticamente equivalente, mas capaz de ser lida pelos processadores (SEBESTA, 2011).

Cada linguagem de programação tem seu próprio compilador, ou seja, para trabalharmos com a linguagem Java, vamos ter que instalar no nosso computador, o compilador referente a ela. Nesse processo de tradução, o código fonte passa por diversas análises para verificar, por exemplo, se a sintaxe e/ou a semântica do seu algoritmo obedeceu a regras da linguagem. Mas geralmente não instalamos somente o compilador, na maioria das vezes instalamos softwares para compor um ambiente de desenvolvimento de programas. Contudo, a boa notícia é que existem softwares que tem várias dessas funcionalidades já integradas em um mesmo produto.

1.6. Ambientes de Desenvolvimento Integrado

O Ambiente de Desenvolvimento Integrado é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de *software*, ele tem o objetivo de agilizar este processo. Muitas pessoas usam a sigla IDE para referenciar a este tipo de programa, essa sigla vem do nome em inglês (*Integrated Development Environment*).

As características/ferramentas mais comuns encontradas nos IDEs são:

- Editor – local de edição do código fonte do programa;
- Compilador – traduz o código fonte do programa no executável;
- Depurador – auxilia no processo de encontrar defeitos no código fonte do programa;

² Existem outras técnicas além da compilação, como por exemplo, a Tradução e a Híbrida, mas não entraremos em detalhes nesses tópicos, por não fazer parte do escopo do curso.

- Modelagem – criação de diagramas explicativos para entender e documentar o código fonte;
- Geração de código – criação de código automaticamente de acordo com *padrões* de código comumente utilizados para solucionar problemas rotineiros.

Nesse curso iremos utilizar a linguagem de programação chamada *Java* e para isso precisaremos do compilador dela. Nós usaremos também a IDE chamada *Netbeans*. A Figura 5 mostra uma tela desse software. Aqui é importante ressaltar que ambos os programas têm versões gratuitas e são utilizados por milhares de profissionais.

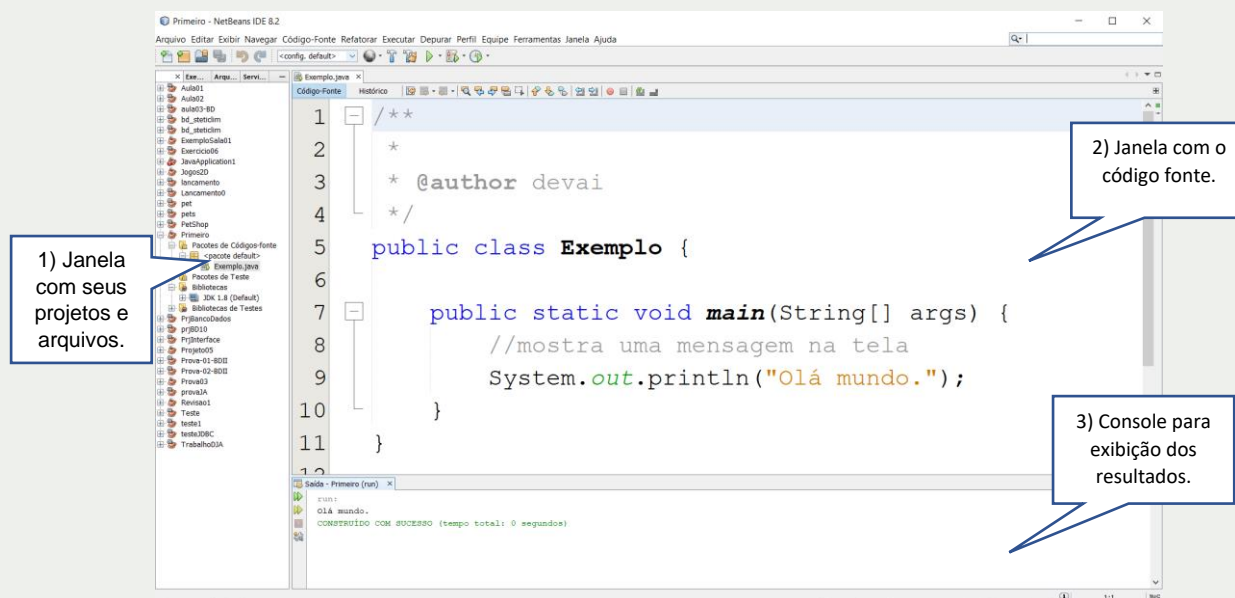


Figura 5 – Tela de exemplo do Netbeans.
Fonte: próprio autor.



Atividade: Os downloads dos dois softwares podem ser feitos diretamente dos sites das empresas mantenedora dos produtos. Contudo, vá até a sala virtual e faça o download dos arquivos através dos links “Download do Compilador Java” e “Download da IDE Netbeans”. Antes de instalá-los, assista à mídia digital indicada abaixo.



Mídia digital: Na sala virtual, assista ao vídeo “Instalando os programas” para apreender como é feita a instalação dos softwares e depois execute os mesmos passos no seu computador.



Mídia digital: Antes de prosseguirmos, assista também ao vídeo “Introdução aos algoritmos” para fazer uma revisão de tudo o que apresentado até agora.



Dica do Professor: O aprendizado de algoritmos é contínuo. Um conceito novo depende de outros já apresentados. Assim, antes de avançar para a próxima seção, tenha certeza que entendeu os conceitos apresentados até aqui e que assistiu todos os vídeos e instalou os softwares requisitados.

1.7. A linguagem Java

Java é uma linguagem de programação introduzida em 1995 pela Sun Microsystems. A história começa em 1991, em *San Hill Road*, empresa filiada à Sun (da qual hoje pertence à empresa Oracle), a empresa era formada por um time de engenheiros liderados por Patrick Naughton, Sun Fellow e James Gosling.

A ideia inicial do Java era permitir que os aparelhos eletrônicos se comunicassem entre si. Mas foi em 1995 que a Sun viu uma oportunidade na Web, nessa época, nas páginas não existia muita interatividade, apenas conteúdos estáticos eram exibidos. Então nesse ano a Sun anunciou o ambiente Java, sendo um absoluto sucesso, gerando uma aceitação aos navegadores populares da época. Hoje o Java é uma das linguagens mais usadas em todo o mundo.



Mídia digital: Para aprofundarmos o conhecimento nessa linguagem e aprender conceitos importantes, assista ao vídeo “Introdução ao Java”.

Concluída essa semana de estudos é hora de uma pausa para a reflexão. Faça a leitura (ou releitura) de tudo que lhe foi sugerido, assista aos vídeos propostos e analise todas essas informações. Esse intervalo é importante para amadurecer as novas concepções que esta etapa lhe apresentou!

Nos encontramos na próxima semana.

Bons estudos!



Objetivos

Nessa segunda semana você irá aprender e praticar sobre as instruções sequenciais e sobre todos os elementos que as compõem, como os operadores e instruções de entrada/saída de dados.

2.1 Considerações iniciais

Já aprendemos que um algoritmo resolve um problema através de um conjunto de instruções ou ações. Sabemos também que a arquitetura de Von Neumann apresenta um ciclo de execução que busca uma próxima instrução, faz a decodificação e a executa.

Contudo, a busca da próxima instrução pode ocorrer de diferentes modos, o ciclo mais básico de execução de um algoritmo é feito de forma **sequencial** da primeira linha de código (linha mais acima) até a última (linha mais abaixo), executando as instruções de cada linha da esquerda para a direita, além disso, cada linha é executada somente após o término da linha anterior (Veja a Figura 6).

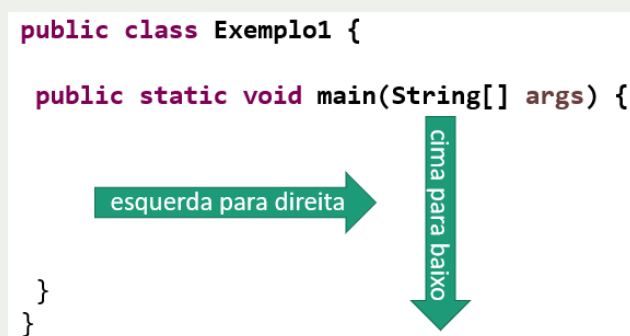


Figura 6 – Fluxo de execução sequencial de algoritmos.
Fonte: próprio autor.



