*Edvaldo Vunge*

# JAVA BÁSICO

## CONCEITOS INICIAIS DE PROGRAMAÇÃO

TRANQUILO COMO UM GRILO,
MANSO QUE NEM UM GANSO.
Edvaldo Vunge

# Agradecimentos

Em primeiro lugar, agradeço a Deus pela força e sabedoria concedidas ao longo desta jornada. Sem sua orientação, muitos momentos teriam sido ainda mais desafiadores. Aos meus pais, pela educação, amor e apoio incondicional. Vocês sempre foram a minha base, oferecendo encorajamento nos momentos de dúvida e celebrando comigo cada conquista.

Ao amor da minha vida, Juelcia Michingi, cuja presença tem sido uma fonte constante de força e inspiração. Obrigado por estar ao meu lado em cada passo dessa caminhada. O teu apoio incondicional e amor tornaram este caminho mais leve e este sonho, uma realidade.

Aos meus amigos e familiares, que de diversas formas contribuíram com palavras de incentivo, conselhos valiosos ou simplesmente estando ao meu lado nos momentos em que mais precisei. O carinho e apoio de vocês foram essenciais.

Um agradecimento especial a todos os professores e mentores que, direta ou indiretamente, me impulsionaram a buscar o conhecimento e a perseverar na busca pelos meus objetivos. Suas orientações foram fundamentais para que este livro se tornasse realidade.

Por fim, agradeço a todos que acreditam no poder dos sonhos e dos objetivos. Que este livro sirva de inspiração e lembrete de que, com determinação e esforço, tudo é possível.

# PREFÁCIO

# DEDICATÓRIA

Dedico este livro aos meus avós, Joana Luís Diogo e Ramiro Manuel Vunge, que já não estão entre nós. Em vida, ambos me inspiraram a perseguir os meus objetivos, cada um à sua maneira, transmitindo o melhor de si. Nunca serão esquecidos, pois enquanto estiverem presentes em meus pensamentos, a vossa partida será apenas física. Viverão por milhares de anos na minha memória.

Dedico também a todos aqueles que têm um objetivo na vida. Não desistam diante das dificuldades. Quem sonha alto deve esperar desafios à mesma altura. Faça da sua vida um caminho onde cada dia seja melhor do que o anterior.

# ÍNDICE GERAL

## MÓDULO 1: COMPUTADORES

## MÓDULO 2: VISÃO GERAL



## MÓDULO 3: NOÇÕES BÁSICAS DO JAVA

## MÓDULO 4: VARIÁVEIS

## MÓDULO 5: ENTRADA E SAÍDA DE DADOS

## MÓDULO 6: OPERADORES



## MÓDULO 7: CONTROLE DE FLUXO

## MÓDULO 8: STRINGS

## MÓDULO 9: ARRAYS

# MÓDULO 1: COMPUTADORES

**O computador veio para resolver os problemas que nós ainda não tínhamos.**

Hernani Soares Neto

- COMPUTADORES
- HARDWARE & SOFTWARE
- LEI DE MOORE
- A INTERNET
- SISTEMAS OPERACIONAIS
- TECNOLOGIAS MÓVEIS
- RESUMO
- EXERCÍCIOS

# Objetivos do Módulo

Este módulo tem como objetivo fornecer uma introdução abrangente aos conceitos fundamentais da computação como:

- Introduzir os conceitos básicos de computadores e seu funcionamento.
- Diferenciar hardware e software.
- Compreender a Lei de Moore e a evolução tecnológica.
- Explorar a organização dos computadores, sistemas operacionais e a internet.
- Apresentar tecnologias móveis e a interação entre sistemas operacionais como Windows e Linux.

# Introdução aos Computadores

Esta seção apresenta a história e a evolução dos computadores, desde os primeiros dispositivos mecânicos até os sistemas digitais modernos. Aborda as características fundamentais que definem um computador, incluindo a capacidade de processamento, armazenamento e comunicação.

O computador eletrônico é um dos desenvolvimentos mais importantes do século XX. Tal como a revolução industrial do século XIX, o computador e a tecnologia de informação e comunicação e as tecnologias de informação e comunicação que lhe estão subjacentes mudaram drasticamente os negócios, a cultura, o governo e a ciência, e tocaram quase todos os aspetos das nossas vidas. Este texto apresenta o campo da computação e detalha os conceitos e práticas fundamentais utilizados no desenvolvimento de aplicações informáticas.

Entrar numa nova área como a informática é um pouco como ir trabalhar para um país que nunca se visitou antes. Embora todos os países partilhem algumas características fundamentais, como a necessidade de uma língua e propensão para a cultura e para o comércio, as profundas diferenças entre estas caraterísticas de um país para outro podem ser desorientados e até debilitantes para os recém-chegados. Além disso, é difícil até descrever as características de um determinado país de uma forma definitiva, porque variam de um sítio para outro e mudam com o tempo. De forma semelhante, entrar no domínio da informática pode ser desorientador e encontrar definições claras e definição clara das suas características pode ser difícil.

No entanto, existem conceitos fundamentais subjacentes ao domínio da informática que podem ser articulados, aprendidos e implementados eficazmente. Toda a computação se baseia na utilização coordenada de dispositivos informáticos, designados por hardware, e os programas de computador que os controlam, designados por software, e todas as

aplicações de software são construídas utilizando especificações de dados e de processos, designadas por estruturas de dados e algoritmos. Estes fundamentos mantiveram-se notavelmente estáveis ao longo da história da computação, apesar do avanço contínuo das tecnologias de, hardware e software e do desenvolvimento contínuo de novos paradigmas para as especificações de dados e processos.

Este capítulo define a noção de computação, discute os conceitos de hardware e software e conclui com uma introdução ao desenvolvimento de software.

O resto do texto centra-se no desenvolvimento de software para computadores, fornecendo uma discussão pormenorizada dos princípios do software, bem como um retrato da cultura atual do campo do desenvolvimento de software.

# Uma breve história do computador

O termo **Computador** foi introduzido pela primeira vez em 1640 e referido como 'aquele que calcula'. Foi derivado da palavra latina '**computare**', que significa 'calcular'. Em 1897, era conhecido como 'máquina de calcular'. Mais tarde, em 1945, o termo 'computador' foi introduzido como computador eletrônico digital programável, que agora é chamado de computador.

Quando os computadores foram introduzidos, eles eram grandes e podiam preencher uma sala inteira. Alguns computadores eram operados usando tubos de vácuo de grande porte. Em 1833, Charles Babbage (conhecido como o pai do computador) inventou uma calculadora antiga, que foi nomeada como mecanismo de diferença. Mais tarde, em 1837, ele introduziu o primeiro computador mecânico de uso geral, **Mecanismo Analítico**.

Com o tempo, os computadores se tornaram poderosos em desempenho e pequenos em tamanho.

# Geração de Computadores

Existem cinco gerações de computadores, que podem ser classificadas da seguinte forma:

**Primeira Geração (1946 - 1959)**: Tubos a vácuo - Durante a primeira geração, os computadores eram baseados em válvulas eletrônicas (Vacuum Tubes). Alguns computadores populares da primeira geração são ENIAC, EDVAC, UNIVAC, etc.

**Segunda Geração (1959 - 1965)**: Transistores - Durante a segunda geração, os computadores eram baseados em Transistores. Alguns computadores populares de segunda geração são IBM 1400, IBM 1620, IBM série 7000, etc.

**Terceira Geração (1965 - 1971)**: Circuitos Integrados - Durante a terceira geração, os computadores eram baseados em Circuitos Integrados (CIs). Alguns computadores populares da terceira geração são IBM 360, IBM 370, PDP, etc.

**Quarta Geração (1971 - 1980)**: Microprocessadores - Durante a quarta geração, os computadores eram baseados em circuitos integrados de escala muito grande (VLSI). Alguns computadores populares de quarta geração são STAR 1000, CRAY-1, CRAY-X-MP, DEC 10, etc.

**Quinta Geração (1980 - Presente)**: Inteligência Artificial e Computação Paralela - A quinta geração caracteriza-se pelo uso de tecnologias avançadas, como IA (inteligência artificial), sistemas especializados e arquitetura de computação paralela e distribuída.

Os computadores atuais utilizam redes neurais, aprendizado de máquina e algoritmos avançados, o que possibilita desde automação em larga escala até processamento de grandes volumes de dados (Big Data).

Computação em nuvem: Com a internet, surgiu a computação em nuvem, que permite o processamento remoto de dados e o armazenamento virtual, facilitando o acesso a recursos e serviços web.

# Hardware e Software

O software de computador é uma coleção de muitas instruções diferentes que executam tarefas no seu computador. O software também é descrito como código de programação utilizado pelo processador do computador. O computador precisa de software como o sistema operativo para realizar as tarefas básicas, mas também existem muitos outros tipos de software.

O poder aritmético que esse hardware fornece é inútil, a menos que possa ser posto ao serviço efetuando cálculos úteis em sequências corretas envolvendo números significativos. É o software de computador que fornece essa direção significativa e útil. De facto, foi o surgimento do software que permitiu aos computadores que permitiu que os computadores evoluíssem de meros calculadores de números para tecnologias que atualmente enriquecem tantas áreas da vida humana.

Consideremos a analogia do cálculo dos impostos. Uma calculadora pode certamente ser útil neste processo, acelerando e melhorando a precisão da aritmética envolvida. No entanto, uma calculadora não pode calcular os impostos por si. É o formulário fiscal que especifica quais as operações aritméticas a efetuar, em que ordem e com que aritméticas devem ser efetuadas, em que ordem e com que números. Neste sentido, um formulário de imposto tem muito em comum com uma calculadora. programa de computador, que é também uma sequência definida de ações envolvendo informação correta que, quando que, quando executado, produz um resultado desejado. Por exemplo, no caso do software fiscal que está atualmente disponível hoje em dia, o programa de computador é modelado de acordo com o programa de ação humana que é prescrito pela forma.

Charles Babbage, já identificado como uma figura-chave na história do hardware de computador, é também uma figura-chave na história do

software. Depois de ter desistido do “Motor de Diferenças”, Babbage começou a trabalhar numa máquina muito mais sofisticada a que chamou O funcionamento desta máquina deveria ser muito mais versátil e automático O funcionamento desta máquina deveria ser muito mais versátil e automático do que a sua invenção anterior. Em termos de hardware, Babbage concebeu uma máquina construída para efetuar as operações básicas da aritmética sobre dígitos numéricos - ou seja, uma calculadora. No entanto, tomando de empréstimo uma tecnologia da máquina automática teares “Jacquard” automatizados que começaram a aparecer no início do século XIX, Babbage planeou alimentar o seu Motor Analítico com sequências de cartões de metal com furos perfurados. Em vez de serem utilizados para definir uma sequência de fios a incorporar numa determinada trama, os de fios a incorporar numa determinada trama, os cartões perfurados seriam usados para definir uma sequência de operações aritméticas para o Motor Analítico efetuar, que, em conjunto, atingiam um resultado matemático desejado. Por outras palavras, ao contrário das máquinas de calcular anteriores, o Motor Analítico seria programável: tal como um único tear automático podia efetuar diferentes tecelagens simplesmente trocando conjuntos de cartões perfurados, também o Motor Analítico de Babbage de Babbage seria capaz de alternar entre diferentes cálculos matemáticos simplesmente mudando o conjunto de cartões perfurados. De uma forma notável, o Motor Analítico antecipou a “arquitetura” fundamental do computador eletrônico moderno, na medida em que estava organizado em quatro subsistemas, subsistemas primários de processamento, armazenamento, entrada e saída.

Ada Lovelace, filha de Lord Byron, foi uma das poucas pessoas, para além de Babbage, que compreendeu o enorme potencial do Motor Analítico.

Ela descreveu a semelhança entre as invenções de Jacquard e de Babbage: O motor analítico tece padrões algébricos tal como o tear de Jacquard tecer flores e folhas, em ambos os casos, simplesmente executando uma sequência cuidadosamente sequência de operações básicas. Lovelace concebeu e escreveu demonstrações de como cálculos matemáticos complexos podiam ser construídos inteiramente a

partir de sequências do conjunto básico de operações aritméticas de que o Motor Analítico seria capaz. analítico seria capaz. Ada Lovelace é frequentemente considerada como “a primeira programadora”, e o seu trabalho tem certamente muito que recomenda este título para ela - ainda mais, de facto, do que é normalmente reconhecido.

Nos seus escritos, Lovelace identificou que uma das principais características de um programa de computador é a sua natureza cuidadosamente sequencial. Os matemáticos usam frequentemente o termo “algoritmo” para se referirem a uma sequência específica de operações que, se executadas, produzirão um determinado resultado desejado. Por conseguinte, uma das principais atividades-chave da programação de computadores é muitas vezes designada por “concessão de algoritmos”, em que um determinado processo é decomposto com sucesso numa sequência que é composta inteiramente por operações que um computador é capaz de efetuar.

# Hardware

As partes físicas anexadas a um computador que formam um computador inteiro são chamadas de hardware ou componentes de hardware. Pode haver diferentes tipos de hardware, dependendo da estrutura. Alguns hardwares mais comuns são mouse, teclado, monitor, impressora, etc. Essas são as partes que podem ser vistas e tocadas por humanos.

# Partes básicas do computador

Os componentes essenciais do computador podem ser definidos da seguinte forma:

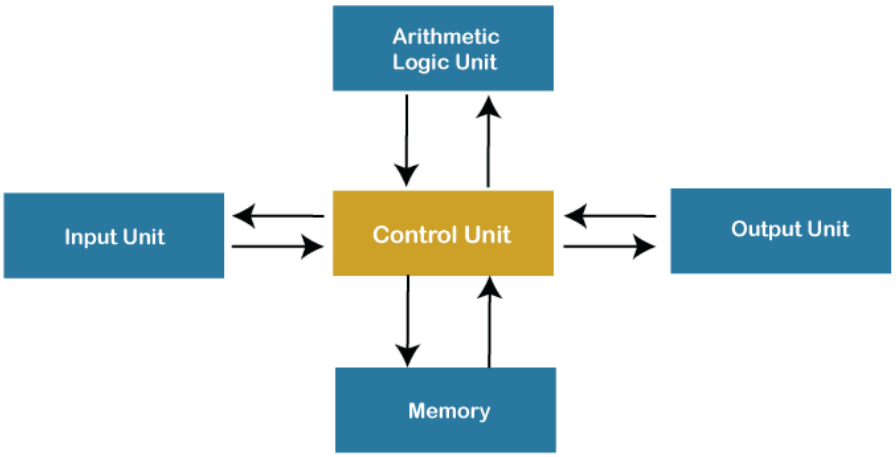
**Unidade de Entrada**: Unidades ou dispositivos de entrada são usados para inserir dados ou instruções nos computadores. Alguns dispositivos de entrada mais comuns são mouse e palavras-chave.

**Unidades de saída**: Unidades ou dispositivos de saída são usados para fornecer saída ao usuário no formato desejado. Os exemplos mais populares de dispositivos de saída são o monitor e a impressora.

**Unidade de Controle**: Como o próprio nome diz, esta unidade é usada principalmente para controlar todas as funções e funcionalidades do computador. Todos os componentes ou dispositivos conectados a um computador interagem entre si por meio da unidade de controle. Em resumo, a unidade de controle é chamada de ' CU '.

**Unidade Lógica Aritmética**: A unidade lógica aritmética ajuda a executar todas as operações lógicas e aritméticas do sistema de computador. Em resumo, a unidade lógica aritmética é chamada de ' ALU '.

**Memória**: A memória é usada para armazenar todos os dados de entrada, instruções e dados de saída. A memória geralmente tem dois tipos: Memória Primária e Memória Secundária. A memória encontrada dentro da CPU é chamada de memória primária, enquanto a memória que não é parte integrante da CPU é chamada de memória secundária.



# Diferenças entre Hardware vs Software

Dê uma olhada na tabela abaixo, que explica qual é a diferença entre hardware e software.

| Hardware | Software |
| --- | --- |
| Diferentes tipos de hardware de computador podem ser divididos em quatro categorias principais: dispositivos de entrada, dispositivos de saída, armazenamento e componentes internos. | O software de computador é dividido em duas categorias principais: software de sistema e software de aplicativo. |
| O hardware não é afetado por vírus de computador, pois o hardware é um objeto físico. | O software pode ser afetado por vírus de computador, que podem interromper sistemas, vazar dados e causar grandes problemas operacionais. |
| O hardware não pode ser transferido eletronicamente, mas se as peças forem compatíveis, o hardware pode ser usado em diferentes computadores. | O software pode ser transferido eletronicamente por meio de um sistema de rede. Dependendo de qual software está sendo transferido, você pode usar uma unidade USB para transferi-lo. Alguns softwares precisam ser reinstalados em cada dispositivo. |
| Como o hardware do computador é feito de objetos físicos, ele pode se desgastar. Essas peças podem ser substituídas. | O software pode desenvolver bugs e falhas ao longo do tempo, mas cópias de segurança podem ser usadas para substituí-lo. |
| Falhas de hardware podem ser causadas por poeira, superaquecimento e fornecimento de energia desregulado. | Falhas de software podem ser causadas por sobrecarga, erros de sistema e erros de versão. |

# Classificação dos computadores

De acordo com o tamanho físico, os computadores são classificados nos seguintes tipos:

**Supercomputador**: Os supercomputadores são o tipo de computador mais rápido e mais caro. São de grandes dimensões e requerem mais espaço para serem instalados. Estes tipos de computadores são principalmente concebidos para executar tarefas complexas e baseadas em dados maciços. Os supercomputadores são capazes de processar trilhões de instruções ao mesmo tempo.

**Mainframe**: Os computadores mainframe são comparativamente mais pequenos em relação aos supercomputadores. No entanto, não são muito pequenos. Estes tipos de computadores são concebidos para executar centenas ou milhares de tarefas em simultâneo. Estes computadores podem efetuar tarefas pesadas, incluindo cálculos complexos, e armazenar grandes quantidades de dados. São mais adequados para grandes organizações, como a banca, as telecomunicações e os sectores da educação.

**Microcomputador**: Os microcomputadores são baratos e suportam uma plataforma multi-utilizador. São computadores de uso geral, concebidos para realizar todas as tarefas necessárias às necessidades individuais. Uma vez que são comparativamente mais lentos do que os computadores mainframe, são adequados para pequenas organizações. São mais adequados para cibercafés, escolas, universidades, escritórios, etc. Um microcomputador é também designado por "computador pessoal (PC)" na vida quotidiana. O computador portátil e o computador de secretária são exemplos de microcomputadores.

**Minicomputador**: Os minicomputadores são também designados por computadores Miniframe. Trata-se de um computador multiprocessador de média dimensão concebido propositadamente para ser fácil de transportar. Estes tipos de computadores são leves e cabem num espaço pequeno. São adequados para fins de faturação, contabilidade, educação e negócios.

**Workstation**: A estação de trabalho é um computador potente e de utilizador único. Uma estação de trabalho é um computador pessoal com um microprocessador mais rápido, uma enorme quantidade de RAM, monitores de maior qualidade, memória gráfica elevada, etc. É o computador mais adequado para realizar um determinado tipo de tarefas a nível profissional. De acordo com o tipo de tarefas, uma estação de trabalho pode ser designada por estação de trabalho de música, estação de trabalho gráfica ou estação de trabalho de projeto de engenharia. A maioria das empresas e dos profissionais utilizam estações de trabalho para executar tarefas como animação, criação de música, edição de vídeo, desenho de cartazes, análise de dados e muito mais.

# Computação Pessoal, Computação Distribuída e Computação Cliente/Servidor

Em 1997, a Apple Computer tornou popular o fenômeno da computação pessoal. Inicialmente, isto era um sonho de quem a tinha como hobby. Computadores tornaram-se suficientemente baratos para serem comprados para uso pessoal ou comercial. Em 1981, a IBM, a maior vendedora de computadores do mundo, criou o IBM PC (Personal Computer, computador pessoal). Do dia para a noite, literalmente, a

computação pessoal se tornou comum no comércio, na indústria e em organizações governamentais.

Mas esses computadores eram unidades "autônomas" — as pessoas faziam suas tarefas em seus próprios equipamentos e então transportavam os discos de um lado para outro para compartilhar as informações. Embora os primeiros computadores pessoais não fossem suficientemente poderosos para serem compartilhados por vários usuários, esses equipamentos podiam ser ligados entre si em redes de computadores, algumas vezes através de linhas telefônicas e algumas vezes em redes locais de organizações.

Isto levou ao fenômeno da computação distribuída, na qual a carga de trabalho computacional de uma organização, em vez de ser realizada exclusivamente em uma instalação central de informática, é distribuída em redes para os locais (sites) nos quais o trabalho real da organização é efetuado. Os computadores pessoais eram suficientemente poderosos para manipular as exigências computacionais de cada usuário em particular e as tarefas básicas de comunicação de passar as informações eletronicamente de um lugar para outro.

Os computadores pessoais mais poderosos de hoje são tão poderosos quanto os equipamentos de milhões de dólares de apenas uma década atrás. Os equipamentos desktop (computadores de mesa) mais poderosos — chamados workstations ou estações de trabalho —

fornecem capacidades enormes a usuários isolados. As informações são compartilhadas facilmente em redes de computadores onde alguns deles, os chamados servidores de arquivos (file servers), oferecem um depósito comum de programas e dados que podem ser usados pelos computadores clientes (clientes) distribuídos ao longo da rede, daí o termo computação cliente/servidor. O C e o C++ tornaram-se as linguagens preferidas de programação para a criação de software destinado a sistemas operacionais, redes de computadores e aplicações distribuídas cliente/servidor.

# Lei de Moore

A Lei de Moore afirma que o número de transistores em um chip dobra aproximadamente a cada dois anos, o que tem impulsionado o aumento do poder de processamento ao longo do tempo. A lei, embora seja uma previsão, ajudou a orientar o ritmo do desenvolvimento tecnológico, mas enfrenta limitações físicas com os avanços em nanometrologia.

Gordon Moore, um dos co-fundadores da Intel Corporation, estabeleceu o termo “Lei de Moore” em 1965. Esta lei explica como o número de transistores em circuitos integrados está aumentando exponencialmente, o que aumenta a capacidade de computação e reduz os preços. A lei de Moore teve um impacto significativo e de longo alcance na tecnologia, desde a introdução de computadores pessoais e smartphones até o avanço da inteligência artificial e dispositivos de Internet das Coisas (IoT). Ela acelerou o desenvolvimento em setores como telecomunicações, saúde, transporte, permitindo que organizações e pessoas realizem coisas que antes eram impensáveis.

## Definição da Lei de Moore

O aumento exponencial no número de transistores em circuitos integrados ao longo do tempo é chamado de lei de Moore. De acordo com isso, a contagem de transistores de um chip tende a dobrar a cada dois anos ou mais, resultando em maior poder de processamento e melhor desempenho.

## Escala de transístor

O dimensionamento de transistores, um componente crucial da lei de Moore, provou ser essencial no avanço da tecnologia em muitos campos diferentes. Os transístores agem como interruptores que regulam o fluxo de corrente elétrica dentro dos microchips, servindo como blocos de construção essenciais de dispositivos eletrônicos.

À medida que a tecnologia se desenvolveu, os engenheiros se concentraram em encolher e empacotar os transístores mais próximos uns dos outros em um chip. Um grande aumento no número de transistores que poderiam ser colocados em um único chip foi possível por esse processo de dimensionamento. Os dispositivos eletrônicos foram capazes de concluir tarefas mais complicadas mais rapidamente devido à maior capacidade de computação trazida pela adição de transístores.

## Impacto no poder de computação

A lei de Moore teve nada menos que um efeito transformador no poder da computação. O poder de computação de dispositivos eletrônicos aumentou exponencialmente com cada duplicação do número de transistores em um chip. A criação de aplicativos de software sofisticados, capacidades de processamento de dados e uma ampla gama de avanços tecnológicos foram todos estimulados por esse crescimento exponencial. Os computadores agora podem processar mais operações por segundo devido ao crescimento constante na densidade de transistores, tornando a computação mais rápida e eficaz. Isso tornou possível desenvolver programas de software complexos.

## Desafios e Limitações

À medida que os transístores se aproximam das escalas atômicas, a Lei de Moore recentemente encontrou dificuldades e restrições consideráveis. O avanço contínuo da lei de Moore é dificultado por uma série de restrições físicas à medida que o tamanho do transístor diminui. Os efeitos quânticos em escalas tão pequenas são uma dessas restrições. Quando os elétrons vazam por barreiras como resultado do tunelamento quântico, por exemplo, as ações do transístor se tornam imprecisas e instáveis.

À medida que os transistores ficam menores, essa tendência fica mais forte, tornando mais desafiador manter um comportamento estável e previsível.

# A Lei de Moore está finalmente chegando ao fim?

A desaceleração da lei de Moore levou muitos a perguntarem: "A lei de Moore está finalmente acabando?" Isso, de fato, não está ocorrendo. Embora a lei de Moore ainda esteja entregando melhorias exponenciais, os resultados estão sendo alcançados em um ritmo mais lento. No entanto, o ritmo da inovação tecnológica NÃO está diminuindo. Em vez disso, a explosão de Hiper conectividade, big data e aplicativos de inteligência artificial aumentou o ritmo da inovação e a necessidade de melhorias no "estilo da lei de Moore" na tecnologia entregue.

Por muitos anos, a complexidade de escala impulsionou a lei de Moore e o crescimento exponencial da tecnologia da indústria de semicondutores. À medida que a capacidade de escalar um único chip diminui, a indústria está encontrando outros métodos de inovação para manter o crescimento exponencial, garantindo que os avanços tecnológicos continuem a progredir e a atender às demandas do futuro.

# A Internet

A internet é o conjunto de redes de computadores que, espalhados por todas as regiões do planeta, conseguem trocar dados e mensagens utilizando um protocolo comum.

Este protocolo compartilhado pela internet é capaz de unir vários usuários particulares, entidades públicas e empresariais em um mesmo acesso.

Ela então é formada por computadores comuns e por outros especiais, chamados de servidores, máquinas com grande poder de processamento e conexões velozes. Os servidores são controlados por universidades, empresas e órgãos do governo.

## História da internet

Ainda que a internet tenha uma história relativamente breve, ela se revela como um grande fator de comunicação e integração social. É hoje um grande repositório de armazenamento de informações de todos os tipos, além da globalização de produtos.

A origem da internet é datada pelos idos de 1960, quando os Estados Unidos encomendaram uma pesquisa para construir uma forma de comunicação robusta e sem falhas através de redes de computadores.

Embora este trabalho, juntamente com projetos no Reino Unido e na França, tenha levado a criação de redes precursoras importantes, ele não criou a internet de fato. Entretanto, ela surgiu oficialmente em 1993, onde deixou de ser utilizada apenas por governos e de natureza acadêmica, e passou a estar presente nos diversos segmentos de empresas, residências, etc. As conexões para acessar as internets também evoluíram muito com o passar dos anos, tornando-a cada vez mais rápidas e práticas.

No período entre a década de 1990 e o início dos anos 2000, notou-se um rápido crescimento do uso da internet, por conta das inovações tecnológicas que foram sendo criadas a partir dela.

# Rede de computadores

Rede, ou rede de computadores, é o processo de conectar dois ou mais dispositivos de computação, como computadores de mesa, dispositivos móveis, roteadores ou aplicações, para permitir a transmissão e troca de informações e recursos.

Os dispositivos em rede dependem de protocolos de comunicação, regras que descrevem como transmitir ou trocar dados em uma rede, para compartilhar informações por meio de conexões físicas ou sem fio.

## Principais componentes e dispositivos de rede

Antes de nos aprofundarmos em tópicos de rede mais complexos, é importante entender os componentes fundamentais de rede, incluindo:

**Endereço IP**: um endereço IP é o número exclusivo atribuído a cada dispositivo de rede em uma rede de Internet Protocol (IP); cada endereço IP identifica a rede de host do dispositivo e sua localização na rede.

**Nós**: Um nó é um ponto de conexão de rede que pode receber, enviar, criar ou armazenar dados. É essencialmente qualquer dispositivo de rede - computadores, impressoras, modems, pontes ou switches - que pode reconhecer, processar e transmitir informações para outro nó de rede.

**Roteadores**: Um roteador é um dispositivo físico ou virtual que envia “pacotes” de dados entre redes. Os roteadores analisam os dados dentro dos pacotes para determinar o melhor caminho de transmissão e usam algoritmos de roteamento sofisticados para encaminhar pacotes de dados até que eles cheguem ao nó de destino.

**Switches**: Um switch é um dispositivo que conecta dispositivos de rede e gerencia a comunicação nó a nó em uma rede, garantindo que os pacotes de dados cheguem ao destino pretendido. Ao contrário dos roteadores, que enviam informações entre redes, os switches enviam informações entre os nós de uma rede.

Portanto, "comutação" se refere a como os dados são transferidos entre dispositivos em uma rede. As redes dependem de três tipos principais de comutação:

- A comutação de circuitos estabelece um caminho de comunicação de dados dedicado entre nós em uma rede, de modo que nenhum outro tráfego possa percorrer o mesmo caminho. A comutação de circuitos garante que a largura de banda total esteja disponível durante cada transmissão.

- A comutação de mensagens envia mensagens inteiras do nó de origem para o nó de destino, com a mensagem viajando de comutador para comutador até alcançar o destino.

- Comutação de pacotes envolve a fragmentação dos dados em componentes independentes para tornar a transmissão de dados menos exigente em relação aos recursos da rede. Com o encaminhamento de pacotes, os pacotes e não as streams de dados completos viajam pela rede até seu destino final.

**Portas**: uma porta indica uma conexão específica entre dispositivos de rede, com cada porta identificada por um número. Se um endereço IP for análogo ao endereço de um hotel, as portas serão as suítes e os números dos quartos.

**Gateways**: os gateways são dispositivos de hardware que facilitam a comunicação entre duas redes diferentes. Roteadores, firewalls e outros dispositivos de gateway utilizam transformadores de taxa.

## Tipos de redes de computadores

Normalmente, as redes de computadores são definidas por área geográfica. Uma rede de área local (LAN) conecta computadores em um espaço físico definido, enquanto uma rede de área ampla (WAN) pode conectar computadores entre continentes. No entanto, as redes também são definidas pelos protocolos que utilizam para se comunicar, pela disposição física de seus componentes.

## Tipos de rede por área geográfica

Os tipos de rede nesta categoria são diferenciados pela área geográfica que a rede cobre.

### Rede de área local (LAN)

Uma LAN conecta computadores a uma distância relativamente curta, como aqueles dentro de um edifício de escritório, escola ou hospital. LANs são normalmente de propriedade privada e gerenciadas.

### Rede de longa distância (WAN)

Como o nome implica, uma WAN conecta computadores em grandes áreas geográficas, como regiões e continentes. As WANs geralmente têm modelos de propriedade coletivos ou distribuídos para fins de gerenciamento de rede. As redes em nuvem servem como um exemplo, já que são hospedadas e entregues por infraestruturas de nuvem públicas e privadas em todo o mundo.

### Rede de área metropolitana (MAN)

As MANs são maiores que as LANs, mas menores que as WANs. Cidades e entidades governamentais normalmente possuem e gerenciam MANs.

### Rede de área pessoal (PAN)

Um PAN atende a uma pessoa. Se um usuário tiver vários dispositivos do mesmo fabricante (como um iPhone e um MacBook, por exemplo), é provável que tenha configurado um PAN que compartilhe e sincronize conteúdo, como mensagens de texto, e-mails, fotos e muito mais, em todos os dispositivos.

## Tipos de rede por meio de transmissão

Os nós de rede podem enviar e receber mensagens usando links com fio ou sem fio (conexões).