Mídia digital: Como primeiro contato com os comandos sequenciais, assista ao vídeo “Indentação, palavras reservadas e comentários”.

Agora vamos entender mais o que pode ser cada linha de código ou quais são as instruções que existem nessas linhas. Vamos começar com a declaração de variáveis.

2.2 Variáveis

Analisando com mais detalhes a Figura 2 que representa a arquitetura de Von Neumann, podemos perceber que a memória é dividida em espaços ou células, cada um desses pedaços armazenam uma informação e ele tem um endereço fixo, ou seja, para acessar uma determinada posição da memória devemos informar qual o número queremos acessar. Isso é semelhante aos números de uma casa em uma rua da sua cidade. O problema é que o computador usa a codificação hexadecimal e fica difícil trabalhar com esse tipo de endereçamento. Contudo, para facilitar, as linguagens de alto nível permitem dar nomes mais sugestivos à estas posições de memória, ao invés, de trabalhar com números do tipo “12B56E3”. É então que surge o conceito de **variáveis**.

Variáveis são espaços reservados e nomeados na memória do computador, elas são destinadas ao armazenamento de dados necessários para a solução de um problema (DEITEL, 2011). Cada variável deve ter um tipo de dados vinculado a ela. Por exemplo: variáveis do tipo “int” recebem apenas números inteiros, variáveis do tipo “float” recebem apenas números reais. Além disso, toda variável deve ser declarada antes de seu uso. A Figura 7 mostra um exemplo de declaração de variáveis em Java:

```

public class Exemplo1 {
    public static void main(String[] args) {
        1) Declaração da
        variável chamada
        idade.
        int idade;
        2) Declaração e
        inicialização da
        variável chamada
        total.
        int total = 0;
        idade = 18;
        3) Atribuição do
        valor 18 à
        variável idade.
    }
}

```

O diagrama mostra um código Java dentro de uma classe `Exemplo1`. O método `main` contém três linhas de código relacionadas a variáveis. A primeira linha, `int idade;`, é apontada por uma caixa de texto explicando a declaração da variável `idade`. A segunda linha, `int total = 0;`, é apontada por uma caixa de texto explicando a declaração e inicialização da variável `total`. A terceira linha, `idade = 18;`, é apontada por uma caixa de texto explicando a atribuição do valor 18 à variável `idade`.

Figura 7 – Declaração de variáveis.
Fonte: próprio autor.

Os tipos de dados mais utilizados são: numéricos, lógicos e literais ou caractere, que descreveremos a seguir. Numéricos: dividem-se em dois grupos: inteiros e reais. Os números inteiros podem ser positivos ou negativos e não possuem parte fracionária. Os números reais podem ser positivos ou negativos e possuem parte fracionária.

As variáveis do tipo lógicas são chamadas de booleanas (oriundos da álgebra de Boole) e podem assumir os valores verdadeiro ou falso, mas com valores em inglês, ou seja, *true* ou *false*.

As variáveis dos tipos caractere ou literais, são tipos de dados formados, respectivamente, por um único caractere ou por uma cadeia de caracteres. Esses caracteres podem ser as letras maiúsculas, as letras minúsculas, os números (não podem ser usados para cálculos) e os caracteres especiais (&, #, @, ?, +, etc.). Quando a variável recebe apenas um caractere, ela será do tipo “*char*”, por exemplo ‘F’ para feminino e ‘M’

para masculino). Se a variável precisar armazenar mais de um caractere, ela deve ser do tipo “String”, por exemplo, uma variável para armazenar o nome completo de sua mãe. A Tabela 1 apresenta um resumo de todos os tipos de dados.

Tipos	Primitivo	Valores possíveis		Valor Padrão	Tamanho	Exemplo
		Menor	Maior			
Inteiro	byte	-128	127	0	8 bits	byte ex1 = (byte)1;
	short	-32768	32767	0	16 bits	short ex2 = (short)1;
	int	-2.147.483.648	2.147.483.647	0	32 bits	int ex3 = 1;
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	0	64 bits	long ex4 = 1l;
Ponto Flutuante	float	-1,4024E-37	3.40282347E + 38	0	32 bits	float ex5 = 5.50f;
	double	-4,94E-307	1.79769313486231570E + 308	0	64 bits	double ex6 = 10.20d; ou double ex6 = 10.20;
Caractere	char	0	65535	\0	16 bits	char ex7 = 194; ou char ex8 = 'a';
Booleano	boolean	false	true	false	1 bit	boolean ex9 = true;

Tabela 1 – Tipos de dados primitivos em Java.
Fonte: Adaptado de (Deitel, 2011).



Mídia digital: Vá até a sala virtual e assista ao vídeo “Variáveis” para reforçar os conceitos apresentados até aqui e também para aprender mais sobre conversão de tipo e sobre constantes.

2.3 Comandos de exibição de dados

Muitas dos algoritmos exigem que dados sejam exibidos em tela, existem diversas formas de fazer isso, para esse curso vamos usar o objeto *System.out* que é a saída padrão. Ele permite exibir *Strings* no console (terminal) quando a aplicação Java é executada. Veja na Figura 8 um exemplo de saída de dados no sistema operacional Windows, a imagem mostra uma mensagem que o algoritmo imprimiu para o professor.

```

C:\Users\devai\workspace_Sala\Aula19-01\bin>java Exemplo1
Olá professor.
Bom dia, tudo bem com você?
Professor, agora são 7:15
  
```

Figura 8 – Exemplo de exibição/impressão de dados.
Fonte: próprio autor.

Dentro do objeto *System.out* existem métodos para gerar saídas de Strings, entre eles têm-se: *println*, *print* e o *printf*. O método *System.out.println()* gera um texto, cria uma nova linha abaixo da atual e então posiciona o cursor nesta linha. Veja abaixo um exemplo

do método sendo empregado para exibir duas mensagens no console (esse trecho de código foi digitado dentro do método *main* do algoritmo):

```
//...
System.out.println("Olá professor.");
System.out.println("Bom dia, tudo bem com você?");
//...
```

Figura 9 – Exemplo de instrução para impressão de mensagens na tela.
Fonte: próprio autor.

O mesmo efeito pode ser obtido, mas agora usando variáveis.

```
//...
String texto1 = "Olá professor.";
String texto2 = "Bom dia, tudo bem com você?";

System.out.println(texto1);
System.out.println(texto2);
//...
```

Figura 10 – Exemplo de instrução para impressão utilizando variáveis.
Fonte: próprio autor.



Mídia digital: Vá até a sala virtual e assista ao vídeo “Comandos para saída de dados”, a ideia é reforçar os conceitos apresentados nessa seção e também para aprender mais sobre os métodos `print` e `printf`.

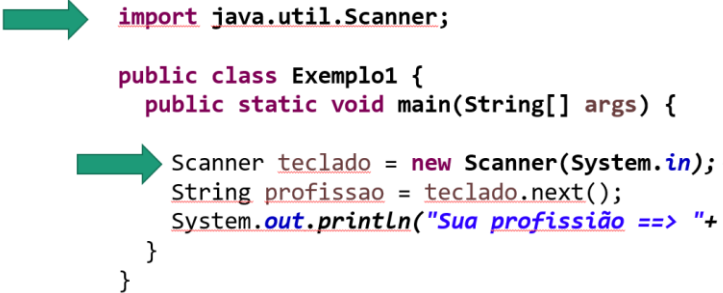
2.4 Comandos de entrada de dados

A maioria dos programas exigem interatividade com os usuários. Ou seja, os algoritmos precisam de informações do mundo externo para executar algum processamento, por exemplo, ao enviar o imposto de renda, o contribuinte deve informar o nome completo e o CPF, então a Receita Federal verifica se o número do documento está correto ou não.

O comando de entrada de dados é utilizado para receber dados digitados pelo usuário. Os dados recebidos são armazenados em variáveis. Uma das formas de entrada utilizada na linguagem *Java* é por meio da classe chamada *Scanner*, que requer um comando de importação do pacote “*java.util*” logo no início do algoritmo.

Veja um exemplo na Figura 11. As setas na cor verde indicam, respectivamente, a importação da classe e a criação do objeto que será responsável por ler os dados informados. Nesse exemplo, foi dado o nome de “teclado” ao objeto que lê os dados. A cada vez que o comando “*teclado.next()*” é encontrado, a execução do algoritmo para e fica esperando pela entrada/digitação de um valor do tipo *String* pelo usuário. Nesse

exemplo da Figura 11, o algoritmo lê a profissão de uma pessoa e depois imprime no console essa profissão.



```
import java.util.Scanner;

public class Exemplo1 {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        String profissao = teclado.next();
        System.out.println("Sua profissão ==> " + profissao);
    }
}
```

Atenção: você precisa somente de um objeto do tipo Scanner (teclado) para fazer a leitura de dados para todo seu programa.

Figura 11 – Exemplo de instrução para impressão utilizando variáveis.
Fonte: próprio autor.



Mídia digital: Agora volte à sala virtual e assista ao vídeo “Comandos para entrada de dados”, a ideia é reforçar os conceitos apresentados nessa seção e também para aprender mais sobre os métodos de leitura como `nextInt` e `nextFloat`.

Uma instrução ou passo sequencial, não se resume apenas às declarações de variáveis e à entrada e saída de dados. Na maioria das vezes, esses passos são **expressões**. Em termos computacionais, essas expressões estão intimamente ligadas ao conceito de fórmula matemática, onde um conjunto de variáveis e constantes numéricas relacionam-se por meio de operadores compondo uma fórmula que, uma vez avaliada, resulta num valor. Na verdade, esse termo “expressão” aplicado à computação, assume uma conotação mais ampla, uma vez que combina posições de memória à constantes e operadores. Esses últimos são elementos fundamentais nessas expressões ou passos. Eles são elementos funcionais que atuam sobre operandos e produzem um determinado resultado. As próximas seções apresentam os diferentes tipos existentes.

2.5 Operador de atribuição

O operador de atribuição, representado pelo símbolo de igualdade (=), não é novidade para nós. Ele permite inserir/alterar facilmente valores nas variáveis. Veja um exemplo de código na Figura 12. Mais uma vez, esse trecho de código foi inserido dentro do método `main` do projeto exemplo. Nele, temos duas variáveis (a, b) e começamos usando o operador de atribuição para inserir o valor de 10 na variável “a”. Logo em seguida substituímos o valor para 20, ou seja, o valor inicial foi perdido. Para finalizar, atribuímos o valor de 30 para “b” e jogamos o valor de “b” em “a”. Mais uma vez o valor de “a” foi substituído. Assim, na última impressão na tela, o valor exibido será o de 30. Como não alteramos o valor da variável “b”, ela também terá o valor de 30.

```
//...
int a;
a = 10;
System.out.println(a);

a = 20;
System.out.println(a);

int b = 30;
a = b;
System.out.println(a);
//...
```

Figura 12 – Exemplo de uso do operador de atribuição.
Fonte: próprio autor.

2.6 Operadores aritméticos e acumuladores

Os operadores aritméticos são símbolos que nos permitem executar operações matemáticas. Eles são aplicados às variáveis numéricas e/ou números fixos. A Tabela 2 resume os principais símbolos disponíveis.

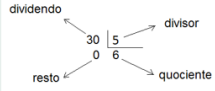
Operador	Exemplo	Descrição
+	$x + y$	Soma o conteúdo de X e de Y
-	$x - y$	Subtrai o conteúdo de Y do conteúdo de X
*	$x * y$	Multiplica o conteúdo de X pelo conteúdo de Y
/	x / y 	<p>Obtém o quociente da divisão de X por Y. Se os operandos são inteiros, o resultado da operação será o quociente inteiro da divisão. Se os operandos são reais, o resultado da operação será a divisão. Por exemplo: <code>int z = 5/2;</code> → a variável z receberá o valor 2. <code>double z = 5.0/2.0;</code> → a variável z receberá o valor 2.5</p>
%	$x \% y$	Obtém o resto da divisão de X por Y (resto como um nº inteiro)

Tabela 2 – Principais operadores aritméticos em Java.
Fonte: Adaptado de (Deitel, 2011).

Esses operadores matemáticos se tornam mais interessantes quando usados em conjunto com o comando de atribuições, pois o resultado pode ser armazenado na memória. Veja na Figura 13 um exemplo. A variável chamada “resultado” recebe a soma dos valores armazenados nas variáveis “x” e “y”. Nessa figura mostramos também a impressão direta dos valores obtidos na aplicação dos outros operadores aritméticos.

O uso de uma variável para armazenar o valor calculado vai depender do contexto. Se você não precisar do resultado em outros pontos do algoritmo, você pode imprimir o valor final e não precisa guardá-lo na memória.


```

int x = 5;
int y = 2;

int resultado = x + y;
System.out.println(resultado); // 7

System.out.println(x - y); // 3
System.out.println(x * y); // 10
System.out.println(x / y); // 2 (divisão de inteiros)
System.out.println(x % y); // 1

```

Figura 13 – Exemplo de uso dos operadores aritméticos.
Fonte: próprio autor.

Vinculado a esses operadores aritméticos básicos, existe uma operação corriqueira na vida do programador que o de acumular valores em uma mesma variável.

Por exemplo, poderíamos implementar um somador em um caixa de supermercado, acumulando assim, na variável “total” todas as compras do cliente. Essa implementação é realizada fazendo com que a variável “total” receba o seu próprio valor **mais** o valor parcial de cada execução. Ou seja, o valor dos produtos já registrados, mais os novos produtos que estão sendo lidos pelo equipamento de leitura de código de barras.



Figura 14 – Ilustração de uso de acumuladores.
Fonte: Site de imagens Pexels³. Acessado em 23/02/2022.



Mídia digital: Agora volte à sala virtual e assista ao vídeo “Operadores aritméticos e acumuladores”, a ideia é reforçar os conceitos apresentados nessa seção e principalmente aprender quais são e como usar os operadores de acumulação.

³ Link da imagem: <http://www.pexels.com/pt-br/foto/negocio-empresa-compra-aquisicao-4199490/>

2.7 Operadores de incremento e os relacionais

Existe um caso especial de acumulador apresentado na Seção 2.6. Isso acontece quando queremos acumular ou decrementar o valor fixo de 1, isso é feito para indicar que um evento ocorreu. Por exemplo, poderíamos implementar um contador em uma fila de banco, incrementando a variável “qtde” para cada nova pessoa na fila. Essa implementação é realizada fazendo com que a variável “qtde” receba o seu próprio valor mais o valor 1 a cada execução. A Tabela 3 resume os operadores de incremento e decremento.

Operador	Exemplo	Descrição
++	x++	Equivale a $X = X + 1$
++	y = ++x	Equivale a $X = X + 1$ e depois $Y = X$
++	y = x++	Equivale a $Y = X$ e depois $X = X + 1$
--	x--	Equivale a $X = X - 1$
--	y = --x	Equivale a $X = X - 1$ e depois $Y = X$
--	y = x--	Equivale a $Y = X$ e depois $X = X - 1$

Tabela 3 – Operadores de incremento e decremento em Java.
Fonte: Adaptado de (Deitel, 2011).

A Figura 15 mostra o exemplo de uso dos operadores. Os valores depois dos símbolos “//” representam o que será impresso na tela. Perceba que existe uma grande diferença ao usar os operadores “++” e “--”, antes e depois das variáveis. Quando esses operadores estão depois da variável, a operação de incremento ou decremento ocorre somente depois de executar a expressão, ou seja, primeiro imprime depois incrementa. Por outro lado, quando esses operadores estão antes da variável, a operação de incremento ou decremento ocorre antes de executar a expressão, ou seja, primeiro incrementa ou decrementa e só depois imprime.

```
int p = 4;
System.out.println(p++); // 4
System.out.println(p);   // 5
System.out.println(++p); // 6
System.out.println(p);   // 6

p = 4;
System.out.println(p--); // 4
System.out.println(p);   // 3
System.out.println(--p); // 2
System.out.println(p);   // 2
```

Figura 15 – Exemplo de uso dos operadores de incremento/decremento.
Fonte: próprio autor.



Mídia digital: Agora volte à sala virtual e assista ao vídeo “Incremento, decremento de valores e os operadores relacionais”, a ideia é reforçar os conceitos apresentados até aqui nessa seção e aprender os conceitos sobre o próximo assunto, os operadores relacionais.