### Redes com fio

Os dispositivos de rede com fio são conectados por fios e cabos físicos, incluindo fios de cobre e cabos Ethernet, par entrelaçados, coaxiais ou de fibra óptica. O tamanho e a velocidade da rede geralmente determinam a escolha do cabo, a disposição dos elementos da rede e a distância física entre os dispositivos.

### Redes sem fio

As redes sem fio dispensam cabos para transmissão de infravermelho, rádio ou ondas eletromagnéticas em dispositivos sem fio com antenas e sensores integrados.

## Tipos de rede por tipo de comunicação

As redes de computação podem transmitir dados usando uma variedade de dinâmicas de transmissão, incluindo:

- Redes multiponto, em uma rede multiponto, vários dispositivos compartilham a capacidade do canal e os links de rede.
- Redes ponto a ponto, os dispositivos de rede estabelecem um link direto de nó a nó para transmitir dados.

### Redes de transmissão

Em redes de transmissão, várias "partes" (dispositivos) interessados podem receber transmissões unidirecionais de um único dispositivo de envio. As estações de televisão são um ótimo exemplo de redes de transmissão.

### Redes privadas virtuais (VPNs)

Uma VPN é uma conexão segura, de ponto a ponto, entre dois endpoints da rede. Ele estabelece um canal criptografado que mantém a identidade e as credenciais de acesso de um usuário, bem como todos os dados transferidos, inacessíveis aos hackers.

# Arquitetura de rede

A arquitetura de rede de computadores estabelece o framework teórico de uma rede de computadores, incluindo princípios de design e protocolos de comunicação.

## Tipos principais de arquiteturas de rede

**Arquiteturas ponto a ponto (P2P)**: em uma arquitetura P2P, dois ou mais computadores são conectados como “pares”, o que significa que eles têm poder e privilégios iguais na rede. Uma rede P2P não requer um servidor central para coordenação. Em vez disso, cada computador na rede atua como um cliente (um computador que precisa acessar um serviço) e um servidor (um computador que fornece serviços aos clientes). Cada peer na rede disponibiliza alguns de seus recursos para outros dispositivos de rede, compartilhando armazenamento, memória, largura de banda e poder de processamento em toda a rede.

**Arquiteturas cliente-servidor**: Em uma rede cliente-servidor, um servidor central (ou grupo de servidores) gerencia recursos e fornece serviços a dispositivos clientes na rede; nesta arquitetura, os clientes não compartilham seus recursos e só interagem por meio do servidor. As arquiteturas cliente-servidor são frequentemente chamadas de arquiteturas em camadas devido às suas múltiplas camadas.

**Arquitecturas híbridas**: Arquitecturas híbridas incorporan elementos de modelos P2P e cliente-servidor.

## Topologia de rede

Enquanto a arquitetura representa a framework teórica de uma rede, a topologia é a implementação prática da framework arquitetónica. A topologia de rede descreve a disposição física e lógica dos nós e links em uma rede, incluindo todo o hardware (roteadores, switches, cabos), software (aplicações e sistemas operacionais) e meios de transmissão (conexões com ou sem fio).

# Protocolos de comunicação de rede

Seja o internet protocol (IP), Ethernet, LAN sem fio (WLAN) ou padrões de comunicação celular, todas as redes de computadores seguem protocolos de comunicação - conjuntos de regras que cada nó da rede deve seguir para compartilhar e receber dados. Os protocolos também dependem de gateways para permitir que dispositivos incompatíveis se comuniquem (um computador Windows tentando acessar servidores Linux, por exemplo).

## Camada de acesso à rede

Também chamada de camada de link de dados ou camada física, a camada de acesso à rede de uma rede TCP/IP inclui a infraestrutura de rede (componentes de hardware e software) necessárias para a interface com o meio de rede. Ela lida com a transmissão física de dados — usando Ethernet e protocolos como o protocolo de resolução de endereços (ARP)

## Camada de Internet

A camada de Internet é responsável pelo endereçamento lógico, roteamento e encaminhamento de pacotes. Ele depende principalmente do protocolo IP e do Internet Control Message Protocol (ICMP), que gerencia o endereçamento e o roteamento de pacotes em diferentes redes.

## Camada de transporte

A camada de transporte TCP/IP permite a transferência de dados entre as camadas superior e inferior da rede. Utilizando protocolos TCP e UDP, também fornece mecanismos para verificação de erros e controle de fluxo.

O TCP é um protocolo baseado em conexão geralmente mais lento, porém mais confiável do que o UDP.

O UDP é um protocolo sem conexão mais rápido que o TCP, mas não oferece transferência garantida.

## Camada de aplicativo

A camada de aplicação do TCP/IP utiliza HTTP, FTP, Post Office Protocol 3 (POP3), SMTP, sistema de nomes de domínio (DNS) e SSH para fornecer serviços de rede diretamente às aplicações.

Embora o TCP/IP seja mais diretamente aplicável à rede, o modelo OSI (Open Systems Interconnection, Interconexão de Sistemas Abertos), às vezes chamado de modelo de referência OSI, também teve um impacto significativo na rede de computadores e na ciência da computação, de modo geral.

O OSI é um modelo conceitual que divide a comunicação em rede em sete camadas abstratas (em vez de quatro), fornecendo uma base teórica que ajuda engenheiros e desenvolvedores a entender as complexidades da comunicação em rede.

## Funcionamento da rede de um computador

Usando o e-mail como exemplo, vamos ver um exemplo de como os dados são migrados através de uma rede.

Se um usuário quiser enviar um e-mail, ele primeiro escreve o e-mail e depois pressiona o botão "enviar". Quando o usuário pressiona “enviar”, um protocolo SMTP ou POP3 usa o wi-fi do remetente para direcionar a mensagem do nó do remetente e por meio dos comutadores de rede, onde ela é comprimida e dividida em segmentos cada vez menores (e, por fim, em bits ou strings de 1s e 0s).

Os gateways de rede direcionam o fluxo de bits para a rede do destinatário, convertendo dados e protocolos de comunicação conforme necessário. Quando o fluxo de bits chega ao computador do destinatário, os mesmos protocolos direcionam os dados de e-mail por meio dos comutadores de rede na rede do destinatário.

# A World Wide Web

World Wide Web, o famoso WWW, é um sistema de documentos dispostos na Internet que permitem o acesso às informações apresentadas no formato de hipertexto. Para ter acesso a tais informações pode-se usar um programa de computador chamado navegador. Os navegadores mais famosos são: Internet Explorer, Mozilla Firefox, Google Chrome e Safari.

A Web funciona através de três parâmetros:

**URL**, que especifica o endereço único que cada página vai receber, e é como ela vai ser encontrada quando os usuários digitarem;

**HTTP**, que é um protocolo de comunicação que permite a transferência de informação entre redes;

**HTML**, que é um método de codificar a informação da internet, para ser exibida de diversas maneiras.

## História da sigla www

A internet nasceu como uma rede fechada, nos anos 60, com o nome de Arpanet. Criada em laboratórios militares dos Estados Unidos, servia para trocar informações entre computadores do governo. Só em 1989 a proposta ganhou a característica que conhecemos hoje com o surgimento do www (World Wide Web).

Desenvolvido pelo físico inglês Tim Berners-Lee, nos laboratórios da Organização Europeia para a Pesquisa Nuclear (CERN).

O www estabeleceu uma linguagem padrão para a circulação de dados na rede, permitindo que qualquer computador, de qualquer parte do planeta, tivesse livre acesso ao mundo virtual.

# Serviços Web

Um serviço Web é um conjunto de funções de aplicação relacionadas que podem ser invocadas programaticamente na Internet. As empresas podem misturar e corresponder dinamicamente os serviços Web para executar transações complexas com uma programação mínima.

Os serviços Web permitem que compradores e vendedores em todo o mundo se identifiquem mutuamente, contactem de forma dinâmica e executem transações em tempo real com a mínima interação humana.

## Os serviços Web são independentes

No lado do cliente, não é necessário nenhum software adicional. Uma linguagem de programação com o suporte do cliente XML e HTTP é suficiente para poder começar. No lado do servidor, são necessários um servidor da Web e um motor servlet.

O cliente e o servidor podem ser implementados em ambientes diferentes, o servidor da Web pode ativar uma aplicação existente sem gravar uma única linha de código. Os serviços Web descrevem-se a si próprios.

O cliente e o servidor necessitam de reconhecer apenas o formato e o conteúdo de mensagens de pedido e de resposta. A definição do formato da mensagem desloca-se com a mensagem.

## Os serviços Web são modulares

Os serviços Web simples podem ser agrupados para formar serviços Web mais complexos, utilizando técnicas de fluxo de trabalho ou chamando serviços Web de nível mais baixo a partir de uma implementação de serviço Web.

Os serviços Web podem ser qualquer coisa, como por exemplo, artigos de crítica de teatro, boletins meteorológicos, análises de crédito, cotações da bolsa, avisos de viagens ou processos de reserva de viagens de avião.

# Categorias de serviços Web

Os serviços Web podem ser agrupados em três categorias:

## Informações sobre a empresa

Uma empresa partilha informações com os consumidores e com outras empresas. Neste caso, a empresa está a utilizar serviços Web para expandir o seu âmbito. Exemplos de serviços Web de informações de empresas são notícias, boletins meteorológicos ou cotações da bolsa. Integração de empresas.

Uma empresa fornece, aos seus clientes, serviços transacionais com comissões. Neste caso, o negócio torna-se parte de uma rede global de fornecedores de valor adicionado que podem ser utilizados para proceder ao comércio. Alguns exemplos de serviços Web de integração de empresas incluem e-Marketplace de licitações e leilões, sistemas de reservas e verificações de crédito. Exteriorização do processo empresarial.

Um negócio distingue-se da competição através da criação de uma cadeia de valores global. Neste caso, a empresa utiliza serviços Web para integrar os seus processos dinamicamente. Um exemplo de serviços Web de processo de exteriorização de uma empresa é a associação entre empresas diferentes de forma a combinar o fabrico, a montagem, a distribuição por grosso e a venda a retalho de um determinado produto.

## Funções e interações de serviços

Uma componente de rede numa arquitetura de serviços Web pode desempenhar um ou mais papeis fundamentais: fornecedor de serviços, corretor de serviços e cliente de serviços.

Os fornecedores de serviços criam e implementam os seus serviços Web e podem publicar a disponibilidade dos seus serviços descritos pelo WSDL através de um registo de serviços, tal como o Registo de Negócio UDDI.

# Tecnologias de Software

O desenvolvimento de produtos digitais está entre as áreas mais dinâmicas e em transformação da atualidade. Para atender ao nível de complexidade técnica e exigência dos usuários, as tecnologias usadas para construção de aplicativos, sistemas e plataformas precisam evoluir constantemente.

Alguns jargões que você ouvirá na comunidade de desenvolvimento de softwares.

## Desenvolvimento ágil de software

Desenvolvimento ágil de software é um conjunto de metodologias que tentam fazer um software ser implementado mais rápido e usando menos recursos.

## Refatoração

Refatoração envolve retrabalhar os programas para torná-los mais claros e fáceis de manter e, ao mesmo tempo, preservar sua exatidão e funcionalidade. Ela é amplamente empregue com metodologias de desenvolvimento ágeis. Muitos IDEs contêm ferramentas de refatoração embutidas para fazer as principais partes do retrabalho automaticamente.

## Padrões de design

Padrões de design são arquiteturas testadas para construir softwares orientados a objetos flexíveis e que podem ser mantidos. O campo dos padrões de design tenta enumerar aqueles padrões recorrentes, encorajando os designers de software a utilizá-los a fim de desenvolver

softwares de melhor qualidade empregando menos tempo, dinheiro e esforço.

## LAMP

LAMP é um acrônimo para as tecnologias de código-fonte aberto que muitos desenvolvedores usam a fim de construir aplicativos web — ele significa Linux, Apache, MySQL e PHP (ou Perl ou Python — duas outras linguagens de script).

O MySQL é um sistema de gerenciamento de bancos de dados de código-fonte aberto. PHP é a linguagem de “script” mais popular de código-fonte aberto no lado do servidor para a criação de aplicativos web. Apache é o software de servidor web mais adotado. O equivalente para o desenvolvimento Windows é WAMP - Windows, Apache, MySQL e PHP.

## Software como serviço (SaaS)

Softwares geralmente são vistos como um produto; a maioria deles ainda é oferecida dessa forma. Se quiser executar um aplicativo, você compra um pacote de software a partir de um fornecedor — com frequência um CD, DVD ou download da web. Você então instala esse software no computador e o executa conforme necessário.

À medida que aparecem novas versões, você atualiza seu software, muitas vezes com um custo considerável em termos de tempo e dinheiro. Esse processo pode se tornar complicado para as organizações que devem manter dezenas de milhares de sistemas em um array diverso de equipamentos de computador.

Com o Software como serviço (Software as a service — SaaS), os softwares executam em servidores em outros locais na internet. Quando esse servidor é atualizado, todos os clientes no mundo inteiro veem novas capacidades — nenhuma instalação local é necessária.

Você acessa o serviço por um navegador. Navegadores são bem portáteis; portanto, você pode executar os mesmos aplicativos em uma ampla variedade de computadores a partir de qualquer lugar no mundo. Salesforce.com, Google e o Office Live e Windows Live da Microsoft oferecem SaaS.

## Plataforma como serviço (PaaS)

Plataforma como serviço (Platform as a service — PaaS) fornece uma plataforma de computação para desenvolver e executar aplicativos como um serviço via web, em vez de instalar as ferramentas no seu computador. Alguns provedores de PaaS são o Google App Engine, Amazon EC2 e Windows Azure™.

## Computação em nuvem

SaaS e PaaS são exemplos da computação em nuvem. Você pode usar o software e os dados armazenados na “nuvem” — isto é, acessados em computadores remotos (ou servidores) via internet e disponíveis sob demanda — em vez de salvá-los em seu desktop, notebook ou dispositivo móvel.

Isso permite aumentar ou diminuir os recursos de computação para atender às suas necessidades em um dado momento qualquer, o que é mais eficaz em termos de custos do que comprar hardware a fim de fornecer armazenamento suficiente e capacidade de processamento com o intuito de suprir as demandas ocasionais de pico.

## Kit de desenvolvimento de software (SDK)

Kits de desenvolvimento de software (SDKs) incluem as ferramentas e a documentação que os desenvolvedores usam para programar aplicativos. Por exemplo, você empregará o Java Development Kit (JDK) para criar e executar aplicativos Java.

## O software é complexo

O projeto e a implementação de grandes aplicativos de software do mundo real podem levar muitos meses ou mesmo anos. Quando grandes produtos de software estão em desenvolvimento, eles normalmente são disponibilizados para as comunidades de usuários como uma série de versões, cada uma mais completa e refinada que a anterior que são:

## Alfa

Software alfa é a primeira versão de um produto que ainda está em desenvolvimento ativo.

Versões alfa muitas vezes são repletas de erros, incompletas e instáveis e são lançadas para um número relativamente pequeno de desenvolvedores a fim de testar novos recursos, receber feedback inicial etc.

## Beta

Versões beta são lançadas para um número maior de desenvolvedores no processo de desenvolvimento depois que a maioria dos principais erros foi corrigida e novos recursos estão quase completos. Software beta é mais estável, mas ainda está sujeito a alterações.

## Candidatos a lançamento

Candidatos a lançamento, ou release candidates, em geral têm todos os recursos, estão (principalmente) livres de erros e prontos para uso pela comunidade, que fornece um ambiente de teste diversificado — o software é empregado em sistemas distintos, com diferentes restrições e para uma variedade de propósitos.

## Versão final

Quaisquer erros que aparecem no candidato a lançamento são corrigidos e, com o tempo, o produto final é lançado para o público em geral. Empresas de software muitas vezes distribuem atualizações incrementais pela internet.

## Beta contínuo

Softwares desenvolvidos usando essa abordagem (por exemplo, pesquisa no Google ou Gmail) geralmente não têm números de versão. Esse tipo de software é hospedado na nuvem (não é instalado no seu computador) e está em constante evolução para que os usuários sempre tenham a versão mais recente.

# Sistemas Operacionais

Os sistemas operacionais (SO) são softwares que atuam como intermediários entre o hardware de um dispositivo e os programas que usamos diariamente, pense em um computador que possui diversos componentes. Sem um sistema, para emitir comandos aos dispositivos eletrônicos, precisaríamos conhecer a função de todos os componentes. Bastante burocrático, no mínimo, não é mesmo?

Em vista disso, o SO torna-se fundamental. Ele faz com que todos os componentes do computador trabalhem em harmonia.

## Para que serve o sistema operacional

O sistema operacional é essencial para qualquer dispositivo eletrônico, gerenciando todos os recursos e operações para garantir que tudo funcione como desejado pelo usuário, por meio de comandos.

Com a maioria dos sistemas operacionais disponíveis no mercado, é possível realizar funções diversas e básicas, como a gestão de arquivos, a execução de aplicativos e o uso de diferentes programas simultaneamente.

É importante destacar que o SO colabora ativamente para a boa experiência do usuário. A interação usuário-dispositivo torna-se muito mais acessível e intuitiva em razão da tecnologia. Portanto, por meio do sistema operacional, é possível acessar os dispositivos com uma interface gráfica amigável, que se dá em ícones, menus e janelas que facilitam o uso.

## Funções Básicas de um Sistema Operacional:

- **Gerenciamento de Processos**: O SO é responsável por criar, agendar e terminar processos. Ele mantém o controle de todos os processos em execução, aloca recursos para processos e facilita a comunicação e sincronização entre eles.

- **Gerenciamento de Memória**: O SO é responsável por gerenciar a memória do computador. Ele mantém o controle de cada byte na memória e decide quais processos receberão memória e quando.

- **Gerenciamento de Dispositivos**: O SO gerencia todos os dispositivos de hardware conectados ao sistema. Ele tem rotinas de software específicas, chamadas drivers de dispositivo, que interagem diretamente com o hardware do dispositivo.

- **Gerenciamento de Arquivos**: O SO é responsável por organizar arquivos em dispositivos de armazenamento, como discos rígidos. Ele controla quem pode acessar quais arquivos e como eles podem ser acessados.

- **Segurança e Acesso**: O SO é responsável por garantir que os recursos do sistema sejam usados apenas por aqueles que têm permissão para fazê-lo. Ele exige que os usuários façam login e podem proteger arquivos e diretórios individuais com senhas.

- **Interface do Usuário**: O SO fornece uma maneira para os usuários interagirem com o sistema. Isso pode ser uma interface de linha de comando (CLI), onde os usuários digitam comandos, ou uma interface gráfica do usuário (GUI), onde os usuários interagem com o sistema por meio de imagens e ícones.

- **Gerenciamento de Entrada/Saída (E/S)**: Controle de dispositivos de E/S, como discos rígidos, impressoras e teclados.

Estruturas de Sistemas Operacionais:

- **Monolítica**: Estrutura tradicional, onde todos os componentes do SO são agrupados em um único bloco.
- Em Camadas: Divide o SO em camadas, com cada camada fornecendo um serviço específico.

- **Micronúcleo**: Arquitetura com um núcleo mínimo que fornece serviços básicos, enquanto outros serviços são executados em processos separados.
- **Cliente-Servidor**: Divide o SO em dois componentes: cliente (interface com o usuário) e servidor (fornece serviços).

# Tipos de Sistemas Operacionais

Existem diversos tipos de sistemas operacionais no mercado, mas alguns são mais debatidos, os mais comuns são os SO de computador, usados em desktops e laptops, como o Windows, macOS e Linux.

Ainda, podemos citar os SO de dispositivos móveis, que são otimizados para telas sensíveis ao toque e eficiência de bateria.

Alguns exemplos são o iOS, usado nos dispositivos Apple e o Android, adotado por muitos fabricantes, além disso, ainda existem os sistemas operacionais de servidor, responsáveis por fornecer serviços e recursos a outros computadores em uma rede. Destacam-se o Windows Server, Linux e UNIX. Abaixo, separamos alguns detalhes sobre os quatro sistemas operacionais mais conhecidos.

## Linux

O Linux é um sistema operacional flexível, conhecido por sua estabilidade, natureza de código aberto e segurança.

Criado em 1991 por Linus Torvalds, o Linux é um SO que se tornou popular entre desenvolvedores e usuários mais assíduos de tecnologia, principalmente por ser de código aberto. Ou seja, com isso, qualquer pessoa pode acessar, modificar e distribuir o código-fonte.

Além disso, ele ainda é altamente personalizável. Com isso, é distribuído em variações como Ubuntu, Fedora e Debian, as quais podem ser ajustadas conforme as necessidades específicas dos usuários.

## Windows

O Windows é um dos sistemas operacionais mais populares e utilizados no mundo. Desenvolvido pela Microsoft e lançado em 1985, este SO destaca-se por apresentar interface gráfica amigável e acessível.

Também se destacando por ser compatível com uma grande quantidade de softwares e hardwares, sendo úteis para usuários comuns e empresas.

Por isso, em geral, é um sistema fácil de utilizar, com interface intuitiva e com inúmeras funcionalidades integradas, como o Microsoft Office e o Windows Defender.

## MacOS

O macOS, desenvolvido pela Apple, costuma ser o “queridinho” em termos de interface intuitiva e esteticamente agradável. Foi lançado em 2001 e tem como características a facilidade de uso, a segurança e a estabilidade de desempenho.

É um sistema que carrega uma interface fluida, uma barra de menus simplificada e atalhos que permitem aumentar a produtividade.

O macOS também é altamente integrado a outros dispositivos Apple, como iPhones, iPads e Apple Watches.

Inclusive, a segurança é mais um tópico de destaque aqui. Isso porque o macOS é atualizado regularmente com medidas de proteção avançadas.

## Android

O Android é mais um sistema operacional desenvolvido por uma gigante, o Google. É um sistema operacional móvel altamente popular, sendo aplicado em uma grande variedade de dispositivos, desde smartphones até TVs e smartwatches.

Lançado em 2008, o Android é valorizado pela flexibilidade, personalização e integração com os serviços do Google.

O Android, assim como o Linux, tem natureza de código aberto, facilitando a personalização e adaptação do sistema conforme as necessidades do usuário ou desenvolvedor.

## Explicação por analogia: hotel e o gerente do hotel

Imagine que um sistema operacional é como um hotel e o gerente do hotel é o núcleo do sistema operacional.

1. **Gerenciamento de Processos**: O gerente do hotel (núcleo) é responsável por verificar os hóspedes (processos), agendar suas atividades (CPU scheduling) e garantir que eles façam o check-out na hora certa (terminando processos).
2. **Gerenciamento de Memória**: O gerente do hotel também é responsável por atribuir quartos aos hóspedes. Ele mantém o controle de quais quartos estão ocupados e quais estão disponíveis, assim como um sistema operacional gerencia a memória do computador.
3. **Gerenciamento de Dispositivos**: O gerente do hotel garante que todos os serviços do hotel, como limpeza, lavanderia e serviço de quarto, estejam funcionando corretamente. Da mesma forma, um sistema operacional gerencia todos os dispositivos de hardware conectados ao sistema.

4. **Gerenciamento de Arquivos**: Assim como o gerente do hotel mantém um registo de todos os hóspedes e suas reservas, o sistema operacional é responsável por organizar arquivos em dispositivos de armazenamento.

5. **Segurança e Acesso**: O gerente do hotel garante que apenas os hóspedes com uma chave possam entrar em seus quartos, assim como o sistema operacional garante que os recursos do sistema sejam usados apenas por aqueles que têm permissão para fazê-lo.

6. **Interface do Usuário**: Finalmente, o gerente do hotel interage com os hóspedes e atende às suas necessidades, assim como o sistema operacional fornece uma maneira para os usuários interagirem com o sistema.

# Tecnologias Móveis

Tecnologia móvel é a tecnologia que acompanha o usuário onde quer que ele vá. Ela consiste em dispositivos móveis de comunicação bidirecional, dispositivos de computação e a tecnologia de rede que os conecta.

Hoje em dia, a tecnologia móvel é feita por aparelhos conectados à internet, como smartphones, tablets e relógios. Essas são as últimas novidades em uma progressão que inclui pagers bidirecionais, laptops, celulares (tipo flip), aparelhos de navegação por GPS e muito mais.

As redes de comunicação que vinculam esses aparelhos são chamadas de tecnologia sem fio. Graças a elas, os aparelhos móveis compartilham voz, dados e aplicativos móveis.

A tecnologia móvel é difundida e vem crescendo. O número de usuários de smartphone aumentou para mais de 3 bilhões, e espera-se que a força de trabalho móvel atinja 1,87 bilhão em todo o mundo até 2022.

## Tipos de redes móveis

### Redes celulares

Redes de rádio que utilizam torres de celular distribuídas que permitem que os dispositivos móveis (celulares) alternam automaticamente as frequências e se comuniquem sem interrupções em grandes regiões geográficas. O mesmo recurso básico de comutação é utilizado para que as redes móveis acomodam vários usuários em um número limitado de frequências de rádio.

### Redes 4G

Hoje é o serviço de celular mais comum nas comunicações sem fio. Ele utiliza a tecnologia de comutação de pacotes, que organiza os dados

em partes ou pacotes para a transmissão e remonta as informações no destino. O 4G (geração "G") é 10 vezes mais rápido do que o 3G. E o 5G, uma rede muito mais rápida, ainda está chegando. é aproximadamente 20 vezes mais rápido do que o 4G e utiliza um conjunto de bandas de frequência agregada para liberar a largura de banda.

### Wi-Fi

São ondas de rádio que conectam os aparelhos à internet via roteadores localizados chamados hotspots. As redes Wi-Fi (do inglês "fidelidade sem fio") são como torres de celular para acesso à internet, mas precisam estabelecer uma conexão Wi-Fi para passarem automaticamente pelo serviço. A maioria dos aparelhos móveis faz a comutação automática entre redes Wi-Fi e celular, dependendo da disponibilidade e da preferência do usuário.

### Bluetooth

É uma especificação do setor de telecomunicações para conectar aparelhos a distâncias curtas via ondas curtas de rádio. O Bluetooth é usado para conectar ou emparelhar rapidamente acessórios como fones de ouvido, alto-falantes, telefones e outros aparelhos.

## O que é 5G?

O 5G é a quinta geração da tecnologia móvel sem fio. Assim como o 4G, ele usa frequências dentro do espectro do rádio, mas o 5G usa frequências muito altas, que oferecem mais largura de banda. Isso resulta em mais dados entregues a velocidades mais altas para mais aparelhos. Pense em um streaming de vídeo para um smartphone — o 5G melhora o streaming em dez vezes não só para uma pessoa, mas para todos que estiverem vendo o vídeo ao mesmo tempo.

# Principais recursos para uma tecnologia móvel eficaz

- **Escalabilidade**: criar soluções pontuais e rígidas em uma empresa pode sair caro em termos de desenvolvimento, gestão e manutenção. Os aplicativos precisam ser concebidos de forma holística, levando em consideração as linhas de negócios, os processos e os ambientes técnicos.

- **Integração**: é fundamental saber vincular os serviços de lógica e de dados ao aplicativo, independentemente da lógica e os dados estarem no local, na nuvem ou em configurações híbridas.

- **Reutilização**: mais de 105 bilhões de aplicativos móveis foram baixados em 2018. Muitos são (ou podem ser) modificados ou combinados para aplicativos de negócios.

**Desenvolvimento na nuvem**: a nuvem oferece uma plataforma eficiente para desenvolver, testar e gerenciar aplicativos. Os desenvolvedores podem usar uma interface de programação de aplicativos (API) para vincular os aplicativos aos dados do backend e se dedicarem às funções no frontend. Eles podem adicionar autenticação para reforçar a segurança e acessar serviços cognitivos e de inteligência artificial (IA).

**Gerenciamento da mobilidade**: à medida que a tecnologia móvel é implementada, as organizações buscam soluções em gerenciamento de mobilidade empresarial (EMM) para configurar aparelhos e aplicativos; monitorar o uso e o inventário dos aparelhos; controlar e proteger os dados; e prestar suporte e solucionar problemas.

**BYOD**: Bring your own device (BYOD) é uma política de TI na qual os funcionários usam um aparelho pessoal para acessar dados e sistemas.

Adotado de forma eficaz, o BYOD melhora a produtividade, aumenta a satisfação dos funcionários e economiza dinheiro. Por outro lado, ele traz questões que precisam ser abordadas quanto à segurança e à gestão dos aparelhos.

**Segurança**: a batalha pela segurança móvel é enorme em termos de volume e complexidade. A Inteligência Artificial (IA) está emergindo como uma arma fundamental para diferenciar anomalias na segurança em grandes quantidades de dados. Ela pode identificar e remediar incidentes com malware ou recomendar ações para atender aos requisitos regulatórios — tudo em um dashboard central.

**Edge computing**: uma das principais vantagens do 5G é que ele pode aproximar as aplicações das fontes de dados ou servidores de ponta. A proximidade dos dados com a fonte traz benefícios à rede, como melhor tempo de resposta e mais disponibilidade de largura de banda. Do ponto de vista comercial, o edge computing pode realizar análises de dados mais abrangentes e gerar insights mais profundos com mais rapidez.

# Lista de Exercícios Sobre Computadores

1. Desenvolva uma análise sobre como a computação em nuvem redefiniu o conceito de computadores.
2. Investigue como dispositivos embarcados ampliam o conceito de computação além do PC.
3. Avalie o impacto da computação em criptografia e segurança digital.
4. Relacione a evolução das gerações de computadores com as transformações no mercado de trabalho.
5. Explique como o conceito de "Moore's Law" guiou a transição entre gerações.
6. Elabore um relatório detalhado sobre a evolução do armazenamento de dados em cada geração.
7. Qual é a relação entre as gerações de computadores e as mudanças sociais globais?
8. Explique como arquiteturas de hardware específicas otimizam software de inteligência artificial.
9. Elabore um relatório sobre o impacto da arquitetura RISC-V na relação hardware-software.
10. Compare e contraste as arquiteturas ARM e x86 na execução de sistemas operacionais.
11. Explique os desafios de criar drivers para hardwares não padronizados.
12. Explique as barreiras para o desenvolvimento de software para dispositivos com hardware personalizado.
13. Qual é o impacto da computação distribuída em hardware e software?
14. Como o software de simulação pode antecipar problemas em novos designs de hardware?
15. Projete um cenário onde o hardware ajusta dinamicamente suas especificações para softwares variáveis.
16. Desenvolva um relatório comparando supercomputadores e computação em nuvem.
17. Explique a convergência entre sistemas embarcados e dispositivos móveis.

18. Avalie como os microcontroladores estão sendo usados para automação industrial.
19. Explique o conceito de empacotamento 3D de transístores e sua relevância na Lei de Moore.
20. Discuta como os desafios térmicos estão limitando o crescimento descrito pela Lei de Moore.
21. Desenvolva uma previsão sobre como as arquiteturas heterogêneas podem substituir a dependência da densidade de transistores.
22. Analise o impacto ambiental do avanço descrito pela Lei de Moore.
23. Compare o crescimento previsto pela Lei de Moore com o aumento real de desempenho de CPUs modernas.
24. Discuta como a miniaturização de transístores está sendo abordada com novos materiais além do silício.
25. Explique os desafios de escalar a Internet para atender bilhões de dispositivos conectados.
26. Discuta as implicações de segurança no uso de redes peer-to-peer na Internet.
27. Avalie o impacto da computação em nuvem no funcionamento da Internet.
28. Como a inteligência artificial está sendo usada para otimizar o tráfego na Internet?
29. Compare redes centralizadas, descentralizadas e distribuídas na arquitetura da Internet.
30. Relacione o impacto das redes 5G no futuro da Internet móvel.
31. Explique o papel de tecnologias emergentes, como redes definidas por software (SDN), no gerenciamento da Internet.
32. Analise as implicações sociais e econômicas do conceito de Internet 3.0.
33. Compare o uso de MPLS e SD-WAN em redes corporativas.
34. Analise os impactos do 5G na convergência de redes de computadores e telecomunicações.
35. Explique como protocolos como OSPF e EIGRP otimizam o roteamento em redes grandes.
36. Discuta o impacto da Edge Computing na arquitetura de redes modernas.