Depois que conversamos sobre os operadores aritméticos fica fácil falar sobre os operadores relacionais. Eles são utilizados para comparar valores, o resultado de uma expressão relacional é um sempre um valor booleano (verdadeiro ou falso). A Tabela 4 resume os operadores relacionais.

Operador	Exemplo	Descrição
==	x == y	O conteúdo de X é igual ao conteúdo de Y
!=	y != x	O conteúdo de X é diferente do conteúdo de Y
<=	x < y	O conteúdo de X é menor que o conteúdo de Y
>=	x > y	O conteúdo de X é maior que o conteúdo de Y
<	x <= y	O conteúdo de X é menor ou igual ao conteúdo de Y
>	x >= y	O conteúdo de X é maior ou igual ao conteúdo de Y

Tabela 4 – Operadores de relacionais em Java.
Fonte: Adaptado de (Deitel, 2011).

A Figura 16 mostra o exemplo de uso dos operadores. Os valores depois dos símbolos “//” simulam o que será impresso na tela.

```
boolean resposta;

resposta = (10 == 11);
System.out.println(resposta); // false

resposta = (a <= b);
System.out.println(resposta); // true

resposta = ((2*2 + 4) != 11);
System.out.println(resposta); // true

String n1 = "Mary";
String n2 = "Mary";
resposta = (n1 == n2);
System.out.println(resposta); // false

resposta = n1.equals(n2);
System.out.println(resposta); // true
```

Figura 16 – Exemplo de uso dos operadores relacionais.
Fonte: próprio autor.

2.8 Operadores lógicos

O último conjunto de operadores que vamos abordar nesse curso são os operadores lógicos. Eles são responsáveis por ligar ideias/expressões e o resultado é um valor booleano (verdadeiro ou falso). Esses operadores seguem a mesma ideia da nossa linguagem natural. Se for necessário expressar uma ideia de junção, nós podemos usar a conjunção “e”. Um exemplo de frase: “uma pessoa só é feliz se tem dinheiro **e** tem saúde”. Se a ideia for dar opções, nós podemos usar o conectivo “ou”, alteramos o exemplo temos, “uma pessoa só é feliz se tem dinheiro **ou** tem saúde”. Nesse exemplo ligamos duas condições opcionais em uma única frase.

Em programação não é diferente. Em *Java* existem três operadores: o símbolo “&&” representa a conjunção “e”, “||” representa a disjunção “ou” e o operador “!” (não) corresponde à negação. No inglês, temos respectivamente os termos: “*and*”, “*or*” e “*not*”.

A Figura 17 apresenta esses operadores na prática. Vamos analisar com mais detalhes as quatro linhas destacadas com uma seta verde. A primeira marcação mostra o uso do operador && (e). Essa linha expressa a ideia de que a pessoa só vai poder viajar se tiver o dinheiro **e** o tempo necessário. A segunda marcação (seta) expressa a ideia que a pessoa fará a prova de recuperação se tiver nota menor que 60 **ou** tiver uma frequência menor que 75. A terceira seta destaca o trecho do algoritmo que usa a negação, nesse caso, apesar da variável ter o valor “falso”, a impressão será “verdade”, pois negamos (invertemos) o valor armazenado na variável. Por fim, a última marcação destaca o uso do operador lógico || (ou), mas em conjunto com os operadores relacionais e expressões aritméticas. O *Java* primeiro resolve as expressões aritméticas, depois consulta o resultado da igualdade/desigualdade para, finalmente, avaliar se algum dos lados da expressão lógica é verdade.

```
boolean temDinheiro = true;
boolean temTempo = false;
boolean viajar = temDinheiro && temTempo;  ←
System.out.println(viajar);                // false

boolean notaMenorDe60 = true;
boolean frequenciaMenorDe75 = false;
boolean recuperacao = notaMenorDe60 || frequenciaMenorDe75; ←
System.out.println(recuperacao);           // true

boolean inverte = true;
System.out.println(!inverte);              // false
System.out.println(!inverte);              // true

boolean menorDeIdade = false;
System.out.println("Pode entrar? " + !menorDeIdade); ←

boolean res = (1 != (2 - 1 * 3)) || (2 == (1 + 1 * 2)); ←
```

Figura 17 – Exemplo de uso dos operadores lógicos.

Fonte: próprio autor.



Mídia digital: Volte à sala virtual e assista ao vídeo “Operadores lógicos”, a ideia é reforçar os conceitos apresentados até aqui nessa seção e aprender sobre a precedência dos operadores em uma expressão.



Atividade: No ambiente virtual, procure pela atividade chamada “Questionário 1”. Essa atividade é avaliativa e a conclusão dela é necessária para a obtenção do seu certificado.



Atenção: O aprendizado de algoritmos requer muita prática, então refaça todos os exemplos apresentados até aqui, modifique-os e veja os resultados. Faça os exercícios propostos e procure material extra.

Concluimos assim a segunda semana de estudos, mais uma vez é hora de uma pausa para assimilar o conteúdo. Faça a leitura (ou releitura) de todo o texto, principalmente, assista aos vídeos propostos e analise todas essas informações.

Nos encontramos na próxima semana.

Bons estudos!



Objetivos

Nessa terceira semana, você irá compreender o que são e como usar as estruturas condicionais em um algoritmo.

3.1 Conceitos iniciais

Muito dos problemas que resolvemos no dia a dia são guiados por condições. Por exemplo, "se chover então tenho que levar guarda-chuva". Um outro exemplo seria, "se tiver dinheiro então eu vou, senão eu fico em casa". Essas condições podem ser representadas em um diagrama de fluxo, os quais são mostrados na Figura 18, aonde a letra "V" representa um valor verdadeiro para a condição e "F" significa o valor falso.

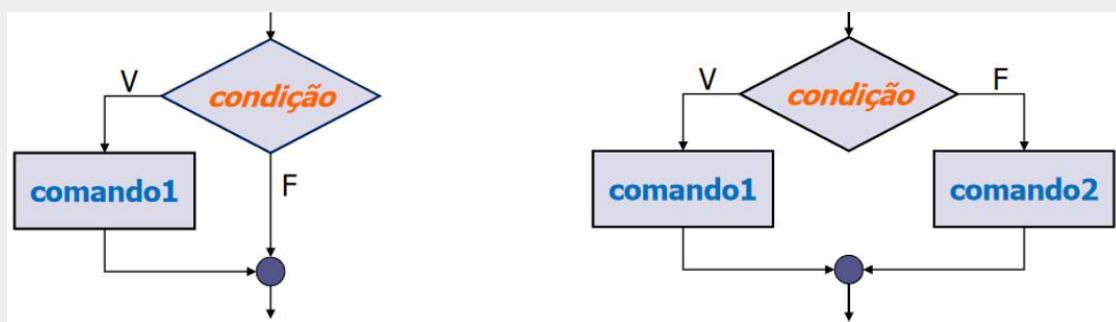


Figura 18 – Exemplo de fluxo de execução condicional.
Fonte: próprio autor.

Esse mesmo comportamento pode ser necessário durante a execução de um algoritmo. Ou seja, a próxima instrução a ser executada não seria a instrução que vem na sequência e sim, algumas linhas de código abaixo. Esse comportamento é realizado através das estruturas condicionais.

Estruturas Condicionais são condições impostas ao programa para que ele receba diferentes alternativas de funcionamento. Ao adicionarmos alternativas a um programa nós aumentamos a flexibilidade do mesmo, permitindo que ele funcione de formas diferentes dependendo da condição que ele atende (ASCENCIO, 2008).

Existem quatro formas básicas para implementarmos essas condições em *Java*, as quais são apresentadas ao longo dessa semana nas próximas seções.

3.2 Comando condicional simples

A primeira instrução condicional que vamos estudar é chamada de instrução condicional simples, ou simplesmente, instrução “se”. Contudo, o Java usa o termo traduzido para o inglês, ou seja, a palavra “if”. O comando “se” é utilizado no programa quando uma ação depende de uma inspeção ou teste de condição (expressão lógica). A Figura 19 apresenta a sintaxe do comando.

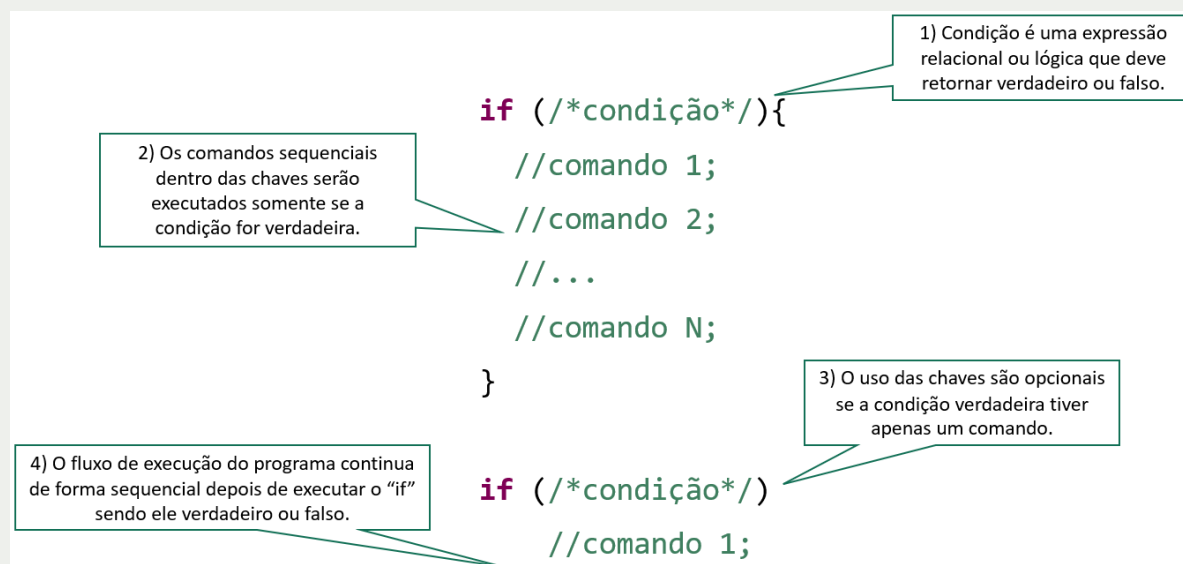


Figura 19 – Sintaxe da estrutura condicional simples.
Fonte: próprio autor.

Já a Figura 20 mostra um exemplo de uso. O algoritmo requisita a idade de uma pessoa. Se a pessoa for maior de idade, ou seja, tem dezoito anos ou mais, o programa apresenta uma mensagem indicando que ela poderá viajar sozinha.

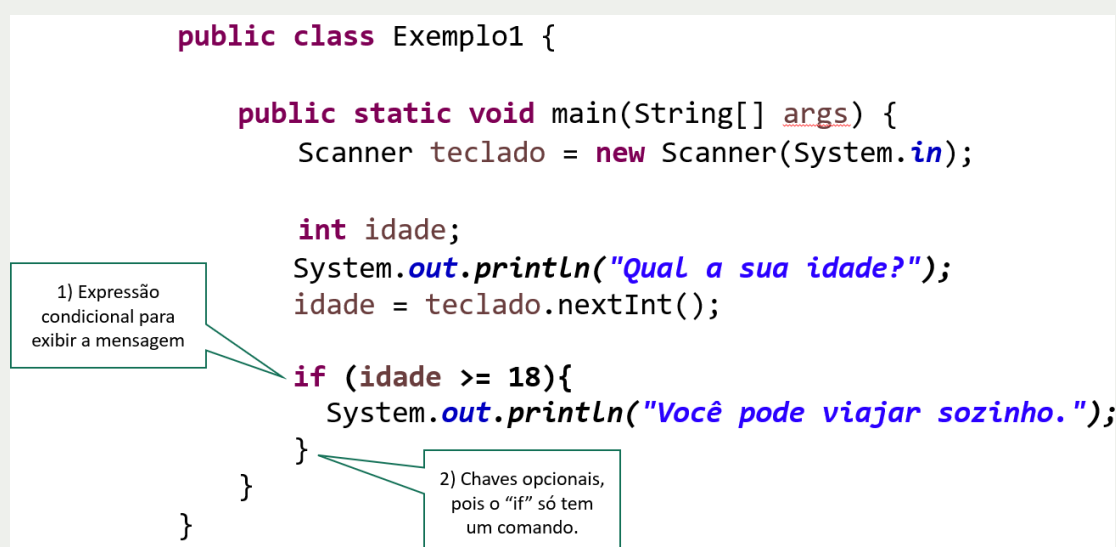


Figura 20 – Exemplo 01 - condicional simples.
Fonte: próprio autor.

Agora, como um segundo exemplo, vamos criar um algoritmo que receba dois números inteiros e imprima o maior deles. A Figura 21 mostra o código fonte para resolver esse problema. Veja que da linha 3 até a 9 temos as instruções de entrada de dados. A novidade aqui acontece das linhas 11 até a 19. Nesse trecho implementamos as condições para impressão na tela.

```

1 public class Exemplo1 {
2     public static void main(String[] args) {
3         Scanner teclado = new Scanner(System.in);
4
5         System.out.println("Qual o primeiro numero");
6         int x1 = teclado.nextInt();
7
8         System.out.println("Qual o segundo numero");
9         int x2 = teclado.nextInt();
10
11         if (x1 > x2) {
12             System.out.printf("%d é o maior numero", x1);
13         }
14         if (x2 > x1) {
15             System.out.printf("%d é o maior numero", x2);
16         }
17         if (x2 == x1) {
18             System.out.printf("Os números são iguais");
19         }
20     }
21 }

```

Figura 21 – Exemplo 02 - condicional simples.
Fonte: próprio autor.



Mídia digital: Volte à sala virtual e assista ao vídeo “Estrutura condicional simples”, a ideia é reforçar os conceitos apresentados e ver mais alguns exemplos.

3.3 Comando condicional composto

No exemplo anterior (Seção 3.2) é fácil notar que uma condição já exclui a outra. Assim, se uma condição é verdadeira, seria interessante descartar as demais opções e, isso é obtido através da cláusula “else” (senão em português). Ela é ideal para descrever ideias com condicional encadeado (ou aninhado). Veja na Figura 22 a sintaxe desse comando. Repare também que quando o fluxo escolhido na execução contem somente uma linha, as chaves são opcionais.

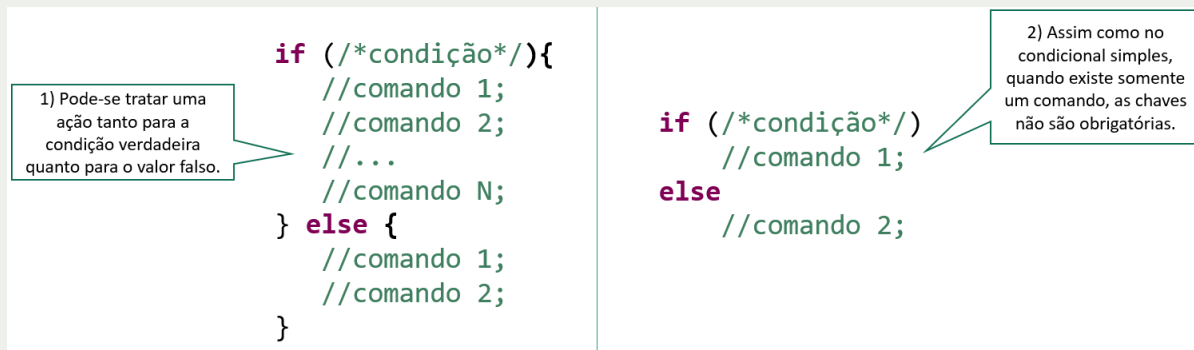


Figura 22 – Sintaxe da estrutura condicional composta.
Fonte: próprio autor.

A Figura 23 refaz o algoritmo que verifica a maioria de uma pessoa. Veja que agora o algoritmo tem dois tipos de mensagens, as quais são acionadas pelo resultado da condição “if”. Nesse algoritmo um caminho exclui a possibilidade de execução do outro.

```

public class Exemplo1 {

    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);

        int idade;
        System.out.println("Qual a sua idade?");
        idade = teclado.nextInt();

        if (idade >= 18){
            System.out.println("Você pode viajar sozinho.");
        }
        else {
            System.out.println("Você NÃO pode viajar sozinho.");
        }
    }
}

```

Figura 23 – Exemplo 01 - condicional composto.
Fonte: próprio autor.