37. Como a segmentação de rede pode ser usada para aumentar a segurança em arquiteturas complexas?
38. Explique a arquitetura de redes 5G e seus principais avanços sobre o 4G.
39. Avalie a importância de arquiteturas resilientes em redes críticas, como as usadas em bancos e hospitais.
40. Como o protocolo SNMP é usado no monitoramento de redes?
41. Explique o funcionamento do protocolo ARP e sua importância em redes locais.
42. Discuta os benefícios e limitações do uso do protocolo IPv6.
43. Como o protocolo NTP é utilizado para sincronização de tempo em redes?
44. Analise o papel de protocolos multicast em transmissão de dados.
45. Qual a diferença entre a Internet e a Web?
46. Explique o que é um navegador web e cite dois exemplos.
47. O que são URLs e como elas funcionam?
48. Descreva o papel do protocolo HTTP na World Wide Web.
49. Cite exemplos de serviços web populares e sua aplicação.
50. Explique a diferença entre serviços SOAP e REST.

# MÓDULO 2: VISÃO GERAL

**Qualquer tecnologia suficientemente avançada é indistinta de magia.**

Arthur C. Clarke

- HISTÓRIA DO JAVA
- LINGUAGENS DE MÁQUINA
- LINGUAGENS DE PROGRAMAÇÃO
- JAVA VIRTUAL MACHINE
- RESUMO
- EXERCÍCIOS

# Objetivos

- Conhecer a história e evolução da linguagem Java.
- Definir o que é um programa e como ele funciona.
- Entender linguagens de máquina e linguagens de programação.
- Aprender como instalar e configurar o ambiente Java, incluindo JDK, JRE, JVM e Eclipse.

# Introdução ao Java

Java é uma linguagem de programação de alto nível, orientada a objetos e desenvolvida pela Sun Microsystems, que foi posteriormente adquirida pela Oracle Corporation, foi lançada em 1995 e rapidamente se tornou uma das linguagens mais populares devido à sua simplicidade e robustez.

Sua sintaxe é semelhante à do C++, mas com menos complexidades, o que a torna mais fácil de aprender e usar, um dos aspectos mais notáveis da linguagem é sua capacidade de ser executada em qualquer plataforma que suporte a JVM.

Isso é conquistado através da compilação do código-fonte em bytecode, que é um formato intermediário interpretado pela JVM, essa característica de portabilidade permite que programas Java sejam executados em diferentes sistemas operacionais sem a necessidade de modificações no código.

A linguagem também é conhecida por seu forte sistema de gerenciamento de memória, que inclui a coleta de lixo (garbage collection), um processo automático que ajuda a prevenir vazamentos de memória e a melhorar a eficiência do aplicativo.

A segurança é seu outro ponto forte, com um ambiente de execução restrito que protege o sistema hospedeiro contra código malicioso.

Pode ser utilizada em uma ampla variedade de aplicações, desde aplicativos móveis e web até sistemas de controle industrial e dispositivos embarcados, sua versatilidade e robustez fazem dela uma escolha preferida para muitos desenvolvedores ao redor do mundo.

# História e Evolução do Java

Embora muitas pessoas iniciantes na programação possam discordar (ou não entender por que abordar a história da linguagem Java possa ser relevante), saiba que conhecê-la é um bom caminho andado para a compreensão dos motivos que fazem com que seja usada há tanto tempo.

Tudo começou em 1991, com a empresa Sun Microsystems. Os desenvolvedores Patrick Naughton, Mike Sheridan e James Gosling trabalhavam em conjunto em um projeto denominado **Green Project**. Na época, almejavam romper com as barreiras entre a computação e dispositivos que utilizamos diariamente, desde um relógio a um eletrodoméstico.

Sabe essas TVs interativas, smartphones conectados a tudo, inclusive, ao nosso Frigorífico e micro-ondas? Já foi um sonho nessa época. Tudo que temos hoje de mais relevante nas funções de nossos dispositivos, muito se deve ao desenvolvimento do Java. A sua filosofia se baseava em um conceito, que enfatiza o “write once, run everywhere”, em português, escreva uma única vez, execute em qualquer lugar.

Os desenvolvedores não tinham a intenção de criar apenas uma linguagem. A primeira das aplicações inventadas pela equipe foi o dispositivo conhecido por “StarSeven”. Era um controle remoto touch screen que se conectava a eletrodomésticos, estendendo funções e possibilidades de acesso remoto aos aparelhos conectados.

Para a interface do dispositivo StarSeven, criaram a linguagem Oak, que significa “carvalho”, em inglês. O próprio Gosling propôs esse nome, o StarSeven acabou não dando muito certo, e suas vendas foram um completo fracasso. Mas nem tudo estava perdido, em meados dos anos 90, no boom da Internet. Então, os desenvolvedores decidiram aplicar a linguagem Oak no desenvolvimento de sistemas Web. Foi aí que o Java deu seu primeiro passo.

# De onde veio o nome “Java”

## GreenTalk

James Gosling que na altura liderava uma equipe chamada Green **Project**. O objetivo desta equipe era criar um novo projeto que inicialmente, C++ foi a escolha original para desenvolver o projeto.

James Gosling queria aprimorar C++ para atingir o objetivo, mas devido ao alto uso de memória, essa ideia foi rejeitada e a equipe começou com uma nova linguagem inicialmente chamada de **GreenTalk**. A extensão de arquivo usada como .gt. Mais tarde, essa linguagem foi denominada Oak e, finalmente, Java.

## Oak

James Gosling renomeou a linguagem para Oak. Havia um carvalho na frente de seu escritório. James Gosling usou esse nome porque Oak representa solidariedade e Oak é a árvore nacional de vários países como EUA, França, Romênia etc. Mas a Oak Technologies já tinha a Oak como marca registrada e a equipe de James teve que pensar em outro título para a linguagem.

## Finalmente Java

A equipe colocou vários nomes como DNA, Silk, Ruby e Java. Java foi finalizado pela equipe. James Gosling apresentou o título Java com base no tipo de grão de café expresso. Java é uma ilha na Indonésia onde um novo café foi descoberto, denominado café Java. De acordo com James Gosling, Java estava entre as principais escolhas, junto com Silk. Finalmente, Java foi selecionado porque era bastante único e representava a essência de ser dinâmico, revolucionário e divertido de dizer.

# O que é um Programa?

Um programa ou programa de computador é um conjunto de instruções que são interpretadas pelo computador a fim de se executar uma determinada tarefa. Essas instruções estão armazenadas em um arquivo que comumente é chamado de executável e, dentro deste arquivo, as instruções aparecem no formato binário que é extremamente difícil de ser lido por um ser humano.

No mercado existem diversas marcas de computador, cada uma produzindo computadores com características variadas, desde a capacidade de armazenamento de dados à estética do gabinete.

Dentro de um mercado tão diversificado, nada mais natural que encontrarmos diversos tipos de processadores que não variam somente o poder de processamento, mas também de arquitetura.

Essa variedade acontece também no mundo dos sistemas operacionais. Hoje temos o Windows, Linux e o Mac OS X entre os mais populares, mas ao longo dos últimos 20 anos tivemos também o OS/2, QNX, DOS, BeOS, entre outros.

# Níveis de Linguagens de Programação

Uma linguagem de programação pode ser definida como sendo um conjunto limitado de instruções (vocabulário), associado a um conjunto de regras (sintaxe) que define como as instruções podem ser associadas, ou seja, como se pode compor os programas para a resolução de um determinado problema.

As linguagens de programação podem ser classificadas em níveis de linguagens, sendo que as linguagens de nível mais baixo estão mais próximas da linguagem interpretada pelo processador e mais distante das linguagens naturais.

- **Linguagem de Máquina**

  Na linguagem de máquina, a representação dos dados e das operações (instruções) que constituem um programa, é baseada no sistema binário, que é a forma compreendida e executada pelo hardware do sistema.

- **Linguagem Hexadecimal**

  Para simplificar a compreensão e a programação de computadores, foi adotado (num primeiro momento) a notação hexadecimal para representar programas em linguagens de máquina. Mas a programação e leitura usando a linguagem hexadecimal continuou impraticável.

- **Linguagem Assembly**

  A linguagem de máquina de cada processador é acompanhada de uma versão “legível” da linguagem de máquina que é a chamada linguagem simbólica Assembly. Simbólica, pois esta linguagem não é composta de números binários e hexadecimais, como nas duas linguagens anteriores. A linguagem Assembly é na realidade uma versão legível da linguagem de máquina.

# Linguagens de Programação

Tal como nós, humanos, utilizamos a linguagem para comunicar uns com os outros, da mesma forma que utilizamos a linguagem para comunicar com os computadores – em suma, para sermos "compreendidos" pelas máquinas – foram criadas várias linguagens utilizadas pelos programadores.

Eis uma definição mais específica: uma linguagem de programação é uma linguagem utilizada por criadores e programadores para transformar um conjunto de comandos e instruções escritas em dados e tarefas específicas.

Uma linguagem de programação é utilizada principalmente para desenvolver aplicações de secretária, sítios Web, aplicações móveis, programas e plataformas comerciais. Ao escrever estes códigos, é possível converter as suas ideias em código de máquina que pode ser "lido" por computadores de vários tipos. Em suma, as linguagens de programação são utilizadas para dar instruções a um computador, dizendo-lhe o que fazer e como o fazer.

No começo da história da computação, os computadores recebiam instruções por meio de dois estados: ausência e presença. As máquinas liam cartões perfurados, em que um furo representava uma ausência. Posteriormente, com a evolução da tecnologia, isso mudou. Havia dispositivos que permitiam que essa presença e ausência fossem realizadas por meio de pulsos elétricos, que ligavam e desligavam componentes da máquina. E a combinação desses dois estados diversas vezes transmitia uma mensagem.

Como são dois estados, ficou decidido que eles seriam representados pelos algarismos 1 e 0. Então, quando falamos para o computador 1, pedimos para ele enviar um pulso elétrico por meio de seus componentes, e, quando dizemos 0, ele não envia nada. Combinando vários uns e zeros, conseguimos codificar instruções.

## Os três componentes básicos de uma linguagem de programação

Vamos conhecer alguns componentes que estarão presentes em todo sistema linguístico, tanto o nosso humano, quanto os da programação.

## Vocabulário

Na nossa linguagem humana, conhecemos o vocabulário como um conjunto de léxicos (palavras) que usamos para construir nossas sentenças. Quanto maior seu vocabulário, mais escolhas linguísticas você poderá fazer na hora de usar a língua. Na programação, isso é exatamente igual. Cada linguagem de programação tem seu próprio vocabulário de palavras aceitas e que são compreendidas.

Assim como uma palavra do vocabulário em inglês pode não estar presente no vocabulário do português brasileiro, uma linguagem de programação poderá ter uma palavra diferente de outra para a mesma ação.

### Mas, por que não criar um vocabulário único para todas as linguagens de programação?

Essa é uma excelente dúvida! Assim como as línguas naturais humanas, as linguagens de programação são utilizadas em diferentes nichos e necessidades. Portanto, tudo depende do contexto em que cada uma delas é criada e qual a intenção por trás dela.

## Sintaxe

Toda linguagem precisa de um conjunto de regras que determinam o que é válido ou não em determinado sistema. No português, por exemplo, a frase “Menina a ser quer programadora” não só é impossível de ser dita naturalmente por uma pessoa falante como dificilmente seria compreendida. Essa mesma lógica se transporta para a programação.

Cada linguagem de programação tem uma estrutura adequada para que o computador consiga entender corretamente o que precisa fazer.

Exemplos de como a sintaxe aparece em linguagens de programação:

- No uso de espaços;
- Na sequência e na relação entre sentenças e elementos;
- Na diferenciação de caracteres maiusculas e minúsculas.

## Semântica

Semântica diz respeito ao sentido, ou seja, aquilo que o termo ou a sentença querem dizer quando utilizados. Na linguagem de programação, isso se reflete na lógica interna de como construir sentenças para determinada ação. Diferentes linguagens de programação utilizam diferentes sintaxes para a mesma semântica.

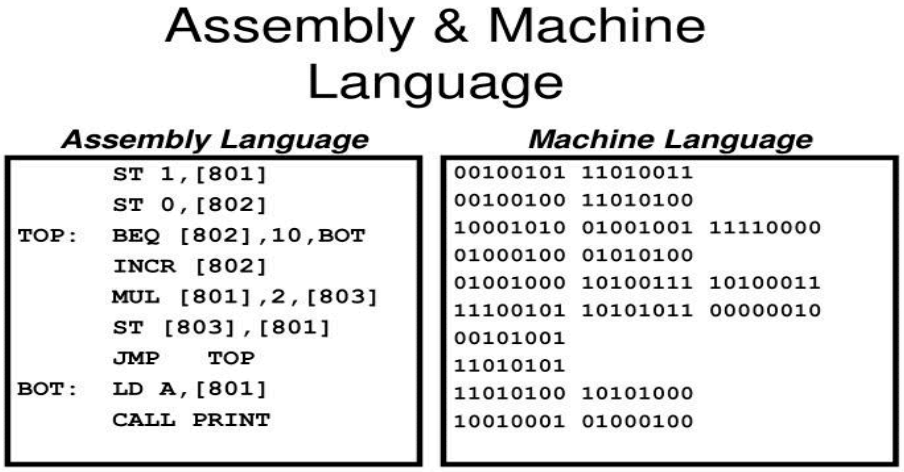
# Tipos de linguagem de programação

De maneira bastante geral, as linguagens de programação se dividem maioritariamente em dois grupos principais, definidos pela distância que eles têm da nossa forma de comunicação, a língua natural.

## Linguagem de baixo nível

As linguagens de baixo nível são mais próximas do código compreendido pelo computador. Baixo nível se refere a um baixo nível de abstração, significando que ela é bem lógica e literal. É como escrever diretamente na língua do computador. Provavelmente, se você olhar para uma linguagem dessas, você não será capaz de definir o que ela faz tão facilmente.

Uma das linguagens de baixo nível se chama Assembly. Veja um exemplo de uso dela abaixo, comparada a uma linguagem de máquina:



## Linguagem de alto nível

As linguagens de alto nível são aquelas com alto nível de abstração, incluindo mais recursos da nossa língua humana. Então, é muito mais fácil reconhecer os elementos e estruturar uma lógica a partir dessas linguagens. Programas especializados vão traduzir esse código para a linguagem de máquina, então as pessoas programadoras não precisam se preocupar com isso. Também, é seguro dizer que é muito mais fácil encontrar um erro a partir de linguagens assim.

## **OBS:**

Como convenção, a maioria das linguagens de programação se baseia no vocabulário da língua inglesa para construir seus termos. Por isso, ter um conhecimento de inglês prévio a começar a programar é bastante benéfico.

# Paradigmas de Programação

Se levarmos ao pé da letra a definição do vocábulo “Paradigma”, encontramos que ele é uma espécie de protótipo ou modelo, uma forma com a qual enxergamos algum assunto a partir de uma referência. Essa definição nos ajuda a compreender um pouco melhor esse termo na programação.

- Dentro da lógica de programação, um paradigma é uma abordagem, ou seja, um ponto de vista que ajuda a gente a resolver um problema.

Uma mesma mensagem pode ser dita ou escrita de mil e uma formas diferentes. Paralelamente, você consegue criar inúmeros programas para um computador realizar a mesma ação, dependendo do paradigma que você tiver como referência.

## Principais Paradigmas de Programação das Linguagens

Vamos conhecer os 6 principais e mais importantes paradigmas de programação.

## Paradigma imperativo

Na língua portuguesa, conhecemos os verbos imperativos e, desde cedo, compreendemos que eles servem para dar ordens, como “Pare!” ou “Não fume!”. A programação imperativa segue a mesma lógica. Esse é o método mais antigo e tradicional que diz, bloco por bloco de código, o que o computador deve fazer e como ele deve chegar no resultado final. É como se ele mostrasse o caminho e ensinasse o computador a realizar a ação.

Esse paradigma é largamente descrito como responsável por alterar estados. Isso significa que, linha após linha, ele muda a situação do computador. Para entender isso, vamos voltar ao português.

Quando utilizamos o imperativo “Pare!”, estamos alterando o estado de algo que estava em movimento, deixando-o inerte. A mesma coisa acontece na programação imperativa, em que o estado do computador muda toda hora.

Linguagens que suportam esse paradigma:

- C;
- C++;
- Python;
- Lua;
- BASIC;
- PHP;
- etc.

## Paradigma declarativo

Se no paradigma imperativo se diz o que o computador deve fazer para chegar em um resultado, no declarativo a gente “declara”, ou seja, descreve, o que a gente quer de resultado e deixa que a máquina dê um jeito de chegar nele por conta própria.

Como o declarativo cria uma nítida oposição com o imperativo, vamos criar uma metáfora para entender como as duas abordagens funcionam.

Linguagens que suportam esse paradigma:

- Prolog;
- Lisp;
- Erlang.

## Paradigma funcional

O paradigma funcional tem seu nome derivado de função. Uma função é uma expressão matemática que a gente pode reutilizar várias vezes em distintos contextos. Logo, o foco dessa programação, quando pura, é criar um código mais enxuto que contém somente funções matemáticas que são executadas e que executam umas às outras. Variáveis nesse estilo de programação são imutáveis.

Linguagens que suportam esse paradigma:

- Haskell;
- JavaScript;
- Python.

## Paradigma lógico

Como o nome indica, o paradigma lógico é uma forma de resolver problemas que utiliza sentenças pautadas na lógica formal, utilizando o conceito de Verdadeiro e Falso. As linhas de código são escritas como sentenças que devem obedecer regras e fatos dentro de uma determinada lógica estabelecida.

Linguagens que suportam esse paradigma:

- ASP;
- Datalog;
- Prolog.

# Paradigma orientado a objetos

A programação orientada a objetos é um paradigma que se voltará completamente aos dados e funções que são chamados de objetos. O conceito de objeto é relacionado a qualquer dado que contenha atributos e métodos. Para entender esses últimos nomes, pensemos em uma pessoa como objeto. Essa pessoa tem atributos, como nome, idade, altura, CPF, para além de métodos, que é como ela age, como ela se movimenta, como ela fala…

Nesse tipo de programação, a gente consegue criar partes de código que servem como “projetos” para gerar objetos mais específicos. A esses projetos, damos o nome de classes.

Seguindo nosso exemplo, uma classe “pessoa” pode ter atributos como “nome”, “Idade” e “altura”, assim como métodos “andar” e “falar”. A partir desse projeto, conseguimos gerar várias “pessoas”, como o Lucas, o João e a Maria, cada qual com um nome, uma idade e uma altura diferentes.

A programação orientada a objetos é uma das mais populares atualmente e tem diversas línguas de programação que a suportam, como, por

exemplo:

- Python;
- Java;
- C++

# Paradigma orientado a eventos

A programação orientada a eventos é bastante popular atualmente para o desenvolvimento de software e aplicações. Como o nome diz, essa programação está voltada em focar no desenvolvimento de programas com a intenção de pensar em como ela vai reagir às ações da pessoa usuária.

Então, o foco dessa programação é pensar no que a pessoa usuária vai fazer, como clicar, fazer upload, arrastar um item, tocar em algo, etc. Essas ações são denominadas eventos. Então, o programa é inteiramente dividido em pequenas partes reutilizáveis.

A grande maioria de tecnologias de desenvolvimento atualmente suportam o paradigma orientado a eventos.

# Como baixar e instalar o Java?

## 1. Acesse o site oficial:

• Vá ao site da Oracle para baixar o **JDK** (Java Development Kit): https://www.oracle.com/java/technologies/javase-downloads.html.

• Alternativamente, você pode usar a versão OpenJDK (uma versão open-source): https://jdk.java.net/.

## 2. Escolha a versão apropriada:

• Selecione a versão do **JDK** que deseja baixar (geralmente a última versão estável é recomendada).

• Escolha a versão correspondente ao sistema operacional (Windows, macOS ou Linux) e ao tipo de arquitetura (x64 ou ARM).


## 3. Baixe o instalador:

• Clique no link de download do instalador apropriado (para Windows, é geralmente um arquivo **.exe**).

## 4. Instale o Java:

• Execute o arquivo **.exe** baixado.

• Siga as instruções na tela, escolhendo as opções padrão, a menos que tenha requisitos específicos.

• O instalador normalmente configura automaticamente as variáveis de ambiente.

# Configurando o ambiente em Java

## 1. Verifique se o Java foi instalado corretamente:

• Abra o Prompt de Comando e digite:

**java -version**

• Se o Java estiver instalado corretamente, você verá a versão instalada.

## 2. Configure as variáveis de ambiente (caso necessário):

• Abra o menu Iniciar e procure por Configurações Avançadas do Sistema.

• Clique em Variáveis de Ambiente na aba Avançado.

• Na seção Variáveis do Sistema, procure por **Path** e clique em Editar.

• Adicione o caminho do diretório bin do Java (geralmente C:\Program Files\Java\jdk-[versão]\bin).

• Crie uma variável chamada **JAVA_HOME** (caso ainda não exista) e defina o valor como o caminho para o diretório raiz do JDK (C:\Program Files\Java\jdk-[versão]).

## 3. Teste novamente no Prompt de Comando:

• Digite:

**javac -version**

• Se o compilador javac funcionar, o ambiente Java está configurado corretamente.

**Boas práticas de programação:**

| |
|---|
| Leia os manuais da versão do Java que estiver usando. Consulte frequentemente estes manuais para se certificar do conhecimento do rico Conjunto de recursos da linguagem e de que eles estão sendo usados corretamente, o site da **Oracle** é uma referência fiável, e está sempre atualizado. |

# Editores de Código Java

Um IDE ou Ambiente de Desenvolvimento Integrado, possui todas as ferramentas que um desenvolvedor pode precisar para escrever, depurar e testar o código que está sendo desenvolvido.

Com o uso de um IDE, o desenvolvedor vai poder trabalhar em apenas um ambiente de desenvolvimento, como resultado, o desenvolvedor poderá ter um desempenho melhor.

Como foi mencionado, existem várias opções de IDEs, cada IDE possui características e recursos diferentes, porém, todo IDE possui um editor de texto, ferramentas para construção e um depurador.

## Bloco de Notas

Na máquina Windows, você pode usar qualquer editor de texto simples como o Bloco de Notas (recomendado para este tutorial), TextPad.

## Eclipse

Eclipse é um ambiente de desenvolvimento integrado moderno que está disponível para desktop e nuvem, a versão para nuvem do Eclipse, se chama Eclipse Che, que traz a possibilidade para os desenvolvedores desenvolverem aplicativos através de algum navegador web.

Às duas versões do Eclipse possuem todas as funcionalidades necessárias, mas, você também poderá adicionar plug-ins, você vai encontrar várias opções de plug-ins no Eclipse Marketplace, um dos recursos que o Eclipse possui é um compilador customizado.

## Netbeans

Netbeans é o IDE oficial para o Java 8, este IDE é de código aberto e é usado para desenvolver aplicativos para desktop, aplicativos para mobile e aplicativos da web, o Netbeans ajuda a aumentar a produtividade e possui extensões que permitem que você use outras linguagens de programação, como, por exemplo, C++ e JavaScript.

O Netbeans possui ferramentas integradas que traz a possibilidade de refatorar o código, evitando escrever os códigos sem erros.

Você vai poder usar esse IDE no Linux, macOS, Solaris e Windows.

https://www.netbeans.org/index.html.

## IntelliJ IDEA

IntelliJ IDEA está disponível em 2 edições, a primeira, é a versão comunidade que é licenciada pelo Apache 2, e a segunda versão, é a edição comercial proprietária, esse IDE oferece recursos de refatoração entre linguagens.

O IDE IntelliJ IDEA também oferece recursos que facilitam a vida do desenvolvedor Java, como, por exemplo, conclusão em cadeia, injeção de linguagem e conclusão inteligente, e também, tem suporte para outras linguagens de programação baseadas em JVM.

IntelliJ IDEA está disponível nas plataformas Linux, macOS e Windows.

Eclipse e pode ser baixado em https://www.eclipse.org/ .

## BlueJ

BlueJ é um IDE Java projetado com o foco em fins educacionais, foi desenvolvido como um IDE voltado para os desenvolvedores iniciantes, no entanto, os desenvolvedores mais experientes usam esse IDE para desenvolver em Java.

BlueJ está disponível nas plataformas Linux, macOS e Windows.

# Como baixar e instalar o Eclipse no Windows?

## 1. Acesse o site oficial do Eclipse:

- Vá para https://www.eclipse.org/downloads/.

## 2. Escolha a versão do Eclipse:

- Clique em Download x86_64 ou escolha uma versão específica como "Eclipse IDE for Java Developers" se for trabalhar principalmente com Java.

## 3. Baixe o instalador:

- O download geralmente será um arquivo .exe (como eclipse-inst-jre-win64.exe).

## 4. Instale o Eclipse:

- Execute o instalador.
- Escolha a versão desejada (por exemplo, "Eclipse IDE for Java Developers").
- Escolha o diretório de instalação ou use o padrão.

## 5. Inicie o Eclipse:

- Após a instalação, abra o Eclipse.
- Escolha um local para o Workspace (diretório onde seus projetos serão armazenados).

## 6. Verifique a integração com o Java:

- Certifique-se de que o Eclipse está reconhecendo o JDK corretamente:

- Vá em Window → Preferences → Java → Installed JREs.

- Se o JDK não aparecer, clique em Add... e adicione o caminho para o diretório raiz do JDK configurado no passo 2.6.

# Kit de Desenvolvimento Java (JDK)

Vamos lá entender o que é o Java Development Kit, ou JDK! O JDK é um conjunto de ferramentas de software que permite aos desenvolvedores criar aplicações e applets na linguagem de programação Java. Pense nele como uma caixa de ferramentas que contém tudo o que você precisa para construir software em Java. Isso inclui um compilador, um interpretador, bibliotecas e outros utilitários que facilitam o desenvolvimento. Sem o JDK, seria muito difícil programar em Java, pois ele fornece a base necessária para compilar e executar seus códigos.

## Para que serve o Java Development Kit?

Você já se pegou tentando montar um móvel sem as ferramentas certas? Aí está a importância do JDK! Ele serve para proporcionar um ambiente completo para o desenvolvimento de software em Java. Com o JDK, você pode:

- **Compilar Código**: O JDK inclui um compilador chamado **javac**, que transforma seu código fonte em bytecode, que pode ser executado na Java Virtual Machine (JVM).

- **Executar Aplicações**: Com o interpretador **java**, você pode rodar seus programas e ver como eles funcionam.

- **Gerar Documentação**: O JDK também possui uma ferramenta chamada **Javadoc**, que gera documentação a partir dos comentários no seu código, facilitando a compreensão do que cada parte faz.

- **Depurar Código**: Ferramentas como o **jdb** ajudam a encontrar e corrigir erros no seu código.

# Java Virtual Machine (JVM)

A Máquina Virtual Java, ou JVM, pode ser considerada como um compilador virtual na linguagem. Para entendermos suas características e necessidades de uso, precisamos compreender, primeiramente, que um arquivo compilado em Java não é um executável. Ele é dependente de uma máquina virtual, a qual está presente em diversas versões — uma para cada sistema operacional.

A JVM é responsável por toda execução e interpretação do código Java no dispositivo em que está sendo executado — memória a ser utilizada, threads a serem executadas e tudo o mais.

## Para que serve a Java Virtual Machine?

A JVM serve para executar programas Java e também outros que foram convertidos para um formato chamado bytecode. Quando você escreve um programa em Java e o compila, ele se transforma em arquivos **.class** que contêm esse bytecode. A JVM pega esses arquivos e os executa. Além disso, ela cuida de várias tarefas importantes, como gerenciamento de memória e segurança. Você já se pegou tentando organizar tantas ideias na cabeça que parece que seu cérebro vai explodir? A JVM ajuda a acalmar essa confusão, gerenciando tudo nos bastidores para que você possa focar no que realmente importa: o seu código!

## Por que usar a Java Virtual Machine?

Usar a JVM é fundamental para qualquer desenvolvedor que trabalha com Java. Imagine uma empresa de software: se eles não usassem a JVM, teriam que reescrever seus programas para cada tipo de computador ou sistema operacional. E sabe o que é melhor? A JVM permite que os desenvolvedores se concentrem na lógica do programa sem se preocupar com as diferenças entre plataformas. Isso aumenta a eficiência e reduz o tempo de desenvolvimento.

# Componentes da JVM e suas funções

A arquitetura da JVM é constituída por vários componentes que trabalham em conjunto para gerir o ciclo de vida das aplicações Java. Esses componentes incluem:

- **Classloader**: O Classloader é responsável por carregar classes Java do disco para a memória da JVM, resolver dependências de classe e inicializar classes enquanto o programa está em execução. O Classloader segue uma hierarquia de delegação, começando com o Bootstrap Classloader, seguido pelo Extension Classloader e pelo Application Classloader.

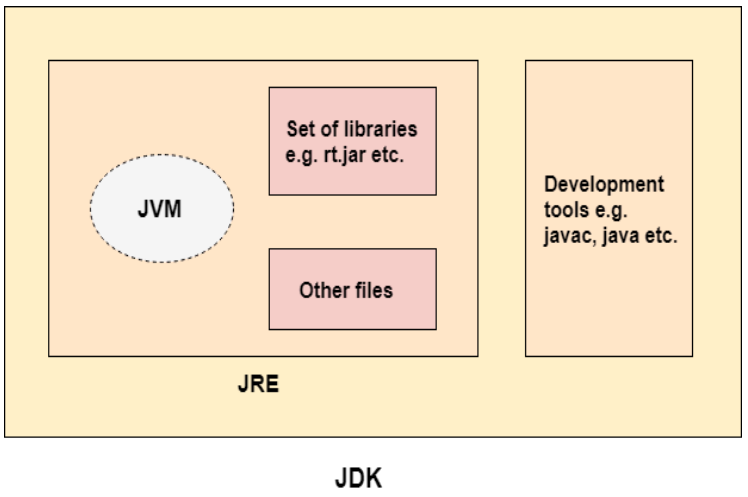
- **Áreas de dados em tempo de execução**: A JVM aloca espaços de memória chamados Áreas de Dados em Tempo de Execução durante a execução do programa. Estes espaços de memória incluem o Heap, Stack, Method Area, Constant Pool e PC Registers, que armazenam dados necessários para diferentes aspectos do ciclo de vida da aplicação.

- **Motor de execução**: O motor de execução é o componente central responsável pela execução do bytecode Java. O motor de execução interpreta o bytecode e converte-o em código de máquina nativo durante o tempo de execução. Ele inclui componentes como o Interpretador, o Compilador Just-In-Time (JIT) e o Coletor de Lixo.

Nas secções seguintes, iremos aprofundar os detalhes da gestão de memória da JVM e os vários espaços de memória que constituem a arquitetura da JVM.

# Diferenças entre JDK, JRE e JVM

Uma grande confusão que paira sobre quem está começando a aprender sobre o mundo Java é a diferença entre JDK, JRE e JVM. Neste capítulo você vai entender, de uma vez por todas, a diferença entre essas três siglas e porque é importante saber para que serve cada um.




A primeira coisa que você precisa saber é que o Java é muito conhecido por trazer o conceito de multiplataforma. Na verdade, esse é o motivo do grande sucesso do Java a mais de vinte anos! O famoso WORA (Write once, run anywhere.), ou em bom português: "Escreva uma vez, execute em qualquer lugar".

Na prática, só de entender esse conceito e como o Java faz para tornar possível escrever um único código e executá-lo em qualquer sistema operacional, você já vai perceber, por alto, a diferença entre JDK, JRE e JVM e onde cada um se encaixa.

O fluxo é basicamente o seguinte:

1. Você escreve o seu código-fonte (arquivo com a extensão **.java**).

2. Você utiliza o JDK para compilar o seu código-fonte e gerar o arquivo bytecode (arquivo com a extensão **.class**).

3. Para executar o seu programa, a JVM lê o seu arquivo compilado (.class) e as bibliotecas padrões do Java que estão no JRE.

4. Pronto, seu programa está rodando e todo mundo está feliz!

Então, de forma geral, já deu para entender para que serve cada um:

- JDK (Java Development Kit) - é o Kit de Desenvolvimento Java responsável por compilar código-fonte (.java) em bytecode (.class)

- JVM (Java Virtual Machine) - é a Máquina Virtual do Java responsável por executar o bytecode (.class)

- JRE (Java Runtime Environment) - Ambiente de Execução do Java que fornece as bibliotecas padrões do Java para o JDK compilar o seu código e para a JVM executar o seu programa.

Este é o básico que você precisa saber sobre essas três siglas e como o Java funciona.

Mas estes 3 componentes têm um certo relacionamento entre eles. Vamos aprofundar um pouco mais o entendimento de cada um e como eles se relacionam entre si para possibilitar a "mágica" do mundo Java.

## Java Runtime Environment (JRE)

O Ambiente de Execução Java, ou o JRE, atua na combinação do código Java — a partir das aplicações desenvolvidas na JDK, juntamente às bibliotecas necessárias para executá-los em uma JVM. A partir disso, o JRE pode criar uma instância da JVM, executando efetivamente os códigos Java, como já falamos, as JVM estão presentes nos mais diversos sistemas operacionais e no ambiente de execução — em outras palavras, é o que faz a ponte entre o JDK e a JVM, tornando possível a execução do programa escrito em Java.

# Compilador JIT e Coletor de Lixo

O Compilador Just-In-Time (JIT) e o Coletor de Lixo são componentes essenciais da JVM que otimizam significativamente o desempenho do aplicativo e gerenciam os recursos do sistema.

## Compilador JIT

O compilador Just-In-Time (JIT) é responsável pela conversão do bytecode Java em código de máquina nativo em tempo de execução. Este processo otimiza a velocidade de execução das aplicações Java. O compilador JIT compila métodos chamados com frequência, armazena em cache o código compilado e o reutiliza em execuções futuras, reduzindo a sobrecarga de interpretar o bytecode repetidamente.

A JVM utiliza um método de "detecção de hotspot" para identificar métodos frequentemente chamados. Quando o limite de hotspot é atingido, o compilador JIT entra em ação e compila o bytecode em código de máquina nativo. A CPU executa este código compilado diretamente, conduzindo a tempos de execução significativamente mais rápidos.

## Coletor de lixo

O Coletor de Lixo (GC) é um componente essencial da JVM responsável pela automatização da gestão da memória. Ele desloca a memória de objetos que a aplicação não precisa mais ou não faz referência. Este processo minimiza as fugas de memória e optimiza a utilização de recursos nas aplicações Java. A JVM usa uma estratégia de coleta de lixo geracional, dividindo a memória heap em gerações Young e Old. A Geração Jovem é subdividida em Espaço do Éden, Espaço do Sobrevivente 0 (S0) e Espaço do Sobrevivente 1 (S1).

A ideia básica por detrás da recolha de lixo geracional é que a maioria dos objetos tem um tempo de vida curto e é provável que sejam recolhidos logo após a sua criação. Assim, a alocação e desalocação frequente de memória na Geração Jovem optimiza o processo de recolha de lixo. O Coletor de Lixo limpa os objetos não utilizados na memória heap usando vários algoritmos, como Mark-Sweep-Compact, Copying e Generational Collection.

## Compiladores

Classicamente, um compilador traduz um programa de uma linguagem textual facilmente entendida por um ser humano para uma linguagem de máquina, específica para um processador e sistema operacional.

Um compilador é um programa de computador (ou um grupo de programas) que, a partir de um código fonte escrito em uma linguagem compilada, cria um programa semanticamente equivalente, porém escrito em outra linguagem, código objeto.

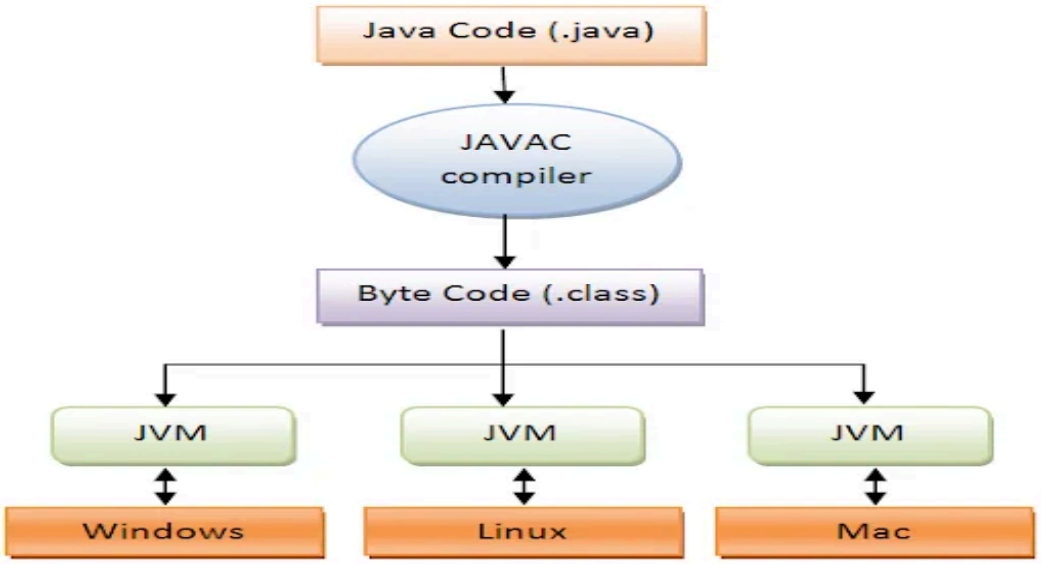
As linguagens de programação modernas abstraem muita coisa, assim quando compilamos ela o computador precisa dar um jeito de entender, por isso ele compila para linguagens de mais baixo nível (mais próximo da linguagem de máquina).

Mas como tudo na vida isso tem exceções, esse comportamento pode ser diferente em alguns casos, como por exemplo:

- Linguagens híbridas: o compilador tem o papel de converter o código fonte em um código chamado de bytecode, que é uma linguagem de baixo nível. Um exemplo deste comportamento é o do compilador da linguagem Java que, em vez de gerar código da máquina, gera código chamado Java Bytecode.

## Bytecode

Bytecode é um intermediário entre a linguagem de programação e a linguagem de máquina. No Java o bytecode é executado na JVM (java virtual machine ou máquina virtual do Java), dessa forma a portabilidade (capacidade de executar em qualquer lugar) é maior, visto que ele não é compilado direto para a máquina e sim para um bytecode que depois vai para a máquina.




## Fases da compilação

Um compilador passa por várias fases, não vou entrar a fundo em cada uma delas, vou dar apenas uma breve explicação:

- Análise Léxica: Ler o código fonte de carácter em caráter, remove comentários, espaços em brancos, etc.

- Análise Sintática: Responsável por ver se os símbolos contidos ali formam um código válido ou não.

- Análise Semântica: Verifica todas as regrinhas, tipagem, etc.

- Geração do código intermediário: Gera um código intermediário com base no código fonte.

- Otimização: Analisa o código intermediário e otimiza para garantir a performance.

- Geração do código final: Gera o código que vai ser entendido.

## Diferença entre JIT e JVM

Explore como o JIT interage com a JVM, suas funções exclusivas e a diferença entre os dois.

| Característica | JVM | JIT |
|---|---|---|
| Definição | Máquina virtual que executa bytecode. | Componente da JVM que compila |
| Função principal | Abstrair o hardware e interpretar bytecode. | Otimizar a execução traduzindo bytecode para código nativo. |
| Alcance | Garante a compatibilidade entre plataformas. | Focado em melhorar o desempenho de trechos específicos. |
| Execução | Pode executar código apenas interpretando. | Atua junto à JVM, compilando hotspots para velocidade. |
| Desempenho | Mais lento sem o JIT. | Aumenta a velocidade, reduzindo o custo da interpretação. |

# Características da Linguagem

## Simples e familiar

Linguagem simples e de fácil manipulação, possui sintaxe muito parecida com C++ que é uma das mais conhecidas no meio. Java é muitas vezes considerada uma versão simplificada da linguagem C++, onde Java não possui características como arquivos headers, ponteiros, sobrecarga de operadores, classes básicas virtuais, dentre outras que somente aumentavam a dificuldade dos programadores com a linguagem C++.

## Orientada a Objetos

Paradigma atualmente mais utilizado na construção de softwares. Permite que se focalize o dado, enfim, o objeto. Java não é uma linguagem 100% orientada a objetos, como Smaltalk, onde qualquer elemento, (operadores, sinais, tipos de dados, ...) são objetos. Em Java há os tipos primitivos de dados que não são objetos, mas foram criados e incorporados ao Java para permitir uma melhor forma de utilização da linguagem pelos programadores. Outra característica importante da linguagem Java em relação à linguagem C++, é que Java não suporta herança múltipla.

## Compilada e Interpretada

Um programa desenvolvido em Java necessita ser compilado, gerando um bytecode. Para executá-lo é necessário então, que um interpretador leia o código binário, o bytecode e repasse as instruções ao processador da máquina específica. Esse interpretador é conhecido como JVM (Java Virtual Machine). Os bytecodes são conjuntos de instruções, parecidas com código de máquina. É um formato próprio do Java para a representação das instruções no código compilado.

## Pronta para Redes

As funcionalidades que são fornecidas pela linguagem Java para desenvolver programas que manipulam as redes através das APIs são simples e de grandes potencialidades, através destas APIs pode-se manipular protocolos como TCP/IP, HTTP, FTP e utilizar objetos da grande rede via URLs.

## Distribuído

Programas Java são “linkados” em tempo de execução. Os bytecodes gerados durante a compilação só serão integrados na execução. Um objeto X existente em um arquivo quando instanciado, somente será alocado na memória em tempo de execução. Se alguma alteração ocorrer na classe que define o objeto X, somente o arquivo da classe com a alteração necessita ser compilado.

## Multi Processamento (Multithread)

Suporta a utilização de threads. Threads são linhas de execução, executadas concorrentemente dentro de um mesmo processo. Diferentemente de outras linguagens, programar utilizando Threads é simples e fácil na linguagem Java.

## Portabilidade

Pode ser executado em qualquer arquitetura de hardware e sistema operacional, sem precisar ser recompilado. Um programa Java pode ser executado em qualquer plataforma que possua um interpretador Java (ambiente de execução). Além disso, não há dependência de implementação, como por exemplo, os tamanhos dos tipos primitivos não diferem entre si, são independentes da máquina em que está a aplicação. Assim, o tipo int possui sempre um tamanho de 32-bits em Java e em qualquer máquina que esteja sendo executada.

## Segura

O Java fornece uma série de mecanismos para garantir a segurança dos aplicativos. Um programa em Java não tem contato com o computador real; ele conhece apenas a máquina virtual (JVM). A máquina virtual decide o que pode ou não ser feito. Um programa Java nunca acessa dispositivos de entrada e saída, sistema de arquivos, memória, ao invés disso ele pede a JVM que acesse.

# Java e as suas plataformas

A linguagem Java usa quatro tipos diferentes de plataformas, cada uma com uma especificidade de aplicação diferente.

Para você entender melhor sobre o tema, trazemos cada uma delas, na sequência — para você já ir se acostumando e decidindo qual será a sua necessidade, lembrando que neste livro vamos usar a Java SE.

## Java ME

Também conhecida por “Micro Edition”, a Plataforma Java ME oferece ao usuário uma API com as ferramentas mais básicas necessárias para o desenvolvimento de uma aplicação em Java. Aqueles aplicativos nos telefones celulares mais antigos podem ser feitos com essa edição, sem problemas. O mesmo serve para dispositivos mais simples, como relógio ou algum outro eletroeletrônico que utilize Java.

## Java SE

Conhecido também como Versão Standard do Java, o Java SE consegue oferecer os principais recursos possíveis de serem utilizados com a linguagem. Seu uso é estendido, se comparado ao Java ME, podendo ter aplicações executadas nas áreas de rede, banco de dados, segurança da informação, entre muitos outros. Saiba que até mesmo interfaces gráficas completas estão incluídas aqui.

A Java SE (Java Platform, Standard Edition) é a plataforma de programação voltada para criação de applets e desenvolvimento de softwares para desktop, destinados a computadores pessoais, notebooks ou outras arquiteturas com maior capacidade de processamento e memória.

Os aplicativos podem ser executados em Windows, Mac OS, Linux, Solaris ou outros sistemas operacionais, contanto que estes tenham instalado o ambiente de execução JRE (Java Runtime Environment).

## Java EE

A Java EE, ou Java Platform, Enterprise Edition, é a plataforma que disponibiliza recursos para o desenvolvimento de aplicações corporativas voltadas para web e servidores de aplicação.

Java EE foi projetada para suportar sistemas de uso em larga escala, ou seja, para uma quantidade significativa de usuários, possibilitando o desenvolvimento de aplicações escaláveis, robustas e multicamadas. Toda essa estrutura incorpora características como segurança e confiabilidade, muitas vezes consideradas difíceis de serem implementadas. Com o objetivo de amenizar as dificuldades de implementação dessas características, o Java EE fornece um conjunto de tecnologias que reduz significativamente o custo e a complexidade de desenvolvimento.

## JavaFX

Nossa última plataforma Java que exploraremos é a JavaFX — ela traz muitos recursos úteis para o desenvolvimento de aplicações Web de maneira mais leve e compacta, por meio de uma interface API. Talvez essa plataforma seja uma das maiores responsáveis pela resiliência do Java após tanto tempo.

Uma de suas maiores utilidades vem de algumas facilidades para a pessoa desenvolvedora, como aceleração gráfica e de mídia, usando os recursos do sistema em que o Java está sendo hospedado para execução das aplicações.

Utilizando essa abordagem, o Java consegue tirar uma imensa vantagem dos dispositivos em que está operando, entregando uma alta taxa de desempenho. Assim, supera muitas outras linguagens e plataformas modernas em vários quesitos.

# Vantagens e Desvantagens de Cada Plataforma

Ao analisar as quatro principais plataformas de desenvolvimento Java, é possível identificar as características-chave, forças e fraquezas de cada uma delas. A Java SE se destaca pela sua robustez, portabilidade e ampla adoção, sendo a base fundamental do ecossistema Java. A Java EE, por sua vez, é a escolha ideal para o desenvolvimento de aplicações empresariais complexas, com recursos avançados de segurança, transações e integração. Já a Java ME é uma plataforma voltada para dispositivos móveis e embarcados, com foco em soluções leves e otimizadas. Por fim, a JavaFX se diferencia por sua ênfase no desenvolvimento de interfaces gráficas ricas e interativas.

Cada uma dessas plataformas apresenta vantagens e desvantagens específicas. A Java SE oferece uma ampla gama de bibliotecas e ferramentas, mas pode exigir uma curva de aprendizado inicial. A Java EE é robusta e escalável, mas também pode ser mais complexa de configurar e implantar. A Java ME é eficiente em dispositivos com recursos limitados, mas enfrenta a fragmentação do mercado de dispositivos móveis. A JavaFX proporciona interfaces de usuário avançadas, mas requer uma especialização adicional em desenvolvimento de interfaces gráficas.

Ao comparar essas plataformas, é importante considerar os requisitos específicos do projeto, as habilidades da equipe de desenvolvimento, as restrições de hardware e as necessidades de integração com outros sistemas. Essa análise cuidadosa permitirá aos desenvolvedores escolher a plataforma Java mais adequada para atender às demandas de seus projetos.

# Lista de Exercícios

1. Explique a diferença entre bytecode e código de máquina.

2. Quais são os principais usos do Java hoje? Cite pelo menos três.

3. Liste três características que tornaram o Java uma linguagem popular.

4. Defina paradigmas de programação e cite dois exemplos comuns.

5. Diferencie entre linguagens de programação compiladas e interpretadas.

6. O que significa Write Once, Run Anywhere no contexto do Java?

7. Explique o papel do Class Loader na JVM.

8. Qual é a diferença básica entre linguagens de alto e baixo nível?

9. Descreva a função do compilador Javac.

10. Quais são os principais componentes do JDK?

11. Qual é o objetivo do Garbage Collector na JVM?

12. Cite três linguagens de programação além do Java que suportam o paradigma orientado a objetos.

13. Explique o que é o bytecode e onde ele é executado.

14. Descreva o processo de instalação do JDK no Windows.

15. Qual é o objetivo do Java Runtime Environment (JRE)?

16. Qual a principal diferença entre um editor de código e uma IDE?

17. Quais são os principais passos para instalar o Eclipse no Windows?

18. Liste pelo menos três editores de código compatíveis com Java.

19. Explique o conceito de Coletor de Lixo em Java.

20. O que diferencia um interpretador de um compilador?

21. Explique como a JVM traduz bytecode em código nativo.

22. Descreva as funções do JIT Compiler e como ele melhora o desempenho do Java.

23. Compare os paradigmas de programação funcional e orientada a objetos.

24. Por que o Java é considerado uma linguagem robusta? Cite três razões.

25. Diferencie o Garbage Collector e o JIT Compiler em termos de funcionalidade.

26. Liste e descreva os principais paradigmas de programação suportados por Java.

27. Quais são as etapas realizadas pelo **Class Loader** ao carregar classes?

28. Explique a relação entre JDK, JRE e JVM.

29. Quais são os benefícios e limitações do uso de bytecode no Java?

30. Por que é importante configurar variáveis de ambiente ao instalar o JDK?

31. Liste e explique as principais diferenças entre a instalação do JDK e do Eclipse no Windows.

32. Qual a importância do **Just-In-Time Compiler (JIT)** na execução de aplicações Java?

33. Descreva o fluxo de execução de um programa Java, desde a compilação até a execução.

34. O que é o **Garbage Collector** e como ele ajuda na gestão de memória?

35. Liste e explique três funções importantes do JDK no desenvolvimento Java.

36. Como o paradigma orientado a objetos influencia o design de software em Java?

37. Explique os diferentes tipos de coletor de lixo disponíveis na JVM.

38. Quais são os benefícios de usar uma IDE como o Eclipse para desenvolvimento em Java?

39. Explique as funções do interpretador na JVM.

40. Descreva como a JVM interage com o sistema operacional para executar um programa.

41. Como o JIT identifica os "hotspots" em um programa?

42. Compare o **Class Loader** e o **Bytecode Verifier** na JVM.

43. Liste os componentes principais da JVM e explique suas funções.

44. Explique a diferença entre **compilação antecipada** e **compilação Just-In-Time**.

45. Como o Eclipse gerencia projetos Java em comparação com um editor de código simples?

46. Por que a JVM é importante para a portabilidade do Java?

47. Descreva como o bytecode do Java é verificado antes da execução.

48. Explique como o Garbage Collector melhora o gerenciamento de memória em relação à alocação manual.

49. Qual é a principal função da **Stack** na JVM?

50. Quais são os principais desafios enfrentados pelo Garbage Collector em aplicações de larga escala?

51. Explique os diferentes tipos de JIT Compilers disponíveis na JVM e suas características.

52. Investigue os algoritmos de Garbage Collection em Java e compare pelo menos dois deles.

53. Desenvolva um programa em Java que simula o comportamento de um coletor de lixo simples.

54. Analise o impacto do uso de bytecode em termos de desempenho e segurança.

55. Explique como as otimizações feitas pelo JIT Compiler podem impactar negativamente em certos casos.

56. Discuta os desafios de implementar a JVM para novas arquiteturas de hardware.

57. Implemente um algoritmo em Java que demonstre a diferença entre memória alocada na heap e na stack.

58. Como a arquitetura modular do Java 9 (Jigsaw) impacta o JDK, JRE e JVM?

59. Desenvolva um programa em Java que provoque e monitore a execução do Garbage Collector.

60. Explique o processo de inicialização do **Class Loader** e como ele resolve dependências.

61. Compare linguagens que compilam para bytecode (como Java e Kotlin) com linguagens puramente compiladas.

62. Proponha melhorias no processo de coleta de lixo para um cenário de baixa latência.

63. Explique como a JVM gerencia threads em sistemas multicore.

64. Implemente um exemplo em Java que explore os limites de otimização do JIT.

65. Pesquise sobre G1 Garbage Collector e desenvolva um resumo técnico de seu funcionamento.

66. Analise os desafios de balancear entre **throughput** e **latência** no Garbage Collection.

67. Explique como a JVM lida com exceções durante a execução de bytecode.

68. Desenvolva um programa Java que explore o uso avançado de reflexões e analise seu impacto na JVM.

69. Discuta a evolução da JVM e como novos recursos, como o Graal VM, têm impactado o desenvolvimento em Java.

70. Analise criticamente os paradigmas de programação suportados pelo Java e como eles se integram na prática.

# MÓDULO 3: NOÇÕES BÁSICAS

**Ser criativo é reutilizar aquilo que todos viram em algo que nunca ninguém pensou.**
Profª Isabel Maria Peres Manso

- CONCEITOS BÁSICOS DA LINGUAGEM
- ALGORITMOS
- FLUXOGRAMA
- PSEUDOCÓDIGO
- O FAMOSO HELLO WORLD
- PARADIGMAS DE PROGRAMAÇÃO
- LÓGICA MATEMÁTICA
- EXERCÍCIOS

# Objetivos

- Aprender o conceito de algoritmo e suas características fundamentais
- Como resolver problemas através de algoritmos
- Como desenhar algoritmos e programas
- Distinguir as diferentes representações de algoritmos
- Como usar fluxogramas em programação visual
- Aprender a utilizar programação estruturada
- Conhecer mas sobre lógica matemática

# Introdução a Lógica de Programação

A lógica de programação e os algoritmos são os dois pilares fundamentais no desenvolvimento de software, essenciais para a criação de soluções eficientes e funcionais.
Esses conceitos formam a base do pensamento computacional, permitindo que programadores construam sistemas capazes de executar tarefas desde as mais simples até as mais complexas.
Neste capítulo, vou apresentar uma introdução à lógica de programação, para que você compreenda de uma vez por todas o que é e por que é importante dominá-la para o desenvolvimento dos seus códigos.
A lógica de programação funciona basicamente como a “receita” do seu código. É nela que você planeja a sequência de ações que deseja executar, estabelecendo um passo a passo para atingir o resultado desejado, aqui, o foco não é o código em si, mas sim definir o que fazer em cada etapa. Depois de pensada e esquematizada, você usará essa lógica como base para escrever e construir seu código na linguagem que escolher, no nosso caso será aplicado no java.

## O que é lógica de programação?

A lógica de programação é a capacidade que todo programador precisa ter para resolver os problemas que aparecem no dia-a-dia. A capacidade de dividir o problema em partes menores é uma etapa essencial da lógica de programação e precisa ser levada em consideração quando nos deparamos com qualquer exercício/desafio. É nesse ponto que entra o conceito de algoritmo, descrito, geralmente, como uma sequência lógica de ações capaz de resolver um problema, é válido ressaltar, no entanto, que o conceito de algoritmo vai muito além da programação. Uma receita de bolo, por exemplo, é um exemplo simples de algoritmo.
Até mesmo algo como “Fazer sumo de laranja” pode ser descrito como um algoritmo.
A maioria dos algoritmos, obviamente, será mais complexa do que simplesmente “somar dois números” ou “fazer um bolo”. No caso da computação, uma soma de dois números é o exemplo mais clássico de

um algoritmo simples.  é bastante comum para observarmos que temos uma sequência lógica de ações que envolvem os três elementos essenciais:

**entrada de dados**, **processamento e saída de dados**.

Exemplo: Algoritmo para soma de dois números

1 – Inserir o primeiro número
2 – Inserir o segundo número
3 – Somar os dois valores
4 – Mostrar o resultado

## Necessidade do Uso da Lógica

Usar a lógica é um fator a ser considerado por todos, principalmente pelos profissionais da área da Tecnologia de Informação (programadores, analistas de sistemas e suporte), pois seu dia-a-dia dentro das organizações é solucionar problemas e atingir os objetivos apresentados por seus usuários com eficiência e eficácia, utilizando recursos computacionais e automatizados mecatrônica mente, saber lidar com problemas de ordem administrativa, de controle, de planejamento e de estratégia requer atenção e boa performance de conhecimento de nosso raciocínio, porém, é necessário considerar que ninguém ensina ninguém a pensar, pois todas as pessoas normais possuem esse “dom”.
O objetivo deste capítulo é mostrar como desenvolver e aperfeiçoar melhor essa técnica, lembrando que para isso você deve ser persistente e praticá-la constantemente, chegando quase à exaustão sempre que julgar necessário.

# Raciocínio lógico

Muitas pessoas gostam de falar ou julgar que possuem e sabem usar o raciocínio lógico, porém, quando questionadas direta ou indiretamente, perdem esta linha de raciocínio, pois este depende de inúmeros fatores para completá-lo, tais como: calma, conhecimento, vivência, versatilidade, experiência, criatividade, ponderação, responsabilidade, entre outros.

Para usar a lógica, é necessário ter domínio sobre o pensamento, bem como saber pensar, ou seja, possuir a “Arte de Pensar”. Alguns definem o raciocínio lógico como um conjunto de estudos que visa determinar os processos intelectuais que são as condições gerais do conhecimento verdadeiro.

Isso é válido para a tradição filosófica clássica aristotélico-tomista, outros preferem dizer que é a sequência coerente, regular e necessária de acontecimentos, de coisas ou fatos, ou até mesmo, que é a maneira do raciocínio particular que cabe a um indivíduo ou a um grupo, estas são algumas definições encontradas no dicionário Aurélio, mas existem outras que expressam o verdadeiro raciocínio lógico dos profissionais da área de Tecnologia da Informação, tais como: um esquema sistemático que define as interações de sinais no equipamento automático do processamento de dados, ou o computador científico com o critério e princípios formais de raciocínio e pensamento.

Para concluir todas estas definições, pode-se dizer que lógica é a ciência que estuda as leis e critérios de validade que regem o pensamento e a demonstração, ou seja, ciência dos princípios formais do raciocínio.

# Aplicabilidade da Lógica no Desenvolvimento de Programas

Muitos programadores (principalmente os mais antigos profissionais desta área) preferem preparar um programa iniciando com um diagrama de blocos para demonstrar sua linha de raciocínio lógico.
Esse diagrama, também denominado por alguns de fluxograma, estabelece a sequência de operações a se efetuar em um programa.
Essa técnica permite uma posterior codificação em qualquer linguagem de programação de computadores, pois na elaboração do diagrama de blocos não se atinge um detalhamento de instruções ou comandos específicos, os quais caracterizam uma linguagem.

A técnica mais importante no projeto da lógica de programas é chamada programação estruturada, a qual consiste em uma metodologia de projeto, objetivando:

- Agilizar a codificação da escrita de programas;
- Facilitar a depuração da sua leitura;
- Permitir a verificação de possíveis falhas apresentadas pelos programas;
- Facilitar as alterações e atualizações dos programas.

E deve ser composta por quatro passos fundamentais:

- Escrever as instruções em sequências ligadas entre si apenas por estruturas sequenciais, repetitivas ou de seleção.
- Escrever instruções em grupos pequenos e combiná-las.
- Distribuir módulos do programa entre os diferentes programadores que trabalharão sob a supervisão de um programador sênior, ou chefe de programação.
- Revisar o trabalho executado em reuniões regulares e previamente programadas, em que compareçam programadores de um mesmo nível.

# Conceito de Algoritmos

Um algoritmo é formalmente uma sequência finita de passos que levam a execução de uma tarefa.
Podemos pensar em algoritmos como uma receita, uma sequência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.
Como exemplos de algoritmos podem citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais.
O termo é muito utilizado na matemática e na computação, em ambas, os conceitos são semelhantes, na primeira área, está associado a um processo de cálculo, o encadeamento de ações necessárias para o cumprimento de uma tarefa, em suma, é o processo efetivo, que produz solução para um problema em um número finito de etapas.
Já na informática: Manzano no seu livro algoritmos define “algoritmo é um conjunto de regras e procedimentos lógicos perfeitamente definidos que levam à solução de um problema em um número finito de etapas”.
Isto é, os algoritmos são sequências finitas de instruções, utilizadas a fim de resolver um problema. Por exemplo, quando você acessa um site, os algoritmos definem o caminho para a correta abertura da página, quando você interage com um link, outros algoritmos são acionados, indicando o que fazer.

# Formas de Representação de Algoritmos

Existem diversas formas de representação de algoritmos, mas não há um consenso com relação à melhor delas.
O critério usado para classificar hierarquicamente estas formas está diretamente ligado ao nível de detalhe ou, inversamente, ao grau de abstração oferecido.
Algumas formas de representação de algoritmos tratam os problemas apenas em nível lógico, abstraindo-se de detalhes de implementação muitas vezes relacionados com alguma linguagem de programação específica, por outro lado, existem outras formas de representação de algoritmos que possuem uma maior riqueza de detalhes e muitas vezes acabam por obscurecer as ideias principais do algoritmo, dificultando seu entendimento.

Os quatro tipos de representação mais comuns dos algoritmos são:

- Descrição narrativa;
- Fluxograma;
- Pseudocódigo, também conhecido como Linguagem Estruturada ou Portugol;
- Linguagem de programação.

Entenda cada um deles em detalhes:

# Descrição narrativa

O algoritmo é representado textualmente, elencando a sequência de instruções, a fim de resolver um problema. Isto é, a pessoa o escreve com suas próprias palavras, um exemplo de descrição narrativa é a receita culinária, nela, o autor diz textualmente ao cozinheiro como cada etapa deve ser cumprida, a fim de preparar uma refeição.
A principal desvantagem deste tipo de representação de algoritmo é a presença de ambiguidades, e ruídos de comunicação.

O texto pode ser interpretado de maneiras diferentes, dificultando a resolução do problema.

Estrutura Geral

1. **Objetivo Geral**: O que o algoritmo pretende alcançar.
2. **Entrada**: O que é necessário para o algoritmo funcionar
3. **Processamento**: Passos sequenciais detalhados.
4. **Saída**: O resultado final ou as informações produzidas.

Exemplo Prático
Nesta forma de representação os algoritmos são expressos diretamente em linguagem natural.
Como exemplo, seguinte:

Algoritmo 1 Troca de pneu do carro.