Agora, como um segundo exemplo do condicional composto, vamos criar um algoritmo que indica se o preço da compra de livros foi boa, razoável ou ruim (cara). Nesse caso temos três possíveis caminhos de execução. A Figura 24 mostra o código fonte para resolver esse problema. O comando “if” começa na linha 11 e termina na 19. Nesse trecho implementamos as condições para impressão na tela. Se a condição da linha 11 for satisfeita, o Java executa a linha 12 e passa o fluxo de execução para a linha 20.

```

1 public class Exemplo1 {
2
3     public static void main(String[] args) {
4
5         double livroLinux = 78.60;
6         double livroBancosDados = 56.75;
7         double total = livroLinux + livroBancosDados;
8
9         System.out.println("O preço total é " + total );
10
11        if (total < 120.00 ) {
12            System.out.println("O preço está bom!");
13        }
14        else if (total > 180.00 ){
15            System.out.println("Livros muito caros!");
16        }
17        else {
18            System.out.println("Preço razoável.");
19        }
20    }
21 }

```

Figura 24 – Exemplo 02 - condicional composto.
Fonte: próprio autor.



Mídia digital: Volte à sala virtual e assista ao vídeo “Estrutura condicional composta”, a ideia é reforçar os conceitos apresentados e ver mais alguns exemplos. Esse vídeo aula também já faz uma introdução das duas próximas seções.

3.4 Comando *switch*

Existem situações que temos muitas opções mutuamente exclusivas, isto é, se uma situação for executada, as demais não serão. Quando este for o caso, o comando seletivo “*switch*” é o mais indicado do que usar vários “*if - else*” (apresentado na Seção 3.3). Veja na Figura 25 como é a sintaxe desse comando.

```

switch (variável)
{
    case valor1: lista de comandos;
                break;
    case valor2: lista de comandos;
                break;
    //....
    default: lista de comandos;
}

```

Quando o valor da variável não coincidir com aqueles especificados nos *cases*, será executado, então, o *default*.

O comando *break* deve ser utilizado para impedir a execução dos comandos definidos nos *cases* subsequentes quando uma condição é satisfeita.

Figura 25 – Sintaxe da estrutura/comando *switch*.
Fonte: próprio autor.

A linha “switch (variável)” analisa o valor de uma variável para decidir qual “case” será executado. Cada “case” está associado a UM possível valor da variável, que deve ser obrigatoriamente do tipo *int*, *short*, *byte* ou *char*.

A Figura 26 mostra um exemplo de uso do “switch”. O algoritmo requisita a entrada de um número inteiro. Caso o número inserido for o valor “1”, o programa irá imprimir “Número 1”. Caso o número inserido for “2”, a frase impressa será “Número 2”. A opção indicada como “default” será acionada para qualquer outro número informado. Essa palavra traduzida para o português significa: padrão. Esse é justamente o comportamento do comando, ou seja, se não entrar na primeira ou na segunda opção, por padrão, será impresso na tela: “Outro número”.

```
public class Exemplo1 {

    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);

        int x;
        System.out.println("Entre com um número");
        x = teclado.nextInt();

        switch (x)
        {
            case 1: System.out.println("Número 1");
                    break;
            case 2: System.out.println("Número 2");
                    break;
            default: System.out.println("Outro número");
        }
    }
}
```

Figura 26 – Exemplo 01 – comando switch.
Fonte: próprio autor.

3.5 Operador condicional ternário

Java oferece também um operador que proporciona uma forma mais compacta de se representar decisões simples. O **operador condicional**, cuja sintaxe é:

condição ? expressão1:expressão2

Esse operador ternário é uma expressão, isso significa que ele deve retornar um valor. Bom, a melhor forma de entendê-lo é praticando. Veja alguns exemplos na Figura 27. Essa imagem mostra três trechos de código sem muita lógica entre eles. A ideia aqui é ver o comando ternário em ação. Assim, destacamos os três pontos de maior importância com setas.

```

    boolean clienteAtivo = true;
    ➡ String mensagem = clienteAtivo ? "Compra autorizada" : "Cliente bloqueado";

    System.out.println("Entre com o valor");
    x = teclado.nextInt();
    ➡ System.out.println( x > 100 ? "Valor grande" : "valor ok");

    int salario = 1000;
    int vr =0;
    ➡ vr = (salario == 9000 ? 100 : 200);

```

Figura 27 – Exemplo de usos do comando ternário.
Fonte: próprio autor.

A primeira seta destaca o primeiro uso do operador condicional. Ele funciona como um comando “*if -else*”. Nesse caso, a condição é representada por uma variável booleana. Como a variável está recebendo o valor verdade (*true*), a variável “mensagem” será preenchida com o texto “Compra autorizada”.

No segundo ponto destacado, não conseguimos prever o que será impresso, pois ele depende da execução do código, ou seja, da entrada do valor pelo usuário. Se o valor informado for maior que cem, a primeira frase será impressa. Se o valor for menor ou igual a cem, o texto “valor ok” será impresso.

No último ponto a ser analisado, a variável “vr” receberá o número 200. Pois o valor atribuído à variável salário não é igual a 9000. É como se o Java acionasse a condição “*else*”, se tivéssemos trabalhando com o comando “*if*”.



Atividade: No ambiente virtual, procure pela atividade chamada “Questionário 2”. Essa atividade é avaliativa e a conclusão dela é necessária para a obtenção do seu certificado.

Concluimos nossa penúltima semana de estudos. Assista aos vídeos propostos e analise todas essas informações com cuidado e quantas vezes forem necessárias.

Nos encontramos na próxima semana.

Bons estudos!



Objetivos

Nessa última semana, você irá compreender o que são e como usar as estruturas de repetição em um algoritmo.

4.1 Conceitos iniciais

Em muitas das tarefas cotidianas, temos que repetir ações até obter um determinado resultado. Por exemplo, você tem que caminhar até chegar ao trabalho (passo por passo) ou, você tem que continuar lendo um livro até chegar ao final dele. Com algoritmos não é diferente. Existem ações ou instruções que devem ser repetidas até que o resultado desejado seja obtido. Hipoteticamente, um algoritmo pode ter a necessidade de calcular o peso ideal de duzentas pessoas. A fórmula para o cálculo é a mesma, assim, o algoritmo deve repetir a expressão de cálculo para cada uma dessas pessoas. Uma forma seria repetir essa mesma linha de código duzentas vezes. Felizmente, existe uma maneira mais prática e isso é feito através das instruções de repetição.

Estruturas de Repetição fornecem a capacidade de executar o mesmo trecho de código diversas vezes, você pode determinar uma quantidade fixa ou criar uma condição para encerrar o ciclo (DEITEL, 2013). Essas estruturas também são conhecidas como laços ou “*loops*”, do termo em inglês. A Figura 28 mostra como as repetições podem ser representadas em um diagrama de fluxo de execução. Na primeira forma (fluxo mais à esquerda), existe uma condição no início do processo. Se ela for verdadeira as linhas de código do corpo do laço são executadas e o Java volta a verificar se a condição continua verdadeira. Enquanto ela for “*true*”, o laço se repete, quando a condição for falsa, o laço se encerra. Já na segunda forma (fluxo mais à direita), não existe uma condição no início, isso garante que o corpo do laço será executado, pelo menos uma vez. Enquanto a condição for “*true*” o laço se repete, quando a condição for falsa, o laço se encerra.

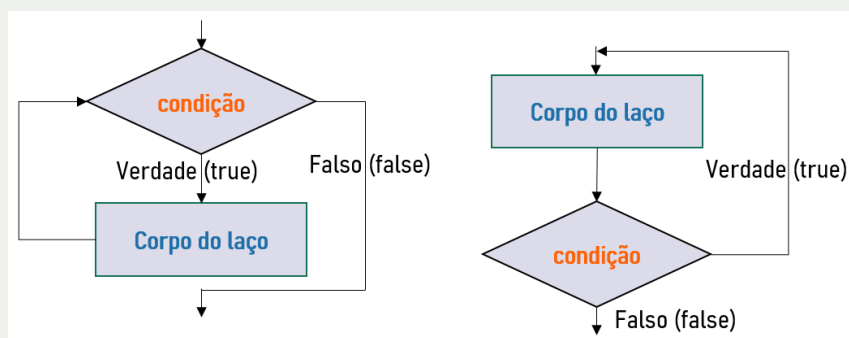


Figura 28 – Diagrama do fluxo de execução dos laços de repetição.
Fonte: próprio autor.

Existem três estruturas básicas para criar as repetições, mas qualquer que seja a forma escolhida, ela contém quatro elementos fundamentais (ASCENCIO,2008):

- **inicialização** - todo o código que determina a condição inicial da repetição;
- **condição** - é uma expressão booleana avaliada após cada leitura do corpo e determina se uma nova leitura deve ser feita ou se a estrutura de repetição deve ser encerrada;
- **corpo** - O corpo compõe-se de todas as instruções que são executadas repetidamente.
- **iteração** - A iteração é a instrução que deve ser executada depois do corpo e antes de uma nova repetição.

As três estruturas de repetição são apresentadas nas próximas seções,

4.2 Estrutura de repetição “for” (para)

A primeira forma de reexecutar um código é usando o comando “for”. Essa estrutura de repetição geralmente é utilizada quando se sabe exatamente o número de vezes que um trecho do algoritmo deve ser repetido, nesse caso é utilizado uma variável do tipo inteiro para controlar o laço. A Figura 29 mostra um exemplo dessa estrutura. Nela vemos três partes separadas pelo “;”, às quais são a inicialização, a condições de execução e a iteração. Aqui temos uma variável chamada “counter”, ela só tem validade dentro do laço, pois foi declarada dentro da instrução “for”. Ela é usada para controlar o fim da execução. Enquanto essa variável for menor ou igual a dez, as linhas de código pertencentes ao laço serão executadas. A cada execução a variável “counter” é incrementa em um, isso é feito pelo operador “++”.

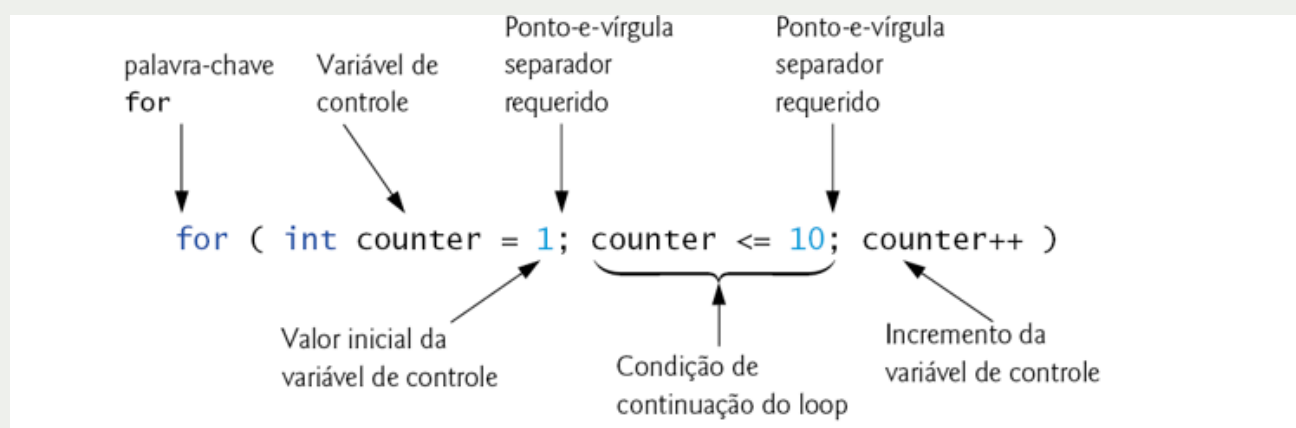


Figura 29 – Exemplo da estrutura “for”.
Fonte: próprio autor.

A Figura 30 mostra um algoritmo que recebe cinco números e acumula o valor deles em uma variável chamada “soma”. A variável foi declarada fora do laço, pois iremos usá-la também no final do algoritmo, ou seja, na hora da impressão dos resultados na tela. Como sabermos, toda variável declara dentro do laço só pode ser usada internamente dentro dele, no exemplo usamos o laço para ler e somar os cinco números.

```
int soma = 0; // inicialização da variável SOMA com o valor zero
for(int i = 1; i <= 5; i++)
{
    System.out.print("Digite um número: ");
    int num = teclado.nextInt();
    soma = soma + num; // acumulando o valor da variável NUM na variável SOMA
}
System.out.println("Soma = "+soma);
```

Figura 30 – Exemplo de uso do comando “for”.

Fonte: próprio autor.



Mídia digital: Volte à sala virtual e assista ao vídeo “Estrutura de repetição - for”, a ideia é reforçar os conceitos apresentados e ver mais alguns exemplos.

4.3 Estrutura de repetição “while” (enquanto)

A segunda estrutura de repetição é o comando “*while*”. Essa opção é indicada quando o número de repetições necessárias não forem fixas, ela usa uma condição booleana para isso. Os códigos internos serão repetidos até a condição assumir o valor falso. Inclusive, existe a possibilidade de a repetição não ser executada quando a condição assumir o valor falso, logo na primeira iteração. A Figura 31 mostra a sintaxe do comando.

```

    Expressão
    lógica
while (condição){
    //bloco de comandos a serem executados.
}
```

Figura 31 – Sintaxe da estrutura “while”.

Fonte: próprio autor.

A Figura 32 mostra um algoritmo de exemplo. Agora criamos um jogo em que o usuário deve acertar o número “mágico”. Nesse caso, o número será o nove. O jogo se repete infinitamente até o usuário acertar. Esse *loop* é controlado pela variável chamada “jogando”. Ela começa com o valor “*true*” para garantir que o “*while*” será executado pelo menos uma vez. Dentro do laço, o programa pede ao usuário um número e depois faz uma condição para ver se o valor é igual a nove, a variável “jogando” recebe “*false*” quando o usuário acerta, assim o laço será encerrado, pois a condição não é mais verdadeira.


```

boolean jogando = true;
int numeroMagico = 9;
int chute;

while (jogando){
    System.out.println("Digite um número: ");
    chute = teclado.nextInt();

    if (chute == numeroMagico){
        jogando = false;
    }
}
System.out.println("Fim do jogo");

```

Figura 32 – Exemplo de uso do comando “while”.
Fonte: próprio autor.



Mídia digital: Volte à sala virtual e assista ao vídeo “Estrutura de repetição - *while*”, a ideia é reforçar os conceitos apresentados e ver mais alguns exemplos.

4.4 Estrutura de repetição “do while” (repita enquanto)

A última estrutura de repetição que iremos estudar é o comando “do..while”. Assim como na estrutura apresentada na Seção 4.3, o do...while é indicado quando o número de repetições necessárias não forem fixas. Os comandos também serão repetidos até a condição obtiver o valor false. Contudo, a novidade é que o teste condicional ocorre no fim do corpo do laço. Assim a repetição ocorrerá pelo menos uma vez. A Figura 33 mostra a sintaxe do comando.

```

do {
    //bloco de comandos a serem executados.
}
while (condição);

```

Expressão lógica

Figura 33 – Sintaxe do comando “do while”.
Fonte: próprio autor.

A Figura 34 mostra um trecho de algoritmo de exemplo. A ideia é imprimir um menu de opções para o usuário e em seguida, ler o que o ele escolheu. O menu e a leitura de

dados têm que aparecer, pelo menos uma vez, assim escolhemos o “do while”. Com a opção armazenada na variável chamada “opc”, podemos executar ações distintas de acordo com o comando “if”. Somente depois a condição de parada é verificada. Se o usuário escolheu a opção três, o laço de repetição é encerrado, caso contrário o menu é reexibido.

```
int opc;
do{
    System.out.println("=====Menu=====");
    System.out.println("1. Mostrar o nome do aluno");
    System.out.println("2. Mostrar a idade do aluno");
    System.out.println("3. Encerrar o programa");
    opc = teclado.nextInt();

    if (opc == 1)
        System.out.println("Aluno: JHONY");
    else if (opc == 2)
        System.out.println("Idade: 19");
    else if (opc != 3)
        System.out.println("Opção inválida, tente novamente.");
}
while (opc != 3);
```

Figura 34 – Exemplo de uso do comando “do while”.
Fonte: próprio autor.