**1: desligar o carro**

**2: pegar as ferramentas (chave e macaco)**

**3: pegar o estepe**

**4: suspender o carro com o macaco**

**5: desenroscar os 4 parafusos do pneu furado**

**6: colocar o estepe**

**7: enroscar os 4 parafusos**

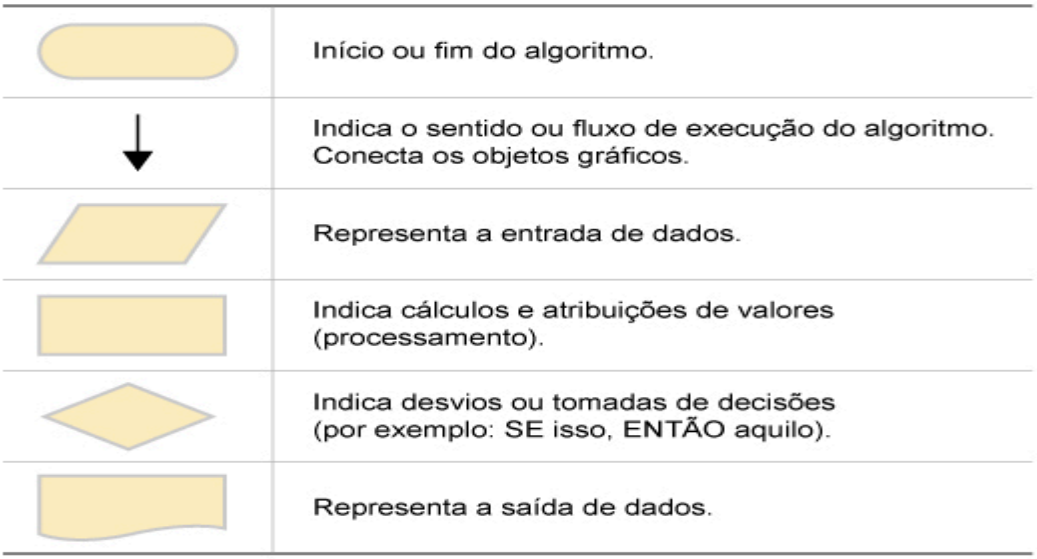
**8: baixar o carro com o macaco**

**9: guardar as ferramentas**

Esta representação é pouco usada na prática porque o uso da linguagem natural muitas vezes dá oportunidade a más interpretações, ambiguidades e imprecisões, por exemplo, a instrução "afrouxar ligeiramente as porcas" no algoritmo da troca de pneus está sujeita a interpretações diferentes por pessoas distintas.

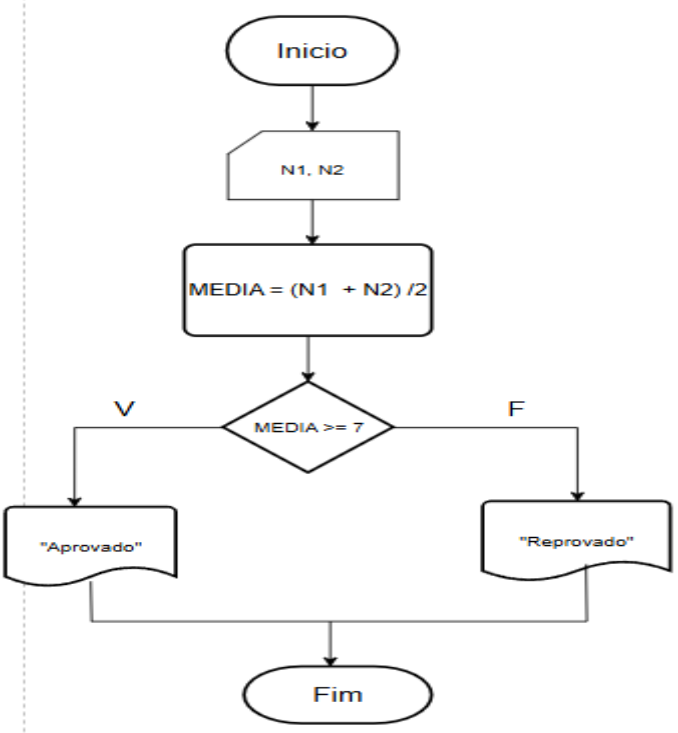
# Fluxograma

É uma representação gráfica de algoritmos onde formas geométricas diferentes implicam ações (instruções, comandos) distintos.
Esta forma é aproximadamente intermediária à descrição narrativa e ao pseudocódigo, pois é menos imprecisa que a primeira e, no entanto, não se preocupa com detalhes de implementação do programa, como o tipo das variáveis usadas.
Nota-se que os fluxogramas convencionais se preocupam com detalhes de nível físico da implementação do algoritmo. Por exemplo, figuras geométricas diferentes são adotadas para representar operações de saída de dados realizadas em dispositivos distintos, como uma fita magnética ou um monitor de vídeo. Como esta apostila não está interessada em detalhes físicos da implementação, mas tão somente com o nível lógico das instruções do algoritmo.




De modo geral, um fluxograma se resume a um único símbolo inicial por onde a execução do algoritmo começa, e um ou mais símbolos finais, que são pontos onde a execução do algoritmo se encerra.
Partindo do símbolo inicial, há sempre um único caminho orientado a ser seguido, representando a existência de uma única sequência de execução das instruções.

Isto pode ser bem visualizado pelo fato de que, apesar de vários caminhos poderem convergir para uma mesma figura do diagrama, há sempre um único caminho saindo desta.
Exceções a esta regra são os símbolos finais, dos quais não há nenhum fluxo saindo, e os símbolos de decisão, de onde pode haver mais de um caminho de saída (usualmente dois caminhos), representando uma bifurcação no fluxo.
A Figura 2.2 mostra a representação do algoritmo de cálculo da média de um aluno sob a forma de um fluxograma.




# Pseudocódigo

Pseudocódigo é um método de descrever um processo ou escrever código de programação e algoritmos usando uma linguagem natural, como o inglês. Ou seja, o pseudocódigo não é o código em si, mas sim uma descrição do que o código deve fazer. Ele é usado como um plano ou passo a passo detalhado, mas compreensível, a partir do qual um programa pode ser escrito.

Essa palavra “pseudocódigo” é uma junção de duas palavras: pseudo e código. “Pseudo” é algo cujo conteúdo não é real ou verdadeiro. Dessa forma, o pseudocódigo nada mais é do que um rascunho de um programa ou algoritmo antes de ser implementado em uma linguagem de programação.

Alguns pontos importantes que você precisa saber sobre pseudocódigos:

- Não é uma linguagem de programação;
- É apenas uma ferramenta de aprendizado e raciocínio, usada por pessoas programadoras para sublinhar como escrever o código real;
- O pseudocódigo não pode ser executado ou compilado por nenhum compilador, interpretador ou montador;
- Ao contrário do código da linguagem de programação, o pseudocódigo não segue uma estrutura e sintaxe rígidas. A pessoa programadora pode escrever a sintaxe do pseudocódigo como quiser;

Os comandos básicos para usar em pseudocódigos
A seguir, vejamos algumas das principais instruções utilizadas nos pseudocódigos:

| COMANDO | UTILIZADO PARA |
|---|---|
| escreva (“ ”) | Mostrar uma mensagem para a pessoa que estiver executando o programa no computador. |
| leia ( ) | Receber alguma informação digitada pela pessoa que está executando o programa. |
| início | Começar o funcionamento de seu algoritmo. Tudo que estiver acima desta instrução, não será executado. |
| fimalgoritmo | Finalizar o funcionamento de seu algoritmo. Tudo |

| | |
|---|---|
| | que estiver abaixo dessa instrução, não será executado. |
| var | Armazenar variáveis em seu programa. |
| <- | Atribuir um valor para um espaço que você criou. |
| + | Somar dois valores. |
| – | Subtrair dois valores. |
| real | Referir-se ao conjunto dos números reais. |

## Quais as ferramentas para criar um pseudocódigo?

A escrita manual, em um papel, pode ser um bom começo, mas, depois de um tempo de prática, é igualmente importante migrar para um software. A seguir, confira as principais alternativas.

**VisuAlg**

O VisuAlg é a principal opção para a interpretação de algoritmos em Portugol. É uma boa alternativa para iniciantes e não apenas para escrever pseudocódigo, mas também para entender melhor como ele é executado.

**Portugol Studio**

O Portugol Studio é outra alternativa para quem quer aprender programação e fala português. Com uma sintaxe baseada nas linguagens de programação C e PHP, conta com diversos exemplos e materiais de apoio e permite até mesmo a criação de jogos.

O pseudocódigo possibilita rapidez na escrita sem se incomodar com os rigores técnicos e sintáticos particulares de cada uma das linguagens de programação existentes. Além disso, por meio dele, fica mais fácil traduzir as informações posteriormente para uma dessas linguagens.

# Linguagem de programação

Se você tem a solução para um problema em forma de texto, fluxograma ou pseudocódigo, e quer aplicá-la em um software, será necessário traduzir este algoritmo para a linguagem utilizada pelo programa.
Dessa forma, além de ter o passo a passo da solução bem definido, você deverá estar por dentro da sintaxe da linguagem de programação utilizada, sabendo como expressar seu algoritmo por meio dela.
Mas não se preocupe agora com a questão de exemplos numa linguagem de programação, neste livro, já a partir deste módulo teremos tempo o suficiente para aprender sobre uma linguagem de programação, e se familiarizar com a semântica e sintaxe da linguagem java e todo seu conceito básico, resolução de problemas computacionais e reais usando a linguagem java.

# Comentários

O desenvolvimento é uma prática compartilhada e por isso é importante deixar instruções claras sobre a estrutura, construção e funcionamento dos códigos. Para que esses procedimentos sejam harmoniosos, o papel dos comentários em Java é fundamental.

## O que são comentários e por que são importantes em Java?

Comentários em Java são anotações textuais inseridas entre linhas de código para fornecer indicações e esclarecimentos sobre o projeto.
Em nenhum caso afetam a execução do programa, o compilador e a máquina virtual Java os ignoram, mas são visíveis para que qualquer desenvolvedor possa lê-los e levá-los em consideração.

## Quais são os propósitos dos comentários em Java?

Eles podem servir como um lembrete para o próprio desenvolvedor, facilitar a colaboração entre vários desenvolvedores e garantir que o código seja legível. Além disso, também servem para desabilitar um bloco de código sem excluí-lo completamente.

## Tipos de comentários em Java

Java suporta vários tipos de comentários:

- Comentário de uma linha
- Comentários de várias linhas
- Comentários Javadoc

## Comentário de uma linha

Ao escrever o código, pode surgir a necessidade de incluir breves anotações para fornecer instruções simples.

Eles ocupam menos de uma linha, por isso são conhecidos como comentários de linha, para incluí-los no código, a primeira coisa que você precisa fazer é abrir a linha com duas barras (//) e depois incluir o comentário.
Lembre-se que o compilador irá ignorar o texto e que ele será utilizado para esclarecimentos internos.

Exemplo de um comentário de uma linha:

```
// Isto é um comentário de uma linha
```

## Comentários de várias linhas

Quando as indicações ocupam mais de uma linha de código em Java, elas são conhecidas como comentários em bloco, além de fornecer explicações mais extensas, elas podem ser usadas para desabilitar temporariamente partes do código, por qualquer motivo.
Para iniciar um comentário em bloco, você deve abrir a anotação com uma barra e um asterisco (/*), depois de inserir o texto, você pode fechar a mensagem com um asterisco e uma barra (*/). Tal como acontece com os comentários de linha, este conteúdo não influenciará a execução do código.

```
/*
  Isto é um comentário
  que se estende por
  várias linhas.
*/
```

## Comentários Javadoc

Os comentários Java não são usados apenas para dar instruções a outros humanos, eles também podem dar instruções ao próprio sistema.

É aqui que entram os comentários JavaDoc, onde o desenvolvedor inclui uma série de tags para gerar automaticamente documentação HTML técnica e legível a partir do código-fonte.
Para inserir um comentário JavaDoc, você deve abrir o prompt com uma barra e dois asteriscos (/**) e depois de inserir o comentário e as tags apropriadas, você terá que fechá-lo com um asterisco e uma barra (*/). Tags abrem com arroba (@) e as mais populares são @return, @param e/ou @exception.

```
/**
* Isto é um comentário Javadoc
* @param meuParametro Descrição do parâmetro
* @return Descrição do valor retornado
*/
```

**Nota Importante:**
Atenção! Use Comentários com Moderação

Os comentários são importantes para explicar o "porquê" de uma decisão de código, mas não o "o quê" ou "como".
O código bem escrito deve ser autoexplicativo na maior parte do tempo. Use comentários para:

• Explicar decisões de design complexas.

• Fornecer informações contextuais importantes.

• Documentar comportamentos esperados em casos complexos.

## Melhores práticas para escrever comentários eficazes

O código Java, acima de tudo, deve ser legível por qualquer profissional que o assuma. Por isso é fundamental seguir algumas práticas para que este princípio seja cumprido e fornecer comentários eficazes. Anote os principais!

- Síntese e clareza, não faça rodeios, tempo é dinheiro e seus comentários devem ser entendidos à primeira vista.

- Inclua o quê e porquê, inclua a finalidade e os motivos pelos quais o código precisa ser aplicado.

- Contribua, não repita, incluir comentários Java quando necessário, encontrar diversas anotações repetidas torna o trabalho mais tedioso.

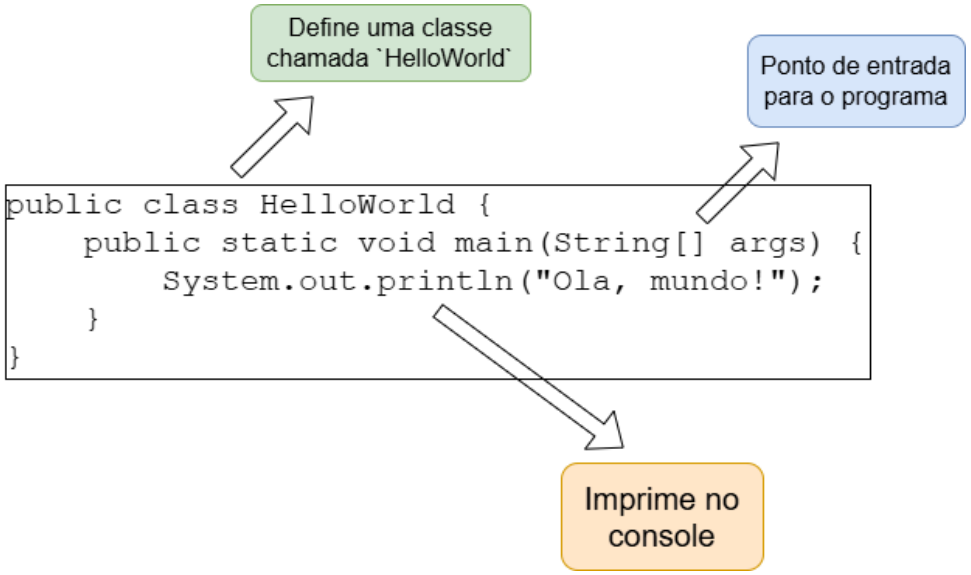
- Explicação de fórmulas, quando é necessária uma fórmula matemática que não é óbvia, é interessante que você explique os passos para resolvê-la.

- Profissionalismo, embora os comentários em Java não sejam compilados pela máquina virtual nem sejam de domínio público, é aconselhável manter alguns padrões profissionais nas anotações. Você nunca sabe quem vai ler aquele insulto velado ao seu melhor amigo do escritório e rezar para que não seja um cliente.

# O famoso Hello World

Para iniciar a nossa jornada de aprendizado em Java, vamos criar e executar um simples programa “Hello World”  neste capítulo, vamos começar já através dos passos para criar um programa Java que exibe a clássica mensagem “Hello World”.

Estrutura Básica
Todo programa Java começa com a criação de um arquivo de código-fonte Java, vamos criar um arquivo chamado HelloWorld e abrir o editor de texto de sua preferência para começar a escrever nosso código.
Dentro do arquivo HelloWorld.java, escreva o seguinte código:




Explicação:

- Class: é utilizada para declarar uma classe em Java.
- public: é um modificador de acesso que representa a visibilidade. Significa que é visível para todos.
- static é uma palavra-chave, se declararmos um método como estático, é conhecido como método estático.

A principal vantagem do método estático é que não é necessário criar um objeto para invocar o método estático, o método main() é executado pela JVM, pelo que não é necessário criar um objeto para invocar o método main(). assim, poupa memória.

- void é o tipo de retorno do método. Isso significa que ele não retorna nenhum valor.
- O método main() representa o ponto inicial do programa.
- String[] args ou String args[] é usado para argumento de linha de comando. Discutiremos isso na próxima seção.
- System.out.println() é usado para imprimir declaração no console. Aqui, System é uma classe, out é um objeto da classe PrintStream, println() é um método da classe PrintStream. Discutiremos o funcionamento interno da declaração System.out.println() na próxima seção.

Após executar este comando, você verá a mensagem "**Ola, mundo**" impressa no terminal.

# Como um programa é executado?

Modelo Tradicional de Execução
O compilador é responsável por traduzir o código-fonte escrito por um desenvolvedor (em uma linguagem como C ou Pascal, por exemplo) em um **código binário**. Esse código binário é específico para o sistema operacional e o processador da máquina em que será executado.
O resultado desse processo é um **arquivo executável**, como os arquivos **.exe** no Windows.

## Dependência do Sistema Operacional

O código binário gerado depende das bibliotecas e da estrutura do sistema operacional onde será executado. Por exemplo:
Um programa compilado para o Windows não funciona no Linux ou macOS. Bibliotecas de sistemas operacionais diferentes podem ter nomes, funções ou comportamentos distintos, exigindo adaptações no código para cada ambiente.

### Impactos na Portabilidade

Para rodar o mesmo programa em diferentes sistemas operacionais, era necessário:

1. Passar o código-fonte por um **compilador específico** para o sistema de destino.
2. Adaptar o código para usar bibliotecas compatíveis com aquele sistema.

Isso resultava em maior esforço e tempo de desenvolvimento, além de problemas de compatibilidade que não garantiam o funcionamento correto do programa em todos os sistemas.

### Desafios para Desenvolvedores e Usuários

Os desenvolvedores gastavam muito tempo adaptando o código para cada plataforma. Usuários poderiam não ter acesso a determinados programas devido à falta de compatibilidade com seu sistema operacional.

## A Evolução: Máquina Virtual e Bytecode

Para resolver o problema da portabilidade, um grupo de cientistas liderado por **Alan Kay** idealizou um novo modelo, introduzindo uma plataforma intermediária capaz de executar o mesmo código em diferentes sistemas.

Essa plataforma intermediária seria uma máquina virtual (VM) que traduziria o código para o sistema operacional subjacente.

# Como funciona?

O código-fonte é compilado, mas ao invés de gerar um executável binário específico para um sistema operacional, ele é transformado em um código intermediário chamado bytecode. O bytecode não é executado diretamente pelo sistema operacional, mas sim pela máquina virtual, que atua como uma camada intermediária entre o bytecode e o sistema operacional.

## O que é Bytecode?

O bytecode é um código intermediário gerado pela compilação do código-fonte, ele é otimizado para ser interpretado por máquinas virtuais como a JVM, como é um arquivo binário, não é legível por humanos, mas pode ser inspecionado com ferramentas específicas.

Compilando para Bytecode:
Use o comando javac Hello.java no terminal.
Isso gera um arquivo Hello.class, que contém o bytecode.
Visualizando o Bytecode:
Para inspecionar o conteúdo do bytecode, utilize o javap, a ferramenta de desmontagem do Java:

```
javap -c Hello.class
```

Isso exibe o bytecode em formato legível, mostrando as instruções que a JVM executará.

# Indentação

A indentação em Java é uma prática fundamental para manter o código limpo, legível e bem organizado. Ela envolve o uso de espaços ou tabulações para alinhar o código de maneira hierárquica, refletindo a estrutura lógica do programa.

## Por que a indentação é importante?

Legibilidade: Código bem indentado é mais fácil de entender.
Manutenção: Facilita a identificação de blocos de código relacionados.
Padronização: Seguir convenções de estilo, como as definidas pelo Java Code Conventions ou por ferramentas como Google Java Style Guide, ajuda em equipes de desenvolvimento.

## Regras Gerais de Indentação em Java

Geralmente, usa-se 4 espaços por nível de indentação, não misture espaços e tabulações. Escolha um e mantenha a consistência.
Seguir boas práticas de indentação em Java é essencial para manter o código claro e compreensível, tanto para você quanto para outros desenvolvedores.
Pratique a consistência e adote ferramentas automatizadas para garantir a qualidade do código.

# Palavras-chave em Java

A estrutura do java é organizada e padronizada, o que a torna robusta e segura para o desenvolvimento de aplicações. Um dos elementos fundamentais dessa estrutura são as palavras reservadas, que têm funções específicas e não podem ser utilizadas como identificadores.
Aqui vamos explorar palavras reservadas do Java, organizando-as por categorias e oferecendo uma breve descrição de cada uma. Isso ajudará você a entender melhor como elas funcionam e como podem ser usadas em seus programas.

## Palavras-chave de Controle de Fluxo

- **if:** Inicia uma estrutura condicional, executando um bloco de código se a condição especificada for verdadeira.
- **else:** Usado em conjunto com if, executa um bloco de código alternativo se a condição if for falsa.
- **switch:** Utilizado para executar um de vários blocos de código, dependendo do valor de uma expressão.
- **case:** Define um caminho específico em um bloco switch.
- **default:** Especifica o bloco de código a ser executado se nenhum caso for atendido.
- **for:** Um laço que executa repetidamente um bloco de código enquanto uma condição é verdadeira.
- **while:** Outro tipo de laço que continua a execução enquanto a condição especificada for verdadeira.

- **do:** Similar ao while, mas garante que o bloco de código seja executado pelo menos uma vez.
- **break:** Interrompe o laço ou switch em execução.
- **continue:** Faz o laço pular o restante de seu corpo e ir para a próxima iteração.
- **return:** Finaliza a execução de um método e, opcionalmente, retorna um valor.

## Palavras-chave de Modificadores de Acesso e Controle

- **public:** Torna a classe, método ou variável acessível de qualquer outro lugar.
- **private:** Restringe o acesso à classe, método ou variável somente à classe onde foi definido.
- **protected:** Permite o acesso a partir de subclasses e classes do mesmo pacote.
- **Static:** Associa o método ou variável à classe, em vez de às instâncias dela.
- **final:** Impede que uma variável seja modificada, que um método seja sobrescrito ou que uma classe seja herdada.
- **abstract:** Define classes que não podem ser instanciadas e métodos que devem ser implementados por subclasses.
- **synchronized:** Garante que um método ou bloco de código seja acessado por apenas um thread por vez.
- **volatile:** Indica que o valor de uma variável pode ser alterado por diferentes threads, garantindo sua leitura sempre da memória principal.
- **transient:** Exclui um campo da serialização.
- **native:** Especifica que um método é implementado em código nativo fora do ambiente Java.
- **strictfp:** Garante que as operações de ponto flutuante sigam as regras estritas do IEEE 754.

## Palavras-chave Relacionadas a Classes e Objetos

- **class:** Define uma nova classe, o modelo para objetos.
- **interface:** Define um contrato que outras classes devem implementar.

- **Enum:** Declara um tipo enumerado, representando um conjunto fixo de constantes.
- **extends:** Indica que uma classe herda de outra.
- **implements:** Indica que uma classe implementa uma interface.
- **new:** Cria uma nova instância de uma classe.
- **this:** Refere-se à instância atual da classe.
- **super:** Refere-se à superclasse imediata.
- **instanceof:** Verifica se um objeto é uma instância de uma classe específica.

## Palavras-chave de Tratamento de Exceções

- **try:** Inicia um bloco de código que pode lançar exceções.
- **catch:** Captura e trata uma exceção lançada.
- **finally:** Executa um bloco de código independentemente do que aconteça no try/catch.
- **throw:** Lança explicitamente uma exceção.
- **throws:** Declara que um método pode lançar exceções.

## Palavras-chave Relacionadas a Tipos de Dados

- **int:** Define números inteiros de 32 bits.
- **byte:** Define números inteiros de 8 bits.
- **short:** Define números inteiros de 16 bits.
- **long:** Define números inteiros de 64 bits.
- **float:** Define números de ponto flutuante de 32 bits.
- **double:** Define números de ponto flutuante de 64 bits.
- **char:** Define um caractere Unicode de 16 bits.
- **boolean:** Define valores true ou false.
- **void:** Indica que um método não retorna um valor.

## Palavras-chave de Controle de Pacotes e Importação

- **package:** Define a que pacote uma classe pertence.
- **import:** Inclui classes e interfaces de outros pacotes.

Essas palavras-chave são a base de muitos conceitos fundamentais no Java. Entender o papel de cada uma delas é essencial para dominar a linguagem e escrever código eficiente e seguro.

# LÓGICA MATEMÁTICA

A lógica iniciou seu desenvolvimento na Grécia Aristóteles (384 – 322 AC) e os antigos filósofos gregos passaram a usar em suas discussões sentenças enunciadas nas formas afirmativa e negativa, resultando em grande simplificação e clareza. Em 1847, Augustus DeMorgam (1806-1871) publicou o tratado Formal Logic. Em 1848, George Boole (1815-1864) escreveu The Mathematical Analysis of Logic e depois publicou um livro sobre o que foi denominado posteriormente de Álgebra de Boole. Em 1879, Gotlob Frege (1848-1925) contribuiu no desenvolvimento da lógica com a obra Begriffsschrift. As ideias de Frege só foram reconhecidas pelos lógicos mais ou menos a partir de 1905. A escola italiana, que desenvolveu quase toda simbologia da matemática utilizada atualmente, é composta de Giuseppe Peano (1858-1932) e também por Burali-Forti, Vacca, Pieri, Pádoa, Vailati, etc.

Bertrand Russell (1872-1970) e Alfred North Whitehead (1861-1947) iniciam o atual período da lógica com a publicação da obra Principia Mathematica no início do século XX. Também contribuem para o estágio atual, David Hilbert (1862-1943) e sua escola alemã com von Neumann, Bernays, Ackerman e outros. Podemos dizer que a lógica estuda as condições objetivas e ideais para justificar a verdade e não cuida da própria verdade. Ela estuda as condições formais para justificar a verdade, isto é, as condições que o pensamento deve preencher para ser coerente consigo mesmo e demonstrar a verdade já conhecida. A lógica estuda as relações do pensamento consigo mesmo para possibilitar a construção de um contexto correto de justificação, isto é, para a definição argumentativa de premissas corretamente dispostas para uma conclusão justificada.

Definições

Lógica é uma palavra que define a ciência do raciocínio. Outro conceito de lógica é “o estudo dos métodos e princípios usados para distinguir o raciocínio correto do incorreto”. Essa ciência abrange vários conceitos, dentre eles a argumentação, a matemática e a informática.

# Proposições

As proposições são palavras ou símbolos que expressam um pensamento com um sentido completo e indicam afirmações de fatos ou de ideias.
Essas afirmações assumem valores lógicos que podem ser verdadeiros ou falsos e para representar uma proposição usualmente utilizamos as letras p e q.

Considerando a lógica matemática, uma proposição não pode ser ao mesmo tempo verdadeira e falsa. Além disso, não existe a possibilidade de uma terceira situação diferente de verdadeiro ou falso.
As proposições podem ser simples, quando apresentam apenas uma sentença, e compostas quando são formadas pela combinação de duas ou mais proposições simples. "O céu é azul" é um exemplo de proposição simples, já a sentença "O céu é azul e as nuvens são brancas" é um exemplo de proposição composta.

# Conectivos

As proposições simples que formam uma proposição composta são ligadas por elementos que são chamados de conectivos. Além disso, também podemos utilizar conectivos para modificar uma proposição.
Na proposição "O céu é azul e as nuvens são brancas" o elemento é um conectivo que une duas proposições, já na proposição "O céu não é azul" o conectivo não modifica a proposição.

# Tabela Verdade

Quando temos proposições compostas, os valores lógicos resultantes dependem única e exclusivamente dos valores de cada proposição simples. Diante disso, utilizamos um dispositivo chamado tabela verdade ou tabela de verdade, onde são colocados os valores de cada proposição e de acordo com os conectivos presentes chegamos ao valor lógico final.

Em uma tabela verdade, o número de linhas e de colunas dependerá da quantidade de proposições simples que formam a proposição composta, sendo que em cada coluna é colocada uma proposição.
Abaixo apresentamos a tabela verdade para duas, três e quatro proposições:

| p | q | r |
|---|---|---|
| v | v | v |
| v | v | f |
| v | f | v |
| v | f | f |
| f | v | v |
| f | v | f |
| f | f | v |
| f | f | f |

# Operações Lógicas

As operações feitas a partir de proposições são chamadas de operações lógicas. Este tipo de operação segue as regras do chamado cálculo proposicional.
As operações lógicas fundamentais são: negação, conjunção, disjunção, condicional e bicondicional.

# Negação

Esta operação representa o valor lógico oposto de uma dada proposição. Desta forma, quando uma proposição é verdadeira, a não proposição será falsa.

Com o objetivo de indicar a negação de uma proposição colocamos o símbolo ~ na frente da letra que representa a proposição, assim, ~p significa a negação de p.

Exemplo
p: Minha filha estuda muito.
~p: Minha filha não estuda muito.

Como o valor lógico da não proposição é o inverso da proposição, teremos a seguinte tabela verdade:

| p | ~p |
|---|---|
| v | f |
| f | v |

# Conjunção

A conjunção é utilizada quando entre as proposições existe o conectivo e. Esta operação será verdadeira quando todas as proposições forem verdadeiras.
O símbolo utilizado para representar essa operação é o ^, colocado entre as proposições. Desta forma, quando temos p ^ q, significa "p e q". Desta forma, a tabela verdade desse operador lógico será:

| p | q | p^q |
|---|---|---|
| v | v | v |
| v | f | f |
| f | v | f |
| f | f | f |

# Disjunção

Nesta operação, o resultado será verdadeiro quando pelo menos uma das proposições é verdadeira. Sendo assim, será falso apenas quando todas as proposições forem falsas.

A disjunção é usada quando entre as proposições existe o conectivo ou e para representar esta operação é usado o símbolo v entre as proposições, assim, p v q significa "p ou q".

Levando em consideração que se uma das proposições for verdadeira o resultado será verdadeiro, temos a seguinte tabela verdade:

| p | q | p v q |
|---|---|---|
| v | v | v |
| v | f | v |
| f | v | v |
| f | f | f |

# Condicional

A condicional é a operação realizada quando na proposição utiliza-se o conectivo se... então.... Para representar esse operador usamos o símbolo →. Assim, p → q significa "se p, então q".

O resultado desta operação só será falso quando a primeira proposição for verdadeira e a consequente for falsa.

É importante ressaltar que uma operação condicional não significa que uma proposição é a consequência da outra, o que estamos tratando é apenas de relações entre valores lógicos.

Exemplo

Qual o resultado da proposição "Se um dia tem 20 horas, então um ano tem 365 dias"?

Solução

Sabemos que um dia não tem 20 horas, logo essa proposição é falsa, também sabemos que um ano tem 365 dias, logo essa proposição é verdadeira.

Desta forma, o resultado será verdadeiro, pois o operador condicional só será falso quando a primeira for verdadeira e a segunda falsa, que não é o caso.

A tabela verdade para este operador será:

| p | q | p -> q |
|---|---|---|
| v | v | v |
| v | f | f |
| f | v | v |
| f | f | v |

# Bicondicional

O operador bicondicional é representado pelo símbolo seta para a esquerda e para a direita e indica uma proposição do tipo ...se e somente se.... Portanto, p seta para a esquerda e para a direita q significa "p se e somente se q", ou seja, p é condição necessária e suficiente para q.
Ao usar esse operador, a sentença será verdadeira quando as proposições forem ambas verdadeiras ou ambas falsas.

Os possíveis resultados que podemos encontrar ao usar esse operador estão na tabela abaixo:

| p | q | p <-> q |
|---|---|---|
| v | v | v |
| v | f | f |
| f | v | f |
| f | f | v |

# Os Princípios Lógicos

Aristóteles desenvolveu três princípios básicos que orientam a lógica clássica.

## Princípio de identidade

Um ser é sempre idêntico a si mesmo: A é A. Se substituirmos A por Maria, por exemplo, fica: Maria é Maria.

## Princípio da não-contradição

É impossível ser e não ser ao mesmo tempo, ou um mesmo ente ser também o seu oposto. É impossível que A seja A e não-A, ao mesmo tempo. Ou, seguindo o exemplo anterior: é impossível que Maria seja Maria e não seja Maria.

## Princípio do terceiro excluído, ou terceiro excluso

Nas proposições (sujeito e predicado), só existem duas opções, ou é afirmativa ou negativa: A é x ou A é não-x. Maria é professora ou Maria não é professora. Não existe uma terceira possibilidade.

# Lista de Exercícios Propostos

1. Escreva o algoritmo e o pseudocódigo para calcular a média de três números fornecidos pelo usuário.
2. Desenhe o fluxograma do algoritmo acima.
3. Crie um programa Java que imprima o texto "Olá, Mundo!".
4. Liste 5 palavras-chave do Java e explique suas funções.
5. Pesquise e explique o que acontece durante a compilação e execução de um programa Java.
6. Escreva um algoritmo para o problema da troca de um único pneu de um carro.
7. Escreva um algoritmo para o problema de trocar um pneu de uma bicicleta.
8. Para cada um dos problemas propostos a seguir, expresse um algoritmo que pode ser usado em sua solução na forma de fluxograma e pseudocódigo.

- A. Calcule a média de quatro números inteiros dados.
- B. Leia uma temperatura dada na escala Celsius (C) e imprima o equivalente em Fahrenheit (F). (Fórmula de conversão: F = 9/5 * C + 32)
- C. Leia a quantidade de chuva dada em polegadas e imprima o equivalente em milímetros (25,4 mm = 1 polegada).
- D. Calcule o quadrado de um número, ou seja, o produto de um número por si mesmo.
- E. O custo ao consumidor de um carro novo é a soma do custo de fábrica com a percentagem do distribuidor e dos impostos, ambos aplicados ao custo de fábrica. Supondo que a percentagem do distribuidor seja de 12% e a dos impostos de 45%, prepare um algoritmo para ler o custo de fábrica do carro e imprimir o custo ao consumidor.
- F. O cardápio de um fast food é dado abaixo. Prepare um algoritmo que leia a quantidade de cada item que você consumiu e calcule a conta final.

  Hambúrguer..... USD$ 3,00
  Cheeseburger..... USD$ 2,50
  Fritas..... USD$ 2,50

Refrigerante..... USD$ 1,00
Milkshake..... USD$ 3,00

- G. Uma companhia de carros paga a seus empregados um salário de USD$500,00 por mês mais uma comissão de USD$50,00 para cada carro vendido e mais 5% do valor da venda. Elabore um algoritmo para calcular e imprimir o salário do vendedor num dado mês recebendo como dados de entrada o nome do vendedor, o número de carros vendidos e o valor total das vendas.

- H. Calcule a média de um aluno na disciplina de MDS. Para isso solicite o nome do aluno, a nota da prova e a nota qualitativa. Sabe-se que a nota da prova tem peso 2 e a nota qualitativa peso 1. Mostre a média como resultado

9. Determine a negação da proposição: “Pedro viaja e Ana estuda”.
   - a) Pedro viaja ou Ana estuda.
   - b) Pedro não viaja e Ana estuda.
   - c) Pedro não viaja ou Ana estuda.
   - d) Pedro não viaja ou Ana não estuda.
   - e) Ana não estuda e Pedro viaja.

10. Determine o número de linhas na tabela-verdade para a proposição composta: “Alice vai ao cinema e Bob não vai ao cinema ou Carlos estuda e Daniela não estuda.”
   - a) 4
   - b) 8
   - c) 16
   - d) 32
   - e) 64

11. Representa as seguintes frases em linguagem natural, utilizando lógica formal. Para cada resposta, devem-se especificar as proposições simples
extraídas:

- a) João mede 1,78m e Maria pesa 60kg.
- b) Fortaleza é capital do Maranhão desde que Rio de Janeiro

- tenha mais de 250 mil habitantes.
- c) Ontem o dólar não fechou a R$2,18 ou o índice Bovespa
- fechou estável.
- d) Só irei ao clube se amanhã fizer sol.

12. Explique, com suas palavras, o que é lógica de programação.
13. Identifique os passos básicos para resolver um problema computacional.
14. Resolva um problema simples usando o raciocínio lógico: "Quantos números de 1 a 100 são múltiplos de 3?"
15. Crie um algoritmo para verificar se um número é par ou ímpar.
16. Elabore um algoritmo que recebe a idade de uma pessoa e informa se ela é maior ou menor de idade.
17. Desenhe um fluxograma para calcular a área de um retângulo dado o comprimento e a largura.
18. Crie um fluxograma que indica se um número é positivo, negativo ou zero.
19. Escreva o pseudocódigo para um programa que verifica se um número é múltiplo de 5.
20. Elabore o pseudocódigo de um sistema que calcula a soma de todos os números de 1 a 100.
21. Explique a importância de usar comentários em um código.
22. Identifique se as frases abaixo são proposições:
- O céu é azul.
- Abra a porta.
- Todos os números pares são divisíveis por 2.

23. Escreva o significado dos conectivos: "E", "OU" e "NÃO".
24. Desenvolva uma solução lógica para o jogo "Torre de Hanoi" com 3 discos.
25. Escreva um algoritmo para calcular o MDC (Máximo Divisor Comum) de dois números.
26. Desenvolva um algoritmo para determinar se uma palavra é um palíndromo, usando fluxograma.

27. Crie um fluxograma para implementar o algoritmo de Euclides para encontrar o MDC de dois números.
28. Desenhe um fluxograma que resolva uma equação de segundo grau.
29. Adicione comentários detalhados em um código.
30. Explique como a falta de indentação pode gerar problemas ao debugar um programa.
31. Identifique erros em um código que utiliza mal as palavras-chave.
32. Construa a tabela verdade para $(\neg P \vee Q) \wedge (P \wedge \neg Q)$.

33. Escreva a expressão lógica equivalente à seguinte proposição: "Se não chover e houver sol, então vou à praia".
34. Reescreva a seguinte proposição aplicando a negação: "Todos os alunos estudaram para a prova".
35. Verifique se a proposição "João está estudando e Maria está trabalhando" é verdadeira, dado que João está estudando e Maria não está trabalhando.

# MÓDULO 4: VARIÁVEIS

**A simplicidade não precede a complexidade, mas a segue.**

Alan Perlis

- VARIÁVEIS
- TIPOS DE DADOS
- TYPE CASTING
- NOMENCLATURA
- RESUMO
- EXERCÍCIOS

# Objetivos

- Introduzir o conceito de variáveis e como usá-las.
- Diferenciar tipos de dados primitivos e não primitivos.
- Explicar conversões de tipo (type casting) e convenções de codificação.
- Apresentar constantes e regras para nomenclatura de variáveis.

# VARIÁVEIS

Este módulo visa fornecer uma compreensão completa sobre variáveis, seu uso, tipos e convenções de nomenclatura. Ao final, você será capaz de declarar, inicializar e manipular variáveis de forma eficiente, além de aplicar as boas práticas de nomenclatura.

# O Conceito de Variáveis

Em Java, variáveis são os containers de dados que salvam os valores de dados durante a execução do programa Java. Cada Variável em Java recebe um tipo de dado que designa o tipo e a quantidade de valor que ela pode conter. Uma variável é um nome de local de memória para os dados.

Variável Java é um nome dado a um local de memória. É a unidade básica de armazenamento em um programa, o valor armazenado em uma variável pode ser alterado durante a execução do programa.

Variáveis em Java são apenas um nome dado a um local de memória. Todas as operações feitas na variável afetam esse local de memória.

Em Java, todas as variáveis devem ser declaradas antes do uso.

Variáveis são as unidades básicas de armazenamento em Java. Para uma compreensão mais profunda de tipos de variáveis, escopo e gerenciamento de memória.

## Como utilizamos as variáveis?

As variáveis são utilizadas para armazenar dados temporários que serão manipulados ao longo do programa. Elas podem armazenar diferentes tipos de informações, como números, caracteres, texto e valores booleanos.

# Declaração e Inicialização de Variáveis

Para usar uma variável em um programa, você primeiro precisa declará-la. Isso envolve informar ao compilador o tipo de dado que a variável irá armazenar e dar um nome a ela.

A declaração de uma variável em Java segue a sintaxe básica:

```
int feriasDias;  // trata-se de uma declaração

feriasDias = 12; // esta é uma tarefa
```

Eis um exemplo de uma atribuição a uma variável de caráter:

```
char simChar;

simChar = 'Y';
```

Uma caraterística interessante de Java é a capacidade de declarar e inicializar uma variável na mesma linha.

Por exemplo:

```
int feriasDias = 12; // trata-se de uma inicialização
```

Finalmente, em Java, pode colocar declarações em qualquer parte do seu código. Por exemplo, o seguinte é um código válido em Java:

```
double salario = 65000.0;
 System.out.println(salario);
 int feriasDias = 12;
// é possível declarar uma variável aqui
```

# Constantes

Em Java, a palavra-chave **final** é utilizada para designar uma constante. Por exemplo,

```
final double CM_PER_INCH = 2.54;
double paperWidth = 8.5;
double paperHeight = 11;
System.out.println("Paper size in centimeter: "
            + paperWidth * CM_PER_INCH + " by "
            + paperHeight * CM_PER_INCH);
```

A palavra-chave **final** indica que pode atribuir à variável uma vez e que o seu valor é definido de uma vez por todas. É habitual nomear as constantes com todas as letras maiusculas.

É provavelmente mais comum em Java querer uma constante que esteja disponível para múltiplos métodos dentro de uma única classe. Estas são normalmente designadas por constantes de classe. Uma constante de classe é definida com as palavras-chave **static final**.

Eis um exemplo de utilização de uma constante de classe:

```
public class Main {

   public static final double CM_PER_INCH = 2.54;;

   public static void main(String[] args) {
     double paperWidth = 8.5;
     double paperHeight = 11;
     System.out.println("Paper size in centimeter: "
                + paperWidth * CM_PER_INCH + " by "
                + paperHeight * CM_PER_INCH);
    }
}
```

Note-se que a definição da constante de classe aparece fora do método principal. Assim, a constante também pode ser usada noutros métodos da mesma classe. Além disso, se (como no nosso

exemplo) a constante for declarada pública, os métodos de outras classes também podem usar a constante.

Exemplo, como Constants2.CM_PER_INCH.

# Tipos de Variáveis

Existem três tipos principais de variáveis em Java:

1. Variáveis de Instância

- **Definição**: São variáveis que pertencem a uma instância (ou objeto) de uma classe. Cada objeto possui a sua própria cópia dessas variáveis.
- **Declaração**: São declaradas dentro de uma classe, mas fora de qualquer método, construtor ou bloco.
- **Escopo**: O escopo de uma variável de instância é limitado ao objeto da classe. Elas podem ser acessadas por qualquer método da classe.
- **Valor padrão**: Se não forem inicializadas explicitamente, assumem um valor padrão, que depende do tipo (ex.: `0` para tipos numéricos, `false` para booleanos, `null` para objetos).

```
int idade; // Variável de instância
```

2. Variáveis Locais

- **Definição**: São variáveis declaradas dentro de métodos, construtores ou blocos de código. Elas existem apenas enquanto o bloco em que foram declaradas está sendo executado.
- **Escopo**: Limitado ao método ou bloco onde a variável foi declarada.
- **Valor padrão**: Não possuem valor padrão, então devem ser inicializadas antes de serem usadas, ou o compilador gerará um erro.

```
public void metod() {
    int x = 10;// Variável local
}
```

3. Variáveis Estáticas (ou Variáveis de Classe)

- **Definição**: São variáveis que pertencem à classe, e não a instâncias (objetos) individuais da classe. Isso significa que todas as instâncias da classe compartilham a mesma cópia dessa variável.
- **Declaração**: São declaradas usando a palavra-chave `static` dentro de uma classe, mas fora de qualquer método ou bloco.
- **Escopo**: O escopo é global à classe. Elas podem ser acessadas por qualquer método, e também podem ser acessadas sem criar uma instância da classe.
- **Valor padrão**: Como as variáveis de instância, variáveis estáticas também têm valores padrão.

```
public class Exemplo {
    static int contador;// Variável estática
}
```

# Tipos Primitivos de Java

Em programação, variáveis podem armazenar diferentes tipos de informações, e o tipo de dado determina como esses dados são armazenados e manipulados. Aqui estão alguns exemplos de tipos de dados comuns:

| Palavras Reservadas | Tamanho | Descrição | Valores Padrões |
|---|---|---|---|
| Boolean | 1 bit | Armazena valores true ou false | false |
| Byte | 1 byte – 8 bits | Armazena números inteiros –128 a 127 | 0 |
| Short | 2 bytes – 16 bits | Armazena números inteiros de –32.768 a 32.767 | 0 |
| Int | 4 bytes – 32 bits | Armazena números inteiros de –2.147.483.648 a 2.147.483.647 | 0 |
| Long | 8 bytes – 64 bits | Armazena números inteiros de –9.223.372.036.854.775.808 a 9.223.372.036.854.775.807 | 0L |
| Float | 4 bytes – 32 bits | Armazena números fracionários de 6 a 7 dígitos decimais | 0.0f |
| Double | 8 bytes – 64 bits | Armazena números fracionários de 15 a 16 dígitos decimais | 0.0d |
| Char | 2 bytes – 16 bits | Armazena um único caractere/letra ou valores ASCII de 0 a 255 | ‘\u0000’ |

**Numéricos**: Armazenam números. Podem ser inteiros (como int ou long) ou números com ponto flutuante (como float ou double).

**Strings**: Armazenam texto. Em Java, o tipo de dado para texto é String.

**Booleanos**: Armazenam valores lógicos, verdadeiro ou falso, representados pelo tipo boolean.

**Caracteres**: Armazenam um único caractere. Em Java, usamos o tipo char para isso.

**Objetos**: Em linguagens orientadas a objetos como Java, variáveis podem armazenar referências a objetos, mas não é objetivo deste livro tratar sobre Programação Orientada a Objeto.

## Números inteiros

Há quatro tipos de inteiros em Java:

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

Um número inteiro pode sempre ser atribuído a outro de maior precisão:

```
int a = 274;
long b = a;
```

A operação inversa requer coerção explícita ("casting"):

```
long a = 274;
int b = ( int ) a;
```

Sem esta coerção, haverá erro de compilação. Note que, mesmo com a coerção, ainda pode haver problema na execução, caso o valor do número atribuído ultrapassar o maior valor possível para o tipo em questão.

# Números em ponto flutuante

Há dois tipos, de precisão simples e de precisão dupla:

| Tipo | Tamanho | Valor |
|---|---|---|
| float | 32 bits | -3.40292347E+38 a +3.40292347E+38 |
| double | 64 bits | -1.79769313486231570E+308 a +1.79769313486231570E+308 |

Um número de precisão simples pode sempre ser atribuído a outro de precisão dupla:

```
 float a = 2.74F;
// F (ou f) após o literal indica precisão simples

double b = a;
```

A operação inversa requer coerção explícita:

```
double a = 2.74e12;

float b = ( float ) a;
```

Sem esta coerção, haverá erro de compilação. De novo, mesmo com a coerção, ainda pode haver problema na execução, caso o valor do número de precisão dupla ultrapassar o maior valor possível para um número de precisão simples.

Note que Java não possui tipos primitivos representando números complexos. Não há recursos para lidar com números complexos nas bibliotecas fornecidas com o JDK, mas vários pacotes avulsos são disponíveis. Não sendo tipos primitivos, números complexos devem ser objetos em Java.

# Carateres

Há um tipo primitivo que representa um caráter:

| Tipo | Tamanho | Valor |
|---|---|---|
| char | 16 bits | '\u0000' a '\uFFFF' |

Java utiliza o padrão de caracteres Unicode, que abrange os conjuntos de caracteres de muitas línguas.

Um literal de caráter é especificado por um único caráter entre apóstrofes simples:

```
char exemplo = 'a';
```

Para caracteres de escape, usa-se a barra invertida:

| Escape | Significando |
|---|---|
| \n | Nova linha |
| \t | Tabulação |
| \b | Backspace |
| \r | Retorno |
| \\ | Barra invertida |
| \’ | Apóstrofe |
| \” | Aspas |

## Booleanos

São variáveis lógicas que podem assumir os valores verdadeiro e falso:

| Tipo | Tamanho | Valor |
|---|---|---|
| Boolean | 1 bit | True ou False |

Note que, em Java, estas variáveis não podem ser interpretadas como os números inteiros 0 e 1.

## Tipos de Dados Primitivos

Java possui oito tipos primitivos: byte, short, int, long, float, double, boolean, e char. Eles armazenam valores simples, como números inteiros e reais, caracteres e valores booleanos.

## Tipos de Dados Não Primitivos

São aqueles que não fazem parte da linguagem diretamente e são definidos por classes, como String, arrays e objetos. Esses tipos podem ser complexos e podem armazenar múltiplos valores ou comportamentos.

## Principais diferenças - tipos primitivos x não primitivos

1 Tipos primitivos são predefinidos em *Java*, e não primitivos são criados pelo programador e não são definidos por *Java* (exceto para *String*).

2 Tipos não primitivos podem ser usados para chamar métodos para realizar certas operações, enquanto tipos primitivos não podem.

3 Um tipo primitivo sempre tem um valor, enquanto tipos não primitivos podem ser *null*.

4 Um tipo primitivo começa com uma letra minúscula, enquanto os tipos não primitivos começam com uma letra maiuscula.

5 O tamanho de um tipo primitivo depende do tipo de dados, enquanto os tipos não primitivos têm todos o mesmo tamanho.

# Type Casting

Type casting, também conhecido como conversão de tipo, é um conceito importante em Java que nos permite converter um tipo de dado em outro.

É útil quando precisamos executar operações em diferentes tipos de dados ou quando você quer armazenar um valor de um tipo de dado em uma variável de outro tipo de dado.

Type casting em Java envolve a conversão de variáveis entre diferentes tipos de dados, essencial para programação versátil.

O domínio do type casting garante o manuseio eficiente de dados, sejam tipos primitivos ou objetos.

Incluímos algumas conversões de tipos comuns em Java com exemplos de vários tipos de conversão de tipos em Java.

Isso permitirá que você entenda a conversão ou conversão de tipos de uma maneira melhor.

## O que é Type Casting?

Typecasting é o processo de converter o valor de um único tipo de dado (como um inteiro [int], float ou double) em outro tipo de dado.

Essa conversão é feita de forma automática ou manual. O compilador realiza a conversão automática, e um programador faz a conversão manual. Para usar uma variável de uma maneira particular na conversão automática, precisamos dizer explicitamente ao compilador Java para converter uma variável de um tipo de dado para outro tipo de dado.

```
<datatype> variableName = (<datatype>) value;
```

# Importância do Type Casting no Java

Conversão de tipo e conversão de tipo em Java têm grande importância. Abaixo estão os motivos mais significativos para usar conversão de tipo.

- Prevenção de perda de dados: quando mudamos dados de um tipo para outro, precisamos garantir que não perderemos nenhuma informação valiosa. A conversão de tipo nos ajuda a converter dados com segurança, mantendo sua integridade intacta.

- Fazendo operações funcionarem: Imagine tentar misturar um copo de água com uma xícara de açúcar. Simplesmente não daria certo! Da mesma forma, algumas operações não podem funcionar com diferentes tipos de dados. A conversão de tipos torna essas operações possíveis ao tornar os tipos de dados compatíveis.

- Uso Eficiente de Memória: A conversão de tipos e a conversão de tipos em Java ajudam a gerenciar melhor a memória. Às vezes, um tipo de dado maior pode ser usado para um valor menor. Isso pode desperdiçar a memória. A conversão de tipos nos permite usar o tamanho certo para os dados certos.

- Manipulando entrada do usuário: Quando recebemos entrada dos usuários, geralmente é em forma de texto. Mas, precisamos de números para cálculos. A conversão de tipo ajuda a converter esse texto em números para processamento adequado.

- Melhor controle de programação: Java é uma linguagem fortemente tipada, o que significa que é rigorosa sobre tipos de dados. A conversão de tipos e a conversão de tipos em Java permitem que os programadores assumam o controle e garantam que o tipo de dados certo seja usado no momento certo.

# Tipos de conversão de tipo em Java

Existem 13 tipos diferentes de conversão de tipos em Java.

Neste tutorial, vamos dar uma olhada apenas nos dois tipos principais.

## 1. Widening Type Casting

A conversão de tipo de alargamento é o processo de converter um tipo de dado inferior para um tipo de dado superior. Também é chamado de conversão implícita ou conversão para baixo. Esse processo é realizado automaticamente e é seguro, pois não há risco de perda de dados.

Esse tipo de conversão de tipo e conversão em Java acontece quando:

- O tipo de destino é maior que o tipo de origem.
- Os dois tipos de dados são compatíveis.

byte -> short -> char -> int -> long -> float -> double (Da esquerda para a direita: Tipo de dado inferior para Tipo de dado superior)

Sintaxe:

larger_data_type variable_name = smaller_data_type_variable;

## 2. Narrowing Type Casting

A conversão de tipo estreito é o processo de redução de um tipo de dado maior para um menor. Outros nomes para isso são conversão para cima ou conversão de tipo explícito em Java. Isso não acontece por si só. Se não fizermos isso explicitamente, obteremos um erro de tempo de compilação. A conversão de tipo estreito não é segura porque pode ocorrer perda de dados devido ao menor intervalo de valores permitidos do tipo de dado inferior. Um operador de conversão ajuda na conversão explícita.

double -> float -> long -> int -> char -> short -> byte (Da esquerda para a direita: Tipo de dado superior para Tipo de dado inferior)

Sintaxe:

larger_data_type variable_name = smaller_data_type_variable;

# Exemplos de conversão de tipo em Java

Aqui estão alguns exemplos de conversão de tipos em Java.

## 1. Convertendo int para double

```
public class Main {
    public static void main(String[] args) {
        // cria variável do tipo int
        int num = 50;
      System.out.println("O valor inteiro: "+ num);

        // converter em tipo double
        double dados = num;
      System.out.println("O valor double: "+ dados);
    }
}
```

Saída:

int valor: 50

double valor: 50.0

## 2. Convertendo Double em Int

```
public class Main {
    public static void main(String[] args) {
        // cria uma variável do tipo double
        double num = 50.55;
       System.out.println("o valor double: "+ num);

        // converter para o tipo int
        int data = (int)num;
     System.out.println("o valor inteiro: "+ data);
    }
}
```

Saída:

double valor: 50.55

int valor: 50

## 3. Convertendo Int em String

```
// cria variável do tipo int

int num = 50;
System.out.println("O valor inteiro é: "+ num);

// saída -> int valor: 50

// converter int para tipo string

String data = String.valueOf(num);
System.out.println("O valor da string é: "+ data);
// saída -> String valor: 50
```

## 4. Convertendo String to int

```
// criação de variável do tipo String

String data = "50";
System.out.println("A variável String é: "+ data);


// convertendo a variável string para int

int num = Integer.parseInt(data);

System.out.println("A variável int é: "+ num);
```

Saída:

String valor: 50

int valor: 50

# Tabela Resumo das Conversões

Esta tabela resume as conversões todas possíveis no java:

| Conversão | Tipo | Exemplo |
|---|---|---|
| byte para short | Implícita | byte b = 10; short s = b; |
| short para int | Implícita | short s = 100; int i = s; |
| char para int | Implícita | char c = 'A'; int i = c; |
| int para long | Implícita | int i = 1000; long l = i; |
| float para double | Implícita | float f = 10.5f;<br>double d = f; |
| int para byte | Explícita | int i = 130; byte b = (byte) i; |
| double para int | Explícita | double d = 10.99;<br>int i = (int) d; |
| float para int | Explícita | float f = 9.99f; int i = (int) f; |
| double para float | Explícita | double d = 100.25;<br>float f = (float) d; |

Essas conversões são fundamentais para garantir que o tipo de dados correto seja usado, mantendo a eficiência e precisão necessárias para aplicações em Java.

# Convenções de Codificação

As convenções de codificação definem um conjunto de regras que devem ser seguidas durante a escrita de programas Java. Os objetivos principais das convenções de codificação são:

- Aumentar a legibilidade do código produzido por cada programador
- Apresentar um aspecto uniforme relativamente ao código produzido por vários programadores.

## Convenção de Nomes

Os nomes de atributos, classes e métodos devem transmitir a funcionalidade da entidade em causa. Isto permite aumentar a legibilidade do código. A regra de codificação relativa à especificação do nome a utilizar para uma dada entidade depende do seu tipo.

Isto permite distinguir o tipo de uma entidade com base no seu nome. Deve-se utilizar bons nomes para as variáveis que devem refletir a razão de utilização da variável, utilizando as regras descritas a seguir. Apesar desta diretiva ser subjetiva, evite ser muito verboso (nome da variável muito grande) ou muito sucinto.

## Nomes de classes e interfaces

- Começar por uma maiuscula. As letras seguintes são minúsculas, exceto no início de novas palavras
- Não usar traços de união
- Evitar o uso de abreviaturas e acrónimos
- Bons exemplos: Gato, PeleGato
- Maus exemplos: gato, PULGA, Gato_pata

## Nomes de métodos

- Letras minúsculas exceto a primeira letra de novas palavras
- Não usar traços de união
- Usar verbos
- Bons exemplos: correr (), correrDepressa(), chamarJerry()
- Maus exemplos: Felix(), Aceder (), obter_Peso()

## Nomes de variáveis

- Letras minúsculas exceto a primeira letra de novas palavras
- Não usar traços de união
- Curtos mas com significado
- Evitar nomes com uma letra, exceto se a variável tem um âmbito reduzido e é para "deitar fora". Isto é o caso dos índices utilizados no contexto de um ciclo com poucas instruções:
    - i, j, k, m, n para inteiros
    - c, d, e para caracteres
- Bons exemplos: acariciado, gato, meuComprimento
- Maus exemplos: Felix, meu Comprimento

## Nomes de variáveis de instância e de classe

- Devem começar com um traço de união (underscores) seguido de letras minúsculas
- Bons exemplos: _pessoa, _contador, _totalCount

## Nomes de constantes

- Letras maiusculas e palavras separadas por traços de união
- Bons exemplos: NUM_OSSOS, MAX_PONTOS

## Produzir Código Limpo

- Apagar o código que está obsoleto e não é usado. Não se limite a comentar o código obsoleto.
- Uma instrução por linha. Não colocar várias instruções na mesma linha.
- Não utilizar métodos “deprecated” da biblioteca do Java
- Comentar o código, mas evitar ser muito verboso ou sintético

## Variáveis Locais

Declarar uma variável local por linha do código.

As variáveis locais devem ser declaradas no início de cada bloco. Esta regra permite a quem lê o código saber facilmente quais são as variáveis locais que são necessárias para um dado bloco de código.

### Espaços em Branco

A utilização de um espaço em branco permite aumentar a legibilidade do código nalgumas situações:

- Entre uma palavra-chave e um parêntese. Exemplo:

```
while (condicao) {
}
```

- Após cada vírgula na especificação dos parâmetros de um método. Exemplo:

```
método(tipo1 param1, tipo2 param2, tipo3 param3)
```

- Entre um operador binário e seus operandos. Exemplo:

```
x += y + z;
a = (a + b) / (c * d);
```

# Indentação de Código e Blocos de instruções

O código deve estar sempre indentado. Use um espaçamento de 2 ou 4 espaços. (De preferência, utilize um editor de texto que realize a indentação de código de forma automática).

A indentação do código aumenta a sua legibilidade.

Siga um esquema coerente para colocar o início e fim de blocos de instruções (representado por '{' e '}', respectivamente). Existem duas abordagens possíveis. Uma consiste em alinhar verticalmente os dois caracteres que representam o início e fim do bloco de instruções.

Exemplo:

```
class Exemplo {
  public void metodo() {
      if(condição) {
          // algum código
      }
  }
}
```

A segunda alternativa corresponde a colocar o caractere que representa o início de um bloco de instruções na extremidade da linha que começa o bloco. O caráter de fim do bloco de instruções fica alinhado com a instrução que começa no bloco. Esta abordagem alternativa torna o código um pouco mais compacto sem que isso diminua a sua legibilidade. Exemplo:

```
class Exemplo{
    public void metodo(){
    if(condição){
        // algum código
        }
    }
}
```

# Regras Para a Nomenclatura

As regras para nomear variáveis em Java seguem convenções de linguagem e boas práticas de programação.

As principais regras são:

1. Iniciar com letra, cifrão ( $ ) ou sublinhado ( _ ).

2. O primeiro caractere de um nome de variável deve ser uma letra (a–z ou A–Z).

3. Nomes de variáveis não podem começar com um número.

4. Não usar palavras reservadas.

5. Case-sensitive, o Java diferencia maiusculas de minúsculas, então idade, Idade e IDADE são três variáveis distintas.

6. Caracteres permitidos: Além do primeiro caractere, o nome da variável pode conter letras, números, cifrões ( $ ) e sublinhados ( _ ).

Espaços e outros caracteres especiais (como @, #, !, etc.) não são permitidos.

7. Tamanho sem limite explícito: Não há limite explícito de tamanho para o nome da variável, mas nomes excessivamente longos não são recomendados.

8. Convenções de nomenclatura (não obrigatórias, mas recomendadas):

Usa-se o padrão camelCase para variáveis e métodos.

O primeiro caractere é minúsculo, e cada palavra subsequente começa com letra maiuscula, como em minhaVariavel ou quantidadeMaxima.

Escolha nomes descritivos que indiquem claramente o propósito da variável.

# Lista De Exercícios

1. Declare e inicialize três variáveis de tipos diferentes.

2. Crie uma variável do tipo `float` e converta seu valor para `int`.

3. Nomeie uma variável seguindo a convenção camelCase e outra como uma constante.

4. Identifique o erro no código abaixo:

```
int 2idade = 30;
```

5. Referente ao código abaixo:

```
int x = 7;
System.out.printf("%d",++x%2);
```

Qual será o resultado:

a) Imprime 6.

b) Imprime 7.

c) Imprime 8.

d) Imprime 1.

e) Imprime 0.

6. Desenvolva um algoritmo em Java que leia um número inteiro e imprima o seu antecessor e seu sucessor.

7. Fazer um programa que imprima a média aritmética dos números 8,9 e 7. A média dos números 4, 5 e 6. A soma das duas médias. A média das médias.

8. Criar uma variável com o nome saldo e atribuir um valor, e imprimir o saldo com reajuste de 1%.

9. Crie uma variável float e imprima o resultado do quadrado desse número.

10. Assinale a alternativa correta sobre os tipos básicos de dados na Linguagem java:

I - char: Caractere - O valor armazenado é um caractere. Caracteres geralmente são armazenados em códigos (usualmente o código ASCII).

II - int: Número inteiro - É o tipo padrão e tamanho de representação vai depender da máquina em que o programa foi compilado variando normalmente de 2 a 4 bytes.

III - float: Número ponto flutuante de precisão simples - São conhecidos normalmente como números reais.

a) Somente I.

b) Somente I e II.

c) Somente III.

d) Somente I, II e III.

11. Leia um número real e imprima a quinta parte deste número.

12. Entrar com os valores dos catetos (8, 9) de um triângulo retângulo e imprimir a hipotenusa.

13. Ler dois números reais e imprimir o quadrado da diferença do primeiro valor pelo segundo e a diferença dos quadrados, onde lado A = 23.12, lado B = 10.23

14. Para vários tributos, a base de cálculo é o salário mínimo. Fazer um algoritmo que leia o valor do salário mínimo e o valor do salário de uma pessoa. Calcular e imprimir quantos salários mínimos ela ganha.