Mídia digital: Volte à sala virtual e assista ao vídeo “Estrutura de repetição – do while”, a ideia é reforçar os conceitos apresentados, ver mais alguns exemplos aprender sobre os comandos break e continue.

4.5 Modificadores de fluxo de laço (break, continue)

Como visto no vídeo sugerido da seção anterior. Existem dois comandos que podem desviar o fluxo normal de execução das estruturas de repetição. Os comandos são o “continue” e o “break”. O comando **continue** encerra imediatamente a iteração atual do laço de repetição, mas continua executando a estrutura de repetição.

A Figura 35 mostra um trecho de algoritmo (com “while”) para executar um código por dez vezes. Mas no corpo do laço existe um comando “continue” e, ele só é acionado quando a condição do “if” for verdadeira, ou seja, quando a variável “contador” armazenar o valor quatro. Quando o comando “continue” é encontrado, as linhas abaixo dele não são executadas, o fluxo normal do laço é desviado de volta para o início do “while”.

Repare que nessa mesma imagem está o resultado da execução. Todos os números entre 1 e 9 foram impressos, menos o quatro. Pois justamente nesse número, o comando `continue` é executado e o Java transfere o fluxo para o início de laço sem alcançar o `"System.out.println"`.

```
int contador = 0;
while (contador < 10){
    contador++;

    if (contador == 4)
        continue;
    System.out.println("==>" + contador);
}
System.out.println("Final do laço");
```

==>1
==>2
==>3
==>5
==>6
==>7
==>8
==>9
Final do laço

Figura 35 – Exemplo de uso do comando `"continue"`.
Fonte: próprio autor.

Já o comando **break** encerra imediatamente o laço e continuação a execução do algoritmo logo ao final da estrutura de repetição, em que ele se encontra. A Figura 36 mostra o mesmo algoritmo da Figura 34, mas agora temos o `"break"`, ao invés, do `"continue"`. Repare que nessa imagem também temos o resultado da execução. Quando a variável `"contado"` é igual a quatro, nenhum número é impresso, pois a condição `"if"` é verdadeira e o comando `"break"` é executado, saindo imediatamente do laço de repetição.

```
int contador = 0;
while (contador < 10){
    contador++;

    if (contador == 4)
        break;
    System.out.println("==>" + contador);
}
System.out.println("Final do laço");
```

==>1
==>2
==>3
Final do laço

Figura 36 – Exemplo de uso do comando `"break"`.
Fonte: próprio autor.



Atividade: No ambiente virtual, procure pelas atividades chamadas “Questionário 3” e “Questionário Final”. Essas atividades são avaliativas e a conclusão delas são necessárias para a obtenção do seu certificado.

Concluimos assim, a nossa última semana de curso. Estudados até aqui as principais estruturas para a criação de algoritmos. Mais uma vez, assista aos vídeos propostos e o texto base do curso quantas vezes forem necessárias.

Espero que tenha sido proveitoso e quero ver você no mercado de trabalho, o quanto antes. Mas lembre-se, essa foi só uma introdução, seu estudo deve ser contínuo e assim que dominar os conceitos apresentados aqui procure novos desafios, a própria Plataforma +IFMG pode te ajudar a evoluir, consulte nossos outros cursos.

Parabéns pela conclusão do curso. Foi um prazer tê-lo conosco!

Boa Sorte!



Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. Fundamentos da Programação de Computadores: Algoritmos, Pascal, C/C++ e Java. 2ª ed. São Paulo:, Pearson Education, 2008.

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. Algoritmos: Teoria e Prática 1. Tradução da segunda edição [americana] Vandenberg D. de Souza. - Rio de Janeiro: Elsevier, 2002 - Reimpressão.

DEITEL, Paul J.; DEITEL, Harvey M. C: Como programar. 6ª ed. São Paulo: Prentice Hall, 2013.

FILHO, Clézio F. História da computação [recurso eletrônico]: O Caminho do Pensamento e da Tecnologia / Clézio Fonseca Filho. – Porto Alegre: EDIPUCRS 205 p., 2007.

MANZANO, J. A. N. G.; OLIVEIRA, J. G. de. Estudo Dirigido de Algoritmos. São Paulo. Editora Érica (Coleção PD). 1997.

SEBERTA Robert, Conceitos de Linguagens de Programação, ed. Bookman, 9a edição, 2011.



Currículo do autor



Bruno Ferreira possui graduação em Ciência da Computação pelo Centro Universitário de Formiga (2003), pós-graduação em Redes de Computadores pelo Centro Universitário do Sul de Minas (2004), mestrado em Modelagem Matemática e Computacional pelo CEFET-MG (2008) e doutorado em Ciência da Computação pela UFMG (2016). Tem experiência na área de Ciência da Computação, com ênfase em Métodos Formais, Linguagens de Programação e Engenharia de Software. Trabalhou no setor privado por 8 anos como programador e analista de sistemas. Atualmente é professor efetivo do Instituto Federal de Minas Gerais - (IFMG - Campus Formiga).

Currículo Lattes: <http://lattes.cnpq.br/9437251245138635>

Feito por (professor-autor)	Data	Revisão de <i>layout</i>	Data	Versão
Bruno Ferreira	27/02/2022	Viviane L Martins	09/03/2022	1.0



Glossário de códigos QR (*Quick Response*)

		Mídia digital Apresentação do curso			Mídia digital Instalando os programas
		Mídia digital Introdução aos algoritmos			Mídia digital Introdução ao Java
		Mídia digital Indentação, comentários e palavras reservas			Mídia digital Variáveis
		Mídia digital Comandos para saída de dados			Mídia digital Comandos para entrada de dados
		Mídia digital Operadores aritméticos e acumuladores			Mídia digital Incremento, decremento e os operadores relacionais
		Mídia digital Operadores lógicos			Mídia digital Estrutura condicional simples
		Mídia digital Estrutura condicional composta			Mídia digital Estrutura de repetição - for
		Mídia digital Estrutura de repetição - while			Mídia digital Estrutura de repetição - do while



Plataforma +IFMG

Formação Inicial e Continuada EaD



A Pró-Reitoria de Extensão (Proex), desde o ano de 2020, concentrou seus esforços na criação do Programa +IFMG. Esta iniciativa consiste em uma plataforma de cursos *online*, cujo objetivo, além de multiplicar o conhecimento institucional em Educação à Distância (EaD), é aumentar a abrangência social do IFMG, incentivando a qualificação profissional. Assim, o programa contribui para o IFMG cumprir seu papel na oferta de uma educação pública, de qualidade e cada vez mais acessível.

Para essa realização, a Proex constituiu uma equipe multidisciplinar, contando com especialistas em educação, *web design*, *design* instrucional, programação, revisão de texto, locução, produção e edição de vídeos e muito mais. Além disso, contamos com o apoio sinérgico de diversos setores institucionais e também com a imprescindível contribuição de muitos servidores (professores e técnico-administrativos) que trabalharam como autores dos materiais didáticos, compartilhando conhecimento em suas áreas de

atuação.

A fim de assegurar a mais alta qualidade na produção destes cursos, a Proex adquiriu estúdios de EaD, equipados com câmeras de vídeo, microfones, sistemas de iluminação e isolamento acústica, para todos os 18 *campi* do IFMG.

Somando à nossa plataforma de cursos *online*, o Programa +IFMG disponibilizará também, para toda a comunidade, uma Rádio Web Educativa, um aplicativo móvel para Android e IOS, um canal no Youtube com a finalidade de promover a divulgação cultural e científica e cursos preparatórios para nosso processo seletivo, bem como para o Enem, considerando os saberes contemplados por todos os nossos cursos.

Parafraseando Freire, acreditamos que a educação muda as pessoas e estas, por sua vez, transformam o mundo. Foi assim que o +IFMG foi criado.

O +IFMG significa um IFMG cada vez mais perto de você!

Professor Carlos Bernardes Rosa Jr.
Pró-Reitor de Extensão do IFMG





Características deste livro:

Formato: A4

Tipologia: Arial e Capriola.

E-book:

1ª. Edição

Formato digital