15. Qual será a saída no terminado referente a execução do código abaixo?

```
char a = 97;
float x = 2.3f;
int resultado = (int) (x + a);

System.out.printf("%.2f\n", resultado);
```

a) 99.3

b) 97.3

c) 99

d) 99.30

e) Nenhuma das alternativas

16. Qual será a saída no terminado referente a execução do código abaixo?

```
char a = 97;
float x = 2.3f;
int resultado = (int) (x + a);

System.out.printf("%.2f\n", resultado);
```

a) 99.3

b) 97.3

c) 99

d) 99.30

e) Nenhuma das alternativas

17. Leia uma temperatura em graus Celsius e apresente-a convertida em graus Fahrenheit. A fórmula de conversão é: F = C * (9.0/5.0) + 32.0, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

18. Faça um algoritmo que leia dois valores para as variáveis A e B e efetue a troca dos valores de forma que a variável A passe a possuir o valor da variável B e a variável B passe a possuir o valor da variável A. Apresenta os valores trocados.

19. Escreva um programa para ler o raio de um círculo, calcular e escrever a sua área. $A = \pi r^2$

20. O cardápio de uma lanchonete é dado abaixo. Prepare um algoritmo que leia a quantidade de cada item que você consumiu e calcule o valor da conta final.

- X-salada.....R$ 5.50
- X-Bacon.....R$ 7.25
- X-Tudo.....R$ 10.30
- X-Egg.....R$ 7.00
- Cerveja.....R$ 4.50
- Refrigerante.....R$ 6.00

21. Escreva um programa para ler as dimensões de uma cozinha retangular (comprimento, largura e altura), calcular e escrever a quantidade de caixas de azulejos para se colocar em todas as suas paredes (considere que não será descontada a área ocupada por portas e janelas). Cada caixa de azulejos possui $3m^2$.

22. Uma equipe de Fórmula 1 deseja calcular o número mínimo de litros que deverá colocar no tanque de seu carro de corrida para que ele possa percorrer um determinado número de voltas até o primeiro reabastecimento.

Faça um programa que leia o comprimento da pista (em metros), o número total de voltas a serem percorridas em toda a corrida, o número de abastecimentos desejados e o consumo de combustível do carro (em Km/L).

Calcular e escrever o número mínimo de litros necessários para percorrer até o primeiro reabastecimento. OBS: Considere que o número de voltas entre os abastecimentos é o mesmo.

23. Escreva um programa que leia um número inteiro e calcule e mostre a sua decomposição em unidade, dezena, centena e milhar. Considere que o número máximo recebido via teclado será de 9999. Exemplo: A entrada 8531 terá a saída: unidade = 1 dezena = 3 centenas = 5 milhar = 8.

24. Suponha que iremos começar a desenvolver o programa de gerenciamento de mercadorias de uma loja. Escreva um código que declare variáveis para representar os seguintes dados: número do pedido, código do produto, quantidade e valor total da compra.

25. Continuando o exercício anterior, inicialize as variáveis com valores de acordo com o tipo de variável que você escolheu em cada declaração.

26. Continuando o exercício anterior, imprima na tela o valor de cada variável.

27. Sobre a afirmação a seguir, assinale a alternativa correta: Uma convenção na linguagem java é que todos os nomes de variáveis devem possuir apenas letras minúsculas ou ao menos iniciar com uma letra minúscula, pois letras maiusculas são reservadas para nomes de “constantes” na linguagem java. Por se tratar de convenção, o código irá funcionar normalmente se você utilizar letras maiusculas, contudo, não é recomendado por dificultar a leitura do código por outros programadores.

a) A afirmação é verdadeira.

b) A afirmação é falsa.

28. Faça um programa em que leia a base e a altura de um retângulo e imprima o perímetro (base + altura) e a área (base * altura).

29. Faça um programa que leia o valor de um produto, o percentual do desconto desejado e imprima o valor do desconto e o valor do produto subtraindo o desconto.

30. Faça um programa que calcule o reajuste do salário de um funcionário. Para isso, o programa deverá ler o salário atual do funcionário e ler o percentual de reajuste. Ao final, imprimir o valor do novo salário.

31. Leia um ângulo em graus e apresente-o convertido em radianos. A fórmula de conversão é: R = G * π /180, sendo G o ângulo em graus e R em radianos e crie uma constante π (PI) = 3.14.

32. Leia um ângulo em radianos e apresente-o convertido em graus. A fórmula de conversão é: G = R * 180/π , sendo G o ângulo em graus e R em radianos e crie uma constante PI = 3.14.

33. Leia um valor de comprimento em polegadas e apresenta-o convertido em centímetros. A fórmula de conversão é: C = P * 2,54, sendo C o comprimento em centímetros e P o comprimento em polegadas.

34. Dados o código abaixo que compila e executa sem gerar erros, assinale a alternativa correta:

```
int a = 7, b = 4, soma;

soma = a + b;

System.out.printf("soma: %d", soma);
```

a) O resultado em tela deverá ser o número 11.

b) O resultado em tela deverá ser o número 10.

c) O resultado em tela deverá ser o número 9.

d) O resultado em tela deverá ser o número 2.

e) O resultado em tela deverá ser o número 28.

35. Faça um Programa para uma loja de tintas. O programa deverá pedir o tamanho em metros quadrados da área a ser pintada.

Considere que a cobertura da tinta é de 1 litro para cada 6 metros quadrados e que a tinta é vendida em latas de 18 litros, que custam $140,00 ou em galões de 3,6 litros, que custam $52,00.

Informe ao usuário as quantidades de tinta a serem compradas e os respectivos preços em 3 situações. Sempre arredonde os valores para cima, isto é, considere latas cheias:

- Comprar apenas latas de 18 litros;
- Comprar apenas galões de 3,6 litros;
- Misturar latas e galões.

36) Marque a opção que em que todos os nomes de variáveis são válidos. Os nomes estão separados por vírgulas.

a) real$, errado!, correta

b) casa, olho gordo, valor

c) terminou, int, k

d) case, x, casa

e) Nenhuma das alternativas

# MÓDULO 5: ENTRADA E SAÍDA DE DADOS

A televisão é a maior maravilha da ciência a serviço da imbecilidade humana.

Barão de Itararé

- ENTRADA & SAÍDA DE DADOS
- CLASSE DE SCANNER
- ENTRADA E SAÍDA DE DADOS
- COMO RECEBER INFORMAÇÕES DE USUÁRIOS
- MÉTODO PRINT() E PRINTLN()
- EXERCÍCIOS

# Objetivos

Neste módulo, exploramos como trabalhar com entrada e saída (I/O) em Java. É uma habilidade essencial para criar programas que interagem com usuários. Vamos abordar conceitos teóricos, exemplos práticos e exercícios.

- Entender como receber dados do usuário.
- Explorar diferentes classes para a entrada de dados.
- Aprender a exibir saídas formatadas no console.
- Diferenciar métodos e práticas de I/O.

# Entrada e Saída de Dados

Um problema pode ser definido como uma pergunta (ou situação) de caráter geral a ser respondida (ou resolvida). Já a lógica se refere à técnica ou à forma de ordenar as ações que representam a resolução de um problema.

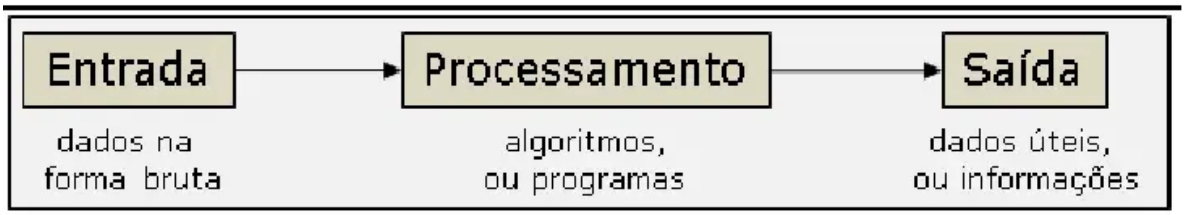
Para resolver um problema no computador é necessário analisá-lo visando descrever uma sequência lógica de passos executáveis que permitam que o problema possa ser resolvido de maneira automática e repetitiva. Esta sequência de passos é chamada de algoritmo, quando os passos são descritos em linguagem natural, e programa, quando eles são implementados utilizando símbolos ou convenções de uma linguagem de programação.

Basicamente, todo programa de computador usado na solução de um problema executa algum tipo processamento de dados, que geralmente é desenvolvido em três etapas:

- A. entrada de dados pelo usuário através do teclado;
- B. processamento realizado através de cálculos;
- C. saídas de informações na tela do computador.

Em resumo, o processamento de dados ocorre quando um conjunto de dados de entrada é transformado através do processamento de algoritmos (ou programas) em um conjunto de informações de saída.

---




Exemplo: Etapas do processamento de dados

A entrada refere-se aos dados que são colhidos do mundo real e o processamento corresponde a uma série finita de operações que são realizadas sobre estes dados para transformá-los em informações úteis na saída.

# Entrada de Dados

Um processo é uma sequência finita ordenada de passos que transforma uma determinada matéria prima. Quando a matéria prima usada no processo é abstrata, isto é, se apresenta na forma de valores e quantidades, então denomina-se de processamento de dados. Quando o processamento é realizado por um computador, a entrada refere-se aos dados obtidos na forma bruta, que são colhidos do mundo real através de um dispositivo de entrada, como por exemplo, o teclado.

# Classe Scanner

Em linguagens de programação estruturada, como C e Pascal, a entrada de dados usando o teclado é realizada utilizando subprogramas desenvolvidos para esta finalidade. Em C, por exemplo, tem-se a função **scanf()** e em Pascal, o procedimento **readln()**.

Na linguagem Java, a partir do Java 1.5 ou Java 5.0, o pacote de classes **Java.util** disponibilizou a classe Scanner, que implementa operações de entrada de dados pelo teclado.

A classe Scanner possui vários métodos que possibilitam a entrada de dados de diferentes tipos, entre eles destacam-se:

**String next()** - retorna uma cadeia de caracteres simples, ou seja, que não usa o caractere espaço em branco;

**double nextDouble()** - retorna um número em notação de ponto flutuante normalizada em precisão dupla de 64 bits (usado para receber valores reais ou monetários);

**boolean hasNextDouble()** - retorna true se o próximo dado de entrada pode ser interpretado como um valor double;

**int nextInt()** - retorna um número inteiro de 32 bits;

**boolean hasNextInt()** - retorna true se o próximo dado de entrada pode ser interpretado como um valor int;

**String nextLine()** - retorna uma cadeia de caracteres, por exemplo: “uma boa escolha programar”;

**long nextLong()** - retorna um número inteiro de 64 bits.

## Para que serve

Acima, expliquei de forma geral o que é a classe Scanner do Java. Agora, chegou a hora de avançar um pouco e entender o objetivo geral dela. Dessa forma, ela possui a finalidade de separar a entrada dos textos em blocos. Deste modo, permite a criação dos tokens. Eles são sequências de caracteres que são separados por delimitadores. O padrão destes delimitadores é definido em tabulações, espaços em branco e em mudanças de linhas.

## Como usar

Chegou, portanto, o momento de aprender como usar a classe Scanner do Java na prática. Para isso, precisamos começar a entender como se dá a criação de uma classe.

Para utilizar a classe Scanner em uma aplicação Java deve-se proceder da seguinte maneira:

- Importar o pacote java.util:

```
import java.util.Scanner;
```

- Instanciar e criar um objeto Scanner usando o dispositivo padrão de entrada (System.in):

```
Scanner ler = new Scanner (System.in);
```

Utilizar os métodos da classe Scanner adequados aos tipos das variáveis envolvidas.

Os exemplos de entradas de dados serão demonstrados usando as seguintes variáveis:

```
int n;
double preco;
String palavra;
String frase;
```

Lendo um valor inteiro:

```
System.out.printf("Informe um número para a tabuada:\n");
n = ler.nextInt();
```

Lendo um valor real:

```
System.out.printf("Informe o preço da mercadoria:\n");
preco = ler.nextDouble();
```

Lendo uma String:

```
System.out.printf("Informe uma palavra:\n");
palavra = ler.next();
```

Lendo uma String:

```
System.out.printf("Informe uma frase:\n");
frase = ler.nextLine();
```

Minha primeira aplicação

Os passos necessários para utilização da classe Scanner são demonstrados na classe Exemplo1, apresentada abaixo. No código desta classe podemos verificar como usar o método nextInt() na entrada

de dados numéricos do tipo inteiro (int), que em nosso exemplo são utilizados para a realização de operações matemáticas.

```
import java.util.Scanner; // 1. importando a classe Scanner


public class Exemplo1 {
  public static void main(String[] args) {
  Scanner ler = new Scanner(System.in);
// 2. instanciando e criando um objeto Scanner
  int a, b;


  System.out.printf("Informe o primeiro valor: ");
  a = ler.nextInt();
// 3. entrada de dados (lendo um valor inteiro)


  System.out.printf("Informe o segundo valor.: ");
  b = ler.nextInt();
// 3. entrada de dados (lendo um valor inteiro)


  System.out.printf("\nResultados:\n");
  System.out.printf("%d + %d = %3d\n", a, b, (a + b));
  System.out.printf("%d - %d = %3d\n", a, b, (a - b));
  System.out.printf("%d * %d = %3d\n", a, b, (a * b));
  System.out.printf("%d / %d = %3d (divisão inteira)\n", a, b,
(a / b));
  System.out.printf("%d / %d = %6.2f (divisão exata)\n", a, b,
((double)a / b));
  }
}
```

exemplo1: Realizando as operações matemáticas fundamentais

O código fonte, apresentado no Exemplo 1, implementa a entrada de dois valores inteiros (variáveis a e b, respectivamente) que serão usados nas operações matemáticas de adição, subtração, multiplicação, divisão inteira e divisão exata realizadas nos comandos de saída **System.out.printf()**. Pode-se observar também que através de comentários foram colocados em destaque às etapas:

1. importando a classe Scanner;

2. instanciando e criando um objeto Scanner denominado ler; e

3. entrada de dados (lendo um valor inteiro); necessárias para utilizar a classe Scanner na entrada de dados do tipo inteiro (int). A Figura 2 ilustra a execução da classe Exemplo1

```
run:
Informe o primeiro valor: 9
Informe o segundo valor.: 2

Resultados:
9 + 2 =  11
9 - 2 =   7
9 * 2 =  18
9 / 2 =   4 (divisão inteira)
9 / 2 =   4,50 (divisão exata)
```

Figura 2: Executando a classe Exemplo1

## Esvaziando o buffer do teclado

O buffer de entrada é uma entidade intermediária estabelecida entre duas outras entidades:

o dispositivo de entrada utilizado (teclado), e a aplicação Java. A primeira produzindo dados e a segunda consumindo através dos comandos de entrada.

Como estas entidades podem estar operando em diferentes velocidades, internamente, um buffer de entrada possui uma área de memória que é utilizada para o armazenamento temporário de dados que foram produzidos, mas ainda não foram consumidos.

Para utilização correta do buffer, quando existe a necessidade de realizar uma entrada consecutiva de dados numéricos e cadeias de caracteres, deve-se esvaziar o buffer depois da leitura do valor numérico e antes da leitura de um valor do tipo String, como implementado na classe Exemplo2, apresentada abaixo.

```
import java.util.Scanner; // 1. importando a classe Scanner


public class Exemplo2 {
  public static void main(String[] args) {
        Scanner ler = new Scanner(System.in)
        // 2. instanciando e criando um objeto Scanner
        int idade;
        String nome;


        System.out.printf("Informe a sua idade:\n");
        idade = ler.nextInt(); // 3. entrada de dados (lendo um valor inteiro)
        ler.nextLine(); // esvazia o buffer do teclado


        System.out.printf("\nInforme o seu nome:\n");
        nome = ler.nextLine(); // 3. entrada de dados (lendo uma String)


        System.out.printf("\nResultado:\n");
        System.out.printf("%s tem %d anos.\n", nome, idade);
  }
}
```

## Lendo um caractere

Diferentemente do que ocorre com os tipos base int e double, a classe Scanner não oferece um método específico para a leitura de dados do tipo caractere (char). Assim, para ler um caractere deve-se utilizar o método read() da classe System através do fluxo de entrada de dados padrão System.in.

## Saída de Dados

Como já informado, no processamento de dados, a entrada refere-se à etapa em que os dados são colhidos do mundo real, externo ao computador, e o processo refere-se a uma série finita de operações que são realizadas a partir destes dados, a fim de transformá-los em alguma informação desejada disponibilizada ao usuário final através de um dispositivo de saída, por exemplo, o monitor de vídeo.

## Método printf()

Uma saída pode ser composta por informações textuais individuais e na forma de tabelas, valores numéricos resultantes de cálculos, constantes e pelo conteúdo de variáveis definidas no programa. Estas informações ou resultados são apresentados, principalmente, na tela do computador e servem como forma de verificar se o problema foi resolvido.

A classe System oferece atributos e métodos que possibilitam obter uma referência às operações de entrada e saída. O atributo estático out, por exemplo, representa o fluxo padrão de saída de dados. Já o método printf(), incorporado à classe System a partir do Java 1.5, permite exibir dados formatados de forma equivalente à função de saída “printf” da linguagem C.

O método printf(), utilizado para realizar uma saída de dados no fluxo de saída padrão System.out, tem a seguinte forma geral:

System.out.printf(expressão_de_controle, argumento1, argumento2, ...);

A expressão_de_controle deve ser uma sequência de caracteres (portanto, delimitada por aspas duplas) que determina as informações que serão mostradas na tela. Nesta expressão podem existir dois tipos de informações: caracteres comuns e códigos de controle (ou especificadores de formato).

Os códigos de controle, mostrados na Tabela 1, são precedidos por um % e são aplicados aos argumentos na mesma ordem em que aparecem. É importante destacar que deve existir para cada código de controle um argumento correspondente.

| Código | Formato (tipo de dados) |
|---|---|
| %c | Caractere simples (char) |
| %s | Cadeia de caracteres (String) |
| %d | Inteiro decimal com sinal (int) |
| %i | Inteiro decimal com sinal (int) |
| %ld | Inteiro decimal longo (long) |
| %f | Real em ponto flutuante (float ou double) |
| %e | Número real em notação científica com o “e” minúsculo (float ou double) |
| %E | Número real em notação científica com o “E” maiusculo (float ou double) |
| %% | Imprimir o próprio caractere % |

Tabela 1: Códigos de controle ou especificadores de formato

Outro elemento que pode ser acrescentado à expressão_de_controle, usada no método printf(), são os caracteres de escape utilizados com o seu significado original alterado. Por exemplo, o caractere aspas dupla (“) é utilizado para delimitar uma cadeia de caracteres, desta forma, para usá-lo como parte do conteúdo da própria cadeia deve-se transformá-lo em um caractere de escape, alterando assim sua finalidade inicial. A Tabela 2 mostra os principais caracteres de escape, que são sempre iniciados por uma barra invertida (\).

| Caractere de escape | Significado |
|---|---|
| \n | Nova linha |
| \t | Tabulação horizontal (o mesmo que pressionar a tecla Tab) |
| \” | Aspas dupla |
| \\ | Barra invertida |

Tabela 2: Principais caracteres de escape

## Usando largura e precisão na saída

A formatação de uma informação na saída de dados pode ser incrementada especificando-se a largura e a precisão do campo de saída. O formato desejado é determinado acrescentando valores que indicam o tamanho e a precisão entre o símbolo % e o código de controle, por exemplo:

**%3d** - mostra o valor justificando à direita em três posições (ou colunas) da tela do computador:

**%8.2f** - indica que um número de ponto flutuante será mostrado justificado à direita dentro de um campo de saída de tamanho oito e com duas casas decimais de precisão fazendo arredondamentos (%8.2f mostra valores no formato 99999,99):

```
//trecho de código:
double a = 135.4528;
double b = 23050.568;
double c = 5.0;
System.out.printf("Variavel 'a' = %8.2f\n", a);
System.out.printf("Variável 'b' = %8.2f\n", b);
System.out.printf("Variável 'c' = %8.2f\n", c);
```

# Formatação usando Printf()

**printf()** usa especificadores de formato para formatação. Há certos tipos de dados mencionados abaixo:

- Para formatação de números
- Formatando números decimais
- Para formatação boolean
- Para formatação de strings
- Para formatação de char

## i). Para formatação de números

O número em si inclui Inteiro, Longo, etc. O especificador de formatação usado é %d.

Abaixo está a implementação do método acima:

```
import java.io.*;


// Driver Class
class Main {
    // main function
    public static void main (String[] args) {
    int a=10000;


        //System.out.printf("%,d%n",a);
        System.out.printf("%,d%n",a);
    }
}
```

## ii). Para formatação de números decimais

A formatação de números decimais pode ser feita usando print() e o especificador de formato %f .

Abaixo está a implementação do método acima:

```
import java.io.*;

// Driver Class
class Main {
    // main function
  public static void main(String[] args)
    {
    // declaring double
    double a = 3.14159265359;

    // Printing Double Value with
    // different Formatting
    System.out.printf("%f\n", a);
    System.out.printf("%5.3f\n", a);
    System.out.printf("%5.2f\n", a);
    }
}
```

## iii). Para formatação booleana

A formatação booleana pode ser feita usando printf e ( '%b' ou '%B' ) dependendo do resultado necessário.

Abaixo está a implementação do método acima:

```
import java.io.*;

// Driver Function
class Main {
    // main function
    public static void main(String[] args)
    {
    int a = 10;
    Boolean b = true, c = false;
    Integer d = null;

    // Fromatting Done using printf
    System.out.printf("%b\n", a);
    System.out.printf("%B\n", b);
    System.out.printf("%b\n", c);
    System.out.printf("%B\n", d);
    }
}
```

## iv). Para formatação de caracteres

A formatação de caracteres é fácil de entender, pois precisa de printf() e os especificadores de formato de caracteres usados são '%c' e '%C'.

Abaixo está a implementação do método acima:

```
import java.io.*;

// Driver Class
class Main {
    // main function
    public static void main(String[] args)
    {
    char c = 'g';

    // Formatting Done
    System.out.printf("%c\n", c);

    // Converting into Uppercase
    System.out.printf("%C\n", c);
    }
}
```

## v). Para formatação de strings

A formatação de strings requer conhecimento de strings e do especificador de formato usado '%s' e '%S'.

Abaixo está a implementação do método acima:

```
import java.io.*;

// Driver Class
class Main {
    // main function
    public static void main(String[] args)
    {
    String str = "uma string";

    // Formatting from lowercase to
    // Uppercase
    System.out.printf("%s \n", str);
    System.out.printf("%S \n", str);

    str = "EDVC";
    // Vice-versa not possible
    System.out.printf("%S \n", str);
    System.out.printf("%s \n", str);
    }
}
```

# Método print() e println()

Usuários recorrem à utilização de programas de computador pela necessidade de informações rápidas para auxiliar no processo de tomada de decisão. Portanto, a tarefa de exibir os resultados na tela do computador acaba se constituindo em uma das principais etapas do processamento de dados.

Outros métodos disponíveis na classe System para realizar uma saída de dados no monitor de vídeo são: **print()** e **println()**. Basicamente, o método println() imprime na tela uma cadeia de caracteres (ou String) que é enviada ao método como argumento usando a seguinte forma geral:

```
System.out.println(String);
```

A diferença entre os métodos print() e println() acontece uma vez que o método println() sempre pulará uma linha após mostrar o valor do seu argumento.

Se o java é a sua primeira linguagem de programação, uma das primeiras coisas que aprenderá é como imprimir texto na tela. É uma habilidade simples, mas essencial, que estabelece a base para muitas tarefas mais complexas. Neste capítulo, exploraremos as diferentes maneiras como Java pode manipular a saída, focando nos métodos comumente usados, como System.out.println() e System.out.print(), e como eles funcionam. Ao longo do caminho, veremos exemplos e detalharemos alguns dos principais conceitos que podem ajudar você a entender melhor a abordagem do Java para manipular a saída.

**Nota Importante:**

| |
|---|
| É importante saber que estas não são as únicas formas de saída de dados, que podem ser usadas em java, as outras são mais avançadas, e não serão abordadas neste livro. |

# Imprimindo texto em Java

Uma das coisas mais fundamentais em qualquer linguagem de programação é imprimir informações, o que é crucial para depuração, exibição de resultados ou interação com o usuário. Java fornece vários métodos para realizar isso, e dois dos mais amplamente usados são println() e print().

## Método println()

Em Java, o método **System.out.println()** é a função mais comumente usada para imprimir saída no console. Ele adiciona automaticamente uma nova linha após imprimir o texto, o que o torna ótimo para exibir linhas individuais de texto de forma limpa e clara.

Por exemplo:

System.out.println( "Olá, mundo!" );

O método **println()** é perfeito para exibir texto onde cada declaração ou valor precisa aparecer em uma linha separada. Você pode encadear várias chamadas **println()** para diferentes partes de texto ou variáveis, e o Java

imprimirá cada uma em uma nova linha.

System.out.println( "Olá, mundo!" );

System.out.println( "Estou aprendendo Java." );

System.out.println( "É incrível!" );

## Compreendendo aspas duplas em Java

Uma regra importante ao imprimir texto em Java é que o texto deve ser colocado entre aspas duplas (` ” ” `). Essas aspas dizem ao Java que o valor incluído é uma string , que é uma sequência de caracteres.

Aqui está um exemplo:

```
System.out.println( "Isso será impresso corretamente." );
```

Entretanto, se você esquecer de adicionar aspas duplas ao redor do seu texto, o Java gerará um erro porque não reconhecerá a entrada como uma string:

```
System.out.println(Isso produzirá um erro);
```

Lembre-se, sempre que estiver trabalhando com strings em Java, aspas duplas são essenciais. Sem elas, Java trata o texto como código e resultará em erros de compilação.

# O método  System.out.print()

Embora **println()** seja ótimo para imprimir texto em linhas separadas, há situações em que você pode querer imprimir várias coisas na mesma linha . É aqui que System.out.print() é útil.

O método print() funciona exatamente como println() , exceto que ele não insere uma nova linha após a saída. Isso permite que você imprima vários pedaços de texto na mesma linha, tornando-o mais eficiente para alguns casos de uso.

Observe que ambas as linhas de texto aparecem na mesma linha. Esta é a principal diferença entre print() e println() . Se você não quiser que o texto seja impresso em linhas separadas, print() é seu método preferido.

## Por que usar println() com mais frequência?

Embora print() seja útil em certos cenários, println() é mais comumente usado, especialmente quando você está apenas começando. O motivo é simples: ele torna o código e a saída mais fáceis de ler. Na maioria dos casos, você vai querer que cada pedaço de informação seja claramente separado em linhas diferentes, e println() lida com isso automaticamente.

Veja este exemplo:

```
System.out.println("Nome: John");

System.out.println("Idade: 25");

System.out.println("Localização: Nova York");
```

Cada pedaço de informação aparece em uma nova linha, tornando-a clara e fácil de ler. Se você usasse print() , a saída ficaria desorganizada e mais difícil de seguir.

### Combinando texto e variáveis

Em Java, você pode facilmente combinar texto e variáveis ao imprimir. Isso facilita a saída de texto estático e valores dinâmicos na mesma instrução. Aqui está um exemplo:

```
int  idade  =  25 ;

System.out.println( "Eu tenho " + idade + " anos." );
```

O operador ` + ` é usado aqui para concatenar a string e a variável inteira , permitindo que elas sejam impressas juntas. Você pode usar essa técnica para combinar qualquer tipo de variável com strings para uma saída mais personalizada.

## Melhores práticas para saída legível

Use **println()** para maior clareza: Como mencionado anteriormente, **println()** torna o código e a saída mais fáceis de ler, especialmente ao exibir várias linhas de informação. É o método padrão para a maioria das tarefas de impressão em Java, especialmente para iniciantes.

Insira espaços onde necessário: Se você estiver usando **print()**, frequentemente precisará adicionar espaços manualmente para garantir que a saída seja legível.

## Técnicas avançadas de impressão

Conforme você progride em sua jornada Java, você encontrará maneiras mais avançadas de imprimir texto e dados. Por exemplo, printf() é outro método disponível em Java, se você vem de uma linguagem como C você já está familiarizado com o printf(), usado para saída formatada. Embora seja mais complexo do que print() ou println() , ele permite que você tenha mais controle sobre o formato do texto impresso, como especificar o número de casas decimais em números de ponto flutuante.

# Melhores Práticas

- Concatenação de Strings: Use o operador (+) para concatenar strings e variáveis dentro de System.out.println. Garanta o uso correto de parênteses para evitar problemas de precedência em expressões.

- Depuração: utilize System.out.println para depuração imprimindo valores de variáveis em diferentes pontos do seu código para rastrear alterações e identificar problemas.

- Desempenho: tenha cuidado com o uso excessivo de System.out.println em aplicativos de desempenho crítico, pois isso pode tornar a execução mais lenta devido às operações de E/S.

- Uso printf para formatação: para saída formatada, considere usar System.out.printf, que permite mais controle sobre o formato de saída.

- Consideração de Nova Linha: Lembre-se de que System.out.println anexa automaticamente um caractere de nova linha. Use System.out.print se quiser continuar imprimindo na mesma linha.

# Lista de Exercícios Propostos

1. Calculadora Simples:
   - Receba dois números e uma operação do usuário (+, -, *, /).
   - Exiba o resultado da operação.

2. Cadastro de Usuário:
   - Receba nome, idade e e-mail do usuário.
   - Exiba as informações formatadas.

3. Conversor de Moedas:
   - Receba um valor em reais e uma taxa de câmbio.
   - Exiba o valor convertido para dólares.

4. Lista de Compras:
   - Receba itens do usuário e armazene-os em um array.
   - Exiba os itens após o usuário terminar.

5. IMC:
   - Receba o peso e altura do usuário.
   - Calcule e exiba o Índice de Massa Corporal (IMC).

6. Crie um programa que receba o nome e a idade de um usuário e exiba em uma frase formatada.

7. Leia o nome completo do usuário e exiba apenas as iniciais.

8. Faça um programa que leia uma linha de texto e conte quantos caracteres ela possui.

9. Receba três números do usuário e exiba o maior deles.

10. Peça ao usuário para digitar um número inteiro e exiba o dobro dele.

11. Leia o valor do salário de um usuário e calcule o aumento de 10%.

12. Faça um programa que receba a altura e largura de um retângulo e calcule a área.

13. Peça ao usuário para digitar uma frase e converta-a para maiusculas.

14. Receba a data de nascimento de um usuário e exiba a idade.

15. Crie um programa que leia dois números e determine se o primeiro é divisível pelo segundo.

16. Receba o nome, idade e cidade de um usuário em uma única linha e exiba-os separadamente.

17. Leia uma lista de números em uma única entrada e exiba a soma deles.

18. Receba um nome e um número. Repita o nome tantas vezes quanto o número informado.

19. Faça um programa que leia dois números inteiros e exiba a média deles.

20. Receba um número decimal e arredonde-o para duas casas decimais.

21. Leia três números inteiros e determine se formam um triângulo.

22. Solicite ao usuário que digite um parágrafo e exiba o número de palavras.

23. Receba uma sequência de números separados por vírgulas e calcule a média.

24. Leia um número inteiro e converta-o para binário.

25. Peça ao usuário que insira um número decimal e converta-o para inteiro.

26. Leia um caractere do teclado e verifique se é uma vogal.

27. Receba dois números e calcule o Mínimo Múltiplo Comum (MMC).

28. Leia uma sequência de números até que o usuário digite zero e exiba a soma total.

29. Peça ao usuário que insira um valor em Fahrenheit e converta para Celsius.

30. Leia um número inteiro e exiba a tabuada dele de 1 a 10.

31. Receba um número e verifique se é primo.

32. Faça um programa que leia o nome de um arquivo e exiba o número de caracteres.

33. Leia uma linha de entrada e conte o número de vogais.

34. Leia um nome e exiba-o letra por letra em linhas separadas.

35. Exiba uma sequência de números de 1 a 5, cada um em uma linha diferente.

36. Leia uma palavra e exiba-a com espaços entre as letras.

37. Use `print()` para exibir um parágrafo em uma única linha.

38. Leia duas frases e exiba-as concatenadas em uma linha.

39. Peça ao usuário para digitar cinco números e exiba-os em uma linha separada por vírgulas.

40. Faça um programa que exiba o texto “Olá” seguido por “Mundo!” na mesma linha.

41. Crie um programa que simule um sistema de login com autenticação.

42. Leia os valores de entrada separados por espaço e calcule o produto deles.

43. Leia dois números e determine o MDC (Máximo Divisor Comum).

# MÓDULO 6: OPERADORES

**Os conceitos e princípios fundamentais da ciência são invenções livres do espírito humano.**

Albert Einstein

- OPERADOR ARITMÉTICO
- OPERADORES UNÁRIOS
- OPERADORES DE ATRIBUIÇÃO
- OPERADOR RELACIONAL
- OPERADORES LÓGICOS
- EXERCÍCIOS

# OBJECTIVOS

- Explorar os diversos operadores em Java, como aritméticos, lógicos e relacionais.
- Explicar o funcionamento de operadores como `ternário`, `incremento` e `decremento`.
- Demonstrar a criação de expressões lógicas com base na tabela verdade.

# Tipos de Operadores

Os tipos operadores em Java são fundamentais para o funcionamento de qualquer aplicação. Eles permitem manipular variáveis e realizar diversas operações matemáticas, lógicas e comparativas. Seja para a criação de expressões aritméticas, condições lógicas ou para controlar o fluxo de execução, entender os operadores em Java é essencial para qualquer desenvolvedor.

Neste capítulo, exploraremos detalhadamente os tipos de operadores que o Java oferece e suas usabilidades. Vamos discutir os operadores aritméticos, operadores de atribuição, operadores relacionais, operadores lógicos, operadores unitários, e os operadores bit a bit. Se você é novo em Java, vamos detalhar cada operador e exemplos de uso.

## O que são operadores em Java?

Operadores em Java são símbolos especiais que realizam operações em variáveis e valores. Java suporta vários tipos de operadores que são usados para realizar cálculos e comparações entre valores.

Alguns dos principais tipos de operadores incluem:

- Operadores Aritméticos, Operadores de Atribuição
- Operadores Relacionais, Operadores Lógicos
- Operadores Unários, Operadores Bit a Bit
- Operador Ternário

## Por que precisamos de operadores Java?

Eles permitem a tomada de decisões no código por meio de comparação e operadores lógicos, direcionando o fluxo de execução com base em condições. Eles são usados para manipular dados em níveis de bits e bytes, permitindo um processamento de dados eficiente.

# Operador Aritmético

Os operadores aritméticos são usados para realizar operações matemáticas, como soma, subtração, multiplicação e divisão. Esses operadores são muito comuns no desenvolvimento de programas que envolvem cálculos.

| Operadores | Nome | Exemplo |
|---|---|---|
| + | Adição | 10 + 5 = 15 |
| - | Subtração | 10 – 5 = 5 |
| * | Multiplicação | 10 * 5 = 50 |
| / | Divisão | 10 / 5 = 2 |
| % | Módulo (resto da divisão) | 10 % 5 = 0 |

Exemplo de Código:

```
class Addition {
    public static void main(String[] args){
        // inicialização de variáveis
        int num1 = 10, num2 = 20, soma = 0;

         // Exibir num1 e num2
        System.out.println("num1 = " + num1);
        System.out.println("num2 = " + num2);

         // adicionando num1 e num2
        soma = num1 + num2;
        System.out.println("The sum = " + soma);
    }
}
```

# Operadores Unários

Operadores unários em Java requerem apenas um operando. Eles são usados para incrementar/decrementar e negociar um valor. Também é possível inverter um valor booleano com operadores unários. A tabela descrita abaixo mostra diferentes operadores unários e o que eles fazem.

| Operador | Nome | Descrição |
|---|---|---|
| + | Unário mais | +um |
| - | Unário menos | -um |
| ++ | Incremento | a++ ou ++a |
| -- | Decremento | a-- ou --a |
| ! | Negação lógica | !true é falso |

Exemplo de Código:

```
class OperadorUnario {
    public static void main(String[] args){
        int n1 = 20;

        // Imprimir a variável acima
        System.out.println("Number = " + n1);

        // Efetuar uma operação unária
        n1 = -n1;
        System.out.println("Result = " + n1);
    }
}
```

# Vantagens de usar os operadores unários

A principal vantagem de usar operadores unários em Java é que eles fornecem uma maneira simples e eficiente de modificar o valor de uma variável. Algumas vantagens específicas de usar operadores unários são:

1. Conciso e fácil de usar: operadores unários são simples de usar e exigem apenas um operando. Eles são fáceis de entender e tornam o código mais legível e conciso.

2. Mais rápido que outros operadores: operadores unários são mais rápidos que outros operadores, pois exigem apenas um operando. Isso os torna ideais para operações que precisam ser executadas rapidamente, como incrementar um contador.

3. Pré e Pós-Incremento/Decremento: Operadores unários fornecem opções de pré e pós-incremento e decremento, o que os torna úteis para uma variedade de casos de uso. Por exemplo, o operador de pré-incremento pode ser usado para incrementar o valor de uma variável antes de usá-la em uma expressão, enquanto o operador de pós-incremento pode ser usado para incrementar o valor de uma variável após usá-la em uma expressão.

4. Modificando tipos primitivos: Operadores unários podem ser usados para modificar o valor de tipos primitivos, como int, long, float, double, etc.

No geral, os operadores unários fornecem uma maneira simples e eficiente de executar operações em variáveis em Java e podem ser usados em diversos cenários para tornar o código mais legível e conciso.

# Operadores de Atribuição

Esses operadores são usados para atribuir valores a uma variável. O operando do lado esquerdo do operador de atribuição é uma variável, e o operando do lado direito do operador de atribuição é um valor. O valor do lado direito deve ser do mesmo tipo de dados do operando do lado esquerdo. Caso contrário, o compilador gerará um erro. Isso significa que os operadores de atribuição têm associatividade da direita para a esquerda, ou seja, o valor fornecido no lado direito do operador é atribuído à variável à esquerda. Portanto, o valor do lado direito deve ser declarado antes de usá-lo ou deve ser uma constante. O formato geral do operador de atribuição é,

```
operador variável valor;
```

## Tipos de operadores de atribuição em Java

Operador de Atribuição Simples: O Operador de Atribuição Simples é usado com o sinal “=” onde o lado esquerdo consiste no operando e o lado direito consiste em um valor. O valor do lado direito deve ser do mesmo tipo de dados que foi definido no lado esquerdo.

Operador de atribuição composto: O operador composto é usado onde +,-,* e / são usados junto com o operador =.

Vamos dar uma olhada em cada um dos operadores de atribuição e como eles operam:

## Operador (=):

Este é o operador de atribuição mais direto, que é usado para atribuir o valor à direita à variável à esquerda. Esta é a definição básica de um operador de atribuição e como ele funciona.

Sintaxe:

```
int numero;
String nome;

numero = 10;
nome = "edvaldo";

System.out.println("numero associado: " + numero);
System.out.println("nome designado: " + nome);
```

## Operador (+=):

Este operador é um composto dos operadores '+' e '='. Ele opera adicionando o valor atual da variável à esquerda ao valor à direita e então atribuindo o resultado ao operando à esquerda.

```
int num1 = 10, num2 = 20;

System.out.println("num1 = " + num1);
System.out.println("num2 = " + num2);

num1 += num2;
System.out.println("num1 = " + num1);
```

**Nota:**

O operador de atribuição composto em Java realiza conversão de tipo implícita. Vamos considerar um cenário em que x é uma int variável com um valor de 5.

## Operador (-=):

Este operador é um composto dos operadores '-' e '='. Ele opera subtraindo o valor da variável à direita do valor atual da variável à esquerda e então atribuindo o resultado ao operando à esquerda.

```
int num1 = 10, num2 = 20;
System.out.println("num1 = " + num1);
System.out.println("num2 = " + num2);

num1 -= num2;
System.out.println("num1 = " + num1);
```

## Operador (*=):

Este operador é um composto dos operadores '*' e '='. Ele opera multiplicando o valor atual da variável à esquerda pelo valor à direita e então atribuindo o resultado ao operando à esquerda.

```
int num1 = 10, num2 = 20;

System.out.println("num1 = " + num1);
System.out.println("num2 = " + num2);

num1 *= num2;
System.out.println("num1 = " + num1);
```

## Operador (/=):

Este operador é um composto dos operadores '/' e '='. Ele opera dividindo o valor atual da variável à esquerda pelo valor à direita e então atribuindo o quociente ao operando à esquerda.

```
int num1 = 20, num2 = 10;
System.out.println("num1 = " + num1);
System.out.println("num2 = " + num2);

num1 /= num2;
System.out.println("num1 = " + num1);
```

## Operador (%=):

Este operador é um composto dos operadores '%' e '='. Ele opera dividindo o valor atual da variável à esquerda pelo valor à direita e então atribuindo o restante ao operando à esquerda.

```
int num1 = 5, num2 = 3;

System.out.println("num1 = " + num1);
System.out.println("num2 = " + num2);

num1 %= num2;
System.out.println("num1 = " + num1);
```

Esta Tabela apresenta todos operadores de atribuição:

| Operador | Nome | Descrição |
|---|---|---|
| = | Tarefa simples | Um = b |
| += | Adicionar e atribuir | a += b(equivale a  a = a+b) |
| -= | Subtrair e atribuir | a -= b(equivale a  a = a-b) |
| *= | Multiplicar e atribuir | a *= b(equivale a  a = a*b) |
| /= | Dividir e atribuir | a /= b(equivale a  a = a/b) |
| %= | Módulo e Atribuição | a %= b(equivale a  a = a%b) |

# Operador Relacional

Operadores relacionais (ou de comparação) em Java são usados para comparar dois valores ou expressões, resultando em um valor booleano ( true ou false). Esses operadores são cruciais para controlar o fluxo de execução no programa, como em instruções condicionais ( if , else ) e loops ( while , for).

| Operador | Nome | Exemplo |
|---|---|---|
| == | Igual a | 5 == 5 resulta em verdadeiro |
| != | Não é igual a | 5 != 4 resulta em verdadeiro |
| > | Maior que | 5 > 3 resulta em verdadeiro |
| < | Menor que | 3 < 5 resulta em verdadeiro |
| >= | Maior ou Igual a | 5 >= 5 resulta em verdadeiro |
| <= | Menor ou Igual a | 3 <= 5 resulta em verdadeiro |

Exemplo de Código:

```
class Main {
    public static void main(String[] args)
    {
        int var1 = 10, var2 = 20, var3 = 5;

        System.out.println("Var1 = " + var1);
        System.out.println("Var2 = " + var2);
        System.out.println("Var3 = " + var3);

        System.out.println("var1 < var2: " + (var1 < var2));
        System.out.println("var2 < var3: " + (var2 < var3));
    }
}
```

# Operadores Lógicos

Operadores lógicos em Java são usados para combinar múltiplas expressões booleanas em uma única expressão, o que também resulta em um valor booleano ( true ou false). Esses operadores são essenciais para controlar o fluxo de execução por meio de condições complexas em construções de programação como instruções if, loops e casos de switch.

| Operador | Nome | Descrição |
|---|---|---|
| && | E lógico | Retorna verdadeiro somente se ambos os operandos forem verdadeiros. |
| \|\| | OU lógico | Retorna true se pelo menos um dos operandos for true. Se ambos os operandos forem false, o resultado será false. |
| ! | Não lógico | Inverte o valor verdade do operando. |

Exemplo de Código:

```
 public class Main {
    public static void main(String[] args) {
    boolean x = true;
    boolean y = false;

    System.out.println(x && y); // saída: false
    System.out.println(x || y); // saída: true
    System.out.println(!x);    // saída: false
    }
}
```

# Operador Ternário

Um operador ternário em Java é uma ferramenta de expressão condicional que permite que você atribua valores com base em uma condição. É uma versão abreviada da instrução if-else. Pode ser usado em vários cenários de programação onde uma decisão rápida entre dois valores é necessária com base em uma condição.

variável = condição ? expressãoVerdadeiro : expressãoFalso;

Por exemplo, digamos que você queira verificar qual dos dois números é maior e atribuir o maior número a uma variável:

```
int a = 5;

int b = 10;

int max = (a > b) ? a : b;
```

Neste exemplo,

condição: a > b - verifica se a é maior que b.

expressãoVerdadeiro: a - se a for maior que b, a será atribuída a max .

expressãoFalso: b - se a não for maior que b, b será atribuída a max .

Então, Max conterá o valor 10 porque b é maior que a.

## Exemplos mostrando o uso do operador ternário

Exemplo 1: Determinando o Máximo de Dois Números

Declaração do problema: Simplifique o processo de encontrar o máximo de dois números usando o operador ternário.

```
public class MaxOfTwo {
    public static void main(String[] args) {
    int a = 5;
    int b = 10;

    int max = (a > b) ? a : b;

    System.out.println(" O máximo é: " + max);
       // O máximo é: 10
    }
}
```

# Operadores de Bitwise e Bit Shift

A linguagem de programação Java também fornece operadores que realizam operações bitwise e bit shift em tipos integrais. Os operadores discutidos nesta seção são menos comumente usados. Portanto, sua cobertura é breve; a intenção é simplesmente fazer com que você saiba que esses operadores existem.

O operador de complemento bit a bit unário " ~" inverte um padrão de bits; ele pode ser aplicado a qualquer um dos tipos integrais, tornando cada "0" um "1" e cada "1" um "0". Por exemplo, a byte contém 8 bits; aplicar esse operador a um valor cujo padrão de bits é "00000000" mudaria seu padrão para "11111111".

O operador de deslocamento esquerdo com sinal " <<" desloca um padrão de bits para a esquerda, e o operador de deslocamento direito com sinal " >>" desloca um padrão de bits para a direita. O padrão de bits é dado pelo operando da esquerda, e o número de posições a deslocar pelo operando da direita. O operador de deslocamento direito sem sinal " " >>>desloca um zero para a posição mais à esquerda, enquanto a posição mais à esquerda depois ">>"depende da extensão do sinal.

- O **&** operador bit a bit executa uma operação **AND** bit a bit.
- O **^** operador bit a bit executa uma operação **OU** exclusiva bit a bit.
- O **|** operador bit a bit executa uma operação **OR** inclusiva bit a bit.

```
int bitmask = 0x000F;

int val = 0x2222;

// SAÍDA "2"

System.out.println(val & bitmask);
```

# Precedência e Associatividade

A precedência do operador determina o agrupamento de termos em uma expressão. Isso afeta como uma expressão é avaliada. Certos operadores têm precedência maior do que outros; por exemplo, o operador de multiplicação tem precedência maior do que o operador de adição, Por exemplo, x = 7 + 3 * 2; aqui, x recebe 13, não 20, porque o operador * tem precedência maior que +, então ele primeiro é multiplicado por 3 * 2 e depois somado a 7.

Aqui, operadores com a precedência mais alta aparecem no topo da tabela, aqueles com a mais baixa aparecem na parte inferior. Dentro de uma expressão, operadores com precedência mais alta serão avaliados primeiro.

| Categoria | Operador | Associatividade |
|---|---|---|
| Sufixo | expressão++ expressão-- | Da esquerda para a direita |
| Unário | ++expressão --expressão +expressão -expressão ~ ! | Da direita para a esquerda |
| Multiplicativo | * / % | Da esquerda para a direita |
| Aditivo | + - | Da esquerda para a direita |
| Shift | << >> >>> | Da esquerda para a direita |
| Relacional | < > <= >= instância de | Da esquerda para a direita |
| Igualdade | == != | Da esquerda para a direita |
| E bit a bit | & | Da esquerda para a direita |
| XOR bit a bit | ^ | Da esquerda para a direita |
| OU bit a bit | \| | Da esquerda para a direita |
| E lógico | && | Da esquerda para a direita |
| OU lógico | \|\| | Da esquerda para a direita |
| Condicional | ?: | Da direita para a esquerda |
| Atribuição | = += -= *= /= %= ^= \|= <<= >>= >>>= | Da direita para a esquerda |

# Classe Math em Java

A Classe Math em Java faz parte do java.lang pacote e fornece uma coleção de métodos para executar operações numéricas básicas, como cálculos exponenciais, logarítmicos, de raiz quadrada e trigonométricos. É uma classe utilitária que contém métodos estáticos, o que significa que você pode chamar esses métodos sem criar uma instância da Math classe.

## Uso da classe Math

A Math classe é usada para executar operações e cálculos matemáticos de forma eficiente e precisa. É amplamente usada em aplicações que exigem cálculos matemáticos.

### Métodos comuns

- **Math.abs(double a)**: Retorna o valor absoluto de um double valor.
- **Math.sqrt(double a)**: Retorna a raiz quadrada positiva corretamente arredondada de um double valor.
- **Math.pow(double a, double b)**: Retorna o valor do primeiro argumento elevado à potência do segundo argumento.
- **Math.max(double a, double b)**: Retorna o maior entre dois double valores.
- **Math.min(double a, double b)**: Retorna o menor dos dois double valores.
- **Math.random()**: Retorna um double valor com sinal positivo, maior ou igual a 0.0 e menor que 1.0.

### Métodos adicionais

- **Math.log(double a)**: Retorna o logaritmo natural (base e) de um double valor.

- **Math.exp(double a)**: Retorna o número de Euler e elevado à potência de um double valor.
- **Math.sin(double a)**: Retorna o seno trigonométrico de um ângulo (em radianos).
- **Math.cos(double a)**: Retorna o cosseno trigonométrico de um ângulo (em radianos).
- **Math.tan(double a)**: Retorna a tangente trigonométrica de um ângulo (em radianos).

Exemplo 1: Calculando a raiz quadrada

```
public class Main {
    public static void main(String[] args) {
        double num = 25.0;
        double raiz = Math.sqrt(num);
        System.out.println("Raiz quadrada de " + num +
" e: " + raiz);
    }
}
```

Neste exemplo, o **Math.sqrt()** é usado para calcular a raiz quadrada de 25,0, resultando em uma saída de 5,0.

Exemplo 2: Encontrando Máximo e Mínimo

```
public class MathMaxMinExample {
    public static void main(String[] args) {
    double num1 = 45.6;
    double num2 = 67.4;
    System.out.println("Maximum of num1 and num2: "
+ Math.max(num1, num2));
    System.out.println("Minimum of num1 and num2: "
+ Math.min(num1, num2));
    }
}
```

Este exemplo demonstra o uso de **Math.max()** e **Math.min()** para encontrar o máximo e o mínimo de dois números, 45,6 e 67,4.

Exemplo 3: Gerando números aleatórios

```
public class MathRandomExample {
    public static void main(String[] args) {
    double randomValue = Math.random();
    System.out.println("Random value between 0.0 and
1.0: " + randomValue);
    }
}
```

Aqui, Math.random() é usado para gerar um double valor aleatório entre 0,0 (inclusivo) e 1,0 (exclusivo).

# Notas Importantes:

**Precisão**: Tenha cuidado com a precisão ao usar operações de ponto flutuante, pois os resultados podem variar ligeiramente devido ao arredondamento.

**Métodos estáticos**: como todos os métodos da Math classe são estáticos, você pode chamá-los diretamente usando o nome da classe sem criar uma instância.

**Desempenho**: Use Math métodos para melhor desempenho e precisão em cálculos matemáticos em comparação à implementação manual de algoritmos.

**Valores aleatórios**: para gerar números aleatórios dentro de um intervalo específico, você pode dimensionar e deslocar o resultado de Math.random():

**Evite estouros**: tenha cuidado com possíveis estouros ao usar métodos como Math.pow(), especialmente com expoentes grandes.

# Melhores Práticas

- Use parênteses: use parênteses para garantir a ordem desejada das operações.

- Código legível: evite expressões complexas que podem tornar o código menos legível.

- Evite divisão por zero: sempre verifique se há zero antes de realizar operações de divisão.

- Use operadores lógicos com cuidado: certifique-se de que os operadores lógicos sejam usados corretamente para evitar resultados inesperados.

- Operações bit a bit: tenha cuidado ao usar operadores bit a bit, especialmente com números assinados.

# Lista de Exercícios Propostos

1. Considere o seguinte trecho de código.

int i = 10;

int n = i++%5;

a. Quais são os valores de i e n depois que o código é executado?

b. Quais são os valores finais de i e n se em vez de usar o operador de incremento pós-fixado (i++), você usar a versão prefixada (++i))?

2. Para inverter o valor de a boolean, qual operador você usaria?

3. Qual operador é usado para comparar dois valores, =ou ==?

4. Explique o seguinte exemplo de código:result = someCondition ? value1 : value2;

5. Implemente um programa para calcular a área de um trapézio, onde:

h = altura

b = base menor

B = base maior

Área = (h . (b + B)) / 2

6. Faça o programa acima calcular utilizando valores de ponto flutuante e depois imprima na tela

duas informações:

Valor exato da área:

Valor arredondado para inteiro:

7. Calcule o valor das seguintes equações:

a. 3 – 2 – 1 + 2 + 1 + 3

b. 2 * 3 – 4 * 5

c. 2 + 6 – 3 / 7 * 9

d. 3 % 4 – 8

8. Indique qual o valor verdade das seguintes expressões:

a. (1 > 2) // exemplo: false

b. (8 == 8) // exemplo: true

c. ((12 – 5) > 6)

d. (0 < 3) && (8 < 9)

e. ((i++) > i)

f. ((10 * 90 / 50 – 2) == 16)

9. Escreva as expressões abaixo na forma convencional:

A. a + b + ((34+e*9)/u-89 ^ (1/2)) =

B. 12+1/((4*a)/45) ^ (1/2) =

C. ((a+x) ^ (2+w)-3*a)/2 =

D. (12*x)/(36-9 ^ y) =

E. (-5 + 96 * x - raizq(w-1)) ^ (1/y)

10. Determine o resultado lógico das expressões mencionadas, indicando se são verdadeiras ou falsas. Considere para as respostas os seguintes valores x=1, a=3, b=5, c=8 e d=7:

A. nao(x > 3)=

B. (x < 1) e nao(b > d) =

C. nao(d < 0) e (c > 5) =

D. nao(x > 3) e (c < 7) =

E. (a > b) ou (c > b) =

F. (x >= 2) =

G. (x < 1) e (b >= d) =

H. (d < 0) ou (c > 5) =

I. nao(d > 3) ou nao(b < 7) =

J. (a > b) ou nao(c > b) =

11. Desenvolva os algoritmos, seus respectivos diagramas de bloco e sua codificação em português estruturado:

- a) Ler uma temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit. A fórmula de conversão é F (9 * C + 160) / 5, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

- b) Ler uma temperatura em graus Fahrenheit e apresentá-la convertida em graus Celsius. A fórmula de conversão é C = (F - 32) * (5/9), sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

- c) Calcular e apresentar o valor do volume de uma lata de óleo, utilizando a fórmula: Volume = p* Raio2 * Altura

- d) Efetuar o cálculo da quantidade de litros de combustível gasta em uma viagem, utilizando um automóvel que faz 12Km por litro. Para obter o cálculo, o usuário deve fornecer o tempo gasto (TEMPO) e a velocidade média (VELOCIDADE) durante a viagem. Desta forma, será possível obter a distância percorrida com a fórmula DISTÂNCIA = TEMPO * VELOCIDADE. Possuindo o valor da distância, basta calcular a quantidade de litros de combustível utilizada na viagem com a fórmula LITROS_USADOS + DISTÂNCIA / 12. Ao final, o

programa deve apresentar os valores da velocidade média (VELOCIDADE), tempo gasto na viagem (TEMPO), a distância percorrida (DISTÂNCIA) e a quantidade de litros (LITROS_USADOS) utilizada na viagem.

- e) Efetuar o cálculo e a apresentação do valor de uma prestação em atraso, utilizando a fórmula PRESTACAO = VALOR + (VALOR * TAXA/100) * TEMPO).
- f) Ler dois valores (inteiros, reais ou caracteres) para as variáveis A e B, e efetuar a troca dos valores de forma que a variável A passe a possuir o valor da variável B e a variável B passe a possuir o valor da variável A. Apresentar os valores trocados

- g) Ler quatro números inteiros e apresentar o resultado da adição e multiplicação, baseando-se na utilização do conceito da propriedade distributiva. Ou seja, se forem lidas as variáveis A, B, C, e D, devem ser somadas e multiplicadas A com B, A com C e A com D. Depois B com C, B com D e por fim C com D. Perceba que será necessário efetuar seis operações de adição e seis operações de multiplicação e apresentar doze resultados de saída.

- h) Elaborar um programa que calcule e apresente o volume de uma caixa retangular, por meio da fórmula VOLUME = COMPRIMENTO * LARGURA * ALTURA.

- i) Ler dois inteiros (variáveis A e B) e imprimir o resultado do quadrado da diferença do primeiro valor pelo segundo.

- j) Elaborar um programa que efetue a apresentação do valor da conversão em real de um valor lido em dólar. O programa deve solicitar o valor da cotação do dólar e também a quantidade de dólares disponível com o usuário, para que seja apresentado o valor em moeda brasileira.

- k) Elaborar um programa que efetue a apresentação do valor da conversão em dólar de um valor lido em real. O programa deve solicitar o valor da cotação do dólar e também a quantidade de reais disponível com o usuário, para que seja apresentado o valor em moeda americana.

- l) Elaborar um programa que efetue a leitura de três valores (A, B e C) e apresente como resultado final a soma dos quadrados dos três valores lidos.

- m) Elaborar um programa que efetue a leitura de três valores (A, B e C) e apresente como resultado final o quadrado da soma dos três valores lidos.

12. Crie um programa que peça dois números ao usuário e demonstre o uso dos operadores básicos (+, -, *, /, %).

13. Implemente um programa que verifique se dois números são iguais usando o operador relacional `==`.

14. Receba dois números e use operadores relacionais para determinar qual é maior.

15. Solicite ao usuário que digite dois números e demonstre o uso de operadores lógicos (`&&`, `||`, `!`).

16. Faça um programa que peça três números e determine se todos são iguais ou diferentes.

17. Leia dois números e calcule o resultado de todas as operações aritméticas possíveis entre eles.

18. Faça um programa que receba um número inteiro e exiba a soma de seus dígitos.

19. Leia dois números e calcule a potência do primeiro elevado ao segundo (`Math.pow()`).

20. Receba dois números e calcule o resto da divisão do primeiro pelo segundo.

21. Leia três números e calcule a média aritmética deles.

22. Peça um número ao usuário e use o operador de incremento (`++`) para exibir o valor antes e depois da operação.

23. Leia um número e use o operador de decremento (`--`) para reduzir seu valor.

24. Demonstra o comportamento de pré-incremento e pós-incremento em uma mesma expressão.

25. Receba um número e use o operador unário negativo (`-`) para inverter seu sinal.

26. Leia um número e determine se ele é positivo ou negativo usando operadores unários.

27. Crie um programa que demonstre o uso dos operadores de atribuição (`+=`, `-=`, `*=`, `/=`, `%=`).

28. Leia dois números e use o operador de atribuição composto para calcular a soma acumulada.

29. Faça um programa que inicialize uma variável com um valor e modifique seu valor usando operadores de atribuição.

30. Demonstre o uso do operador de atribuição para concatenar strings.

31. Leia um número e multiplique-o sucessivamente por um fator, usando `*=`.

32. Leia dois números e determine se o primeiro é maior que, menor que ou igual ao segundo.

33. Faça um programa que peça dois números e determine se um é múltiplo do outro.

34. Crie um programa que peça uma idade e verifique se a pessoa é maior de idade (>= 18).

35. Leia três números e verifique se formam um triângulo válido usando operadores relacionais.

36. Receba dois números e determine se ambos são pares ou ímpares.

37. Implemente um programa que determine se uma pessoa pode votar (idade >= 18) e dirigir (possui habilitação) ao mesmo tempo.

38. Crie um programa que verifique se um número é divisível por 2 e 3, mas não por 5.

39. Leia dois números e determine se pelo menos um deles é negativo usando operadores lógicos.

40. Peça três notas de um aluno e verifique se ele passou (média >= 7) ou ficou de recuperação (média >= 5).

41. Receba dois números e determine o maior deles usando um operador ternário.

42. Peça a idade de uma pessoa e exiba "Criança", "Adolescente" ou "Adulto" usando múltiplos operadores ternários.

43. Implemente um programa que determine se um número é par ou ímpar usando operador ternário.

44. Leia três números e determine o maior deles com um operador ternário aninhado.

45. Faça um programa que determine se um aluno foi aprovado ou reprovado com base em sua nota final (>= 6).

46. Crie um programa que receba dois números inteiros e aplique os operadores `&`, `|` e `^` (bitwise).

47. Implemente um programa que leia um número e realize um deslocamento de bits para a direita e para a esquerda.

48. Leia dois números e determine os bits que eles têm em comum usando o operador `&`.

49. Faça um programa que leia um número inteiro e determine se ele é potência de 2 usando operadores bitwise.

50. Crie um programa que implemente uma simples criptografia XOR em um texto fornecido pelo usuário.

# MÓDULO 7:  CONTROLE DE FLUXO

**Minhas limitações são apenas erros de programação prestes a serem corrigidos.**

Leonardo V. Castro

- INSTRUÇÕES DE DECISÃO
- IF & ELSE
- WHILE
- DO – WHILE
- FOR
- BREAK & CONTINUE
- EXERCÍCIOS

# Objetivos

- Ensinar estruturas de decisão como if, else e switch.
- Trabalhar com laços de repetição (while, do-while, for).
- Explicar o uso de instruções especiais como break e continue.

# Introdução

Antes de escrever um programa para resolver uma determinada questão, é fundamental ter um completo entendimento do problema e um método cuidadosamente planejado para resolvê-lo. Os dois capítulos seguintes analisam técnicas que facilitarão o desenvolvimento de programas computacionais estruturados.

# Tomada de decisão em Java

A tomada de decisão em programação é semelhante à tomada de decisão na vida real. Na programação também enfrentamos algumas situações em que queremos que um certo bloco de código seja executado quando alguma condição for cumprida.

Uma linguagem de programação usa instruções de controle para controlar o fluxo de execução de um programa com base em certas condições. Elas são usadas para fazer com que o fluxo de execução avance e se ramifique com base em mudanças no estado de um programa.

Instruções de seleção do Java:

- if , if-else , nested-if , if-else-if
- switch-case , break, continue, return

A tomada de decisão é uma parte importante da vida. Muitas vezes você tem que tomar decisões que decidirão suas próximas ações. Também fazemos isso na programação. Geralmente, o código é executado de cima para baixo. Mas às vezes você tem que tomar algumas decisões que decidirão qual bloco de código será executado primeiro e qual depois.

As instruções de fluxo de controle vêm em seu auxílio neste momento, e permitem que você controle o fluxo da execução do código no seu programa. Na linguagem de programação Java, você pode controlar o fluxo de execução do código colocando a tomada de decisão, ramificação, looping e adicionando blocos condicionais.

Com base nisso, podemos classificar os tipos de instruções de fluxo de controle da seguinte forma:

1. Declarações de tomada de decisão
2. Instruções de loop
3. Declarações de ramificação

# Declaração if em Java

A declaração if do Java é a declaração de tomada de decisão mais simples. Ela é usada para decidir se uma determinada declaração ou bloco de declarações será executado ou não, ou seja, se uma determinada condição for verdadeira, então um bloco de declarações é executado, caso contrário, não.

Sintaxe:

```
if(condição) {
   // Instruções a serem executadas se
   // a condição for verdadeira
}
```

A estrutura de seleção if é usada para se fazer uma escolha entre várias linhas de ação alternativas. Por exemplo, suponha que o grau de aprovação em um exame é 60. A instrução seguinte em pseudocódigo

Se o grau do aluno for maior que ou igual que 60 Imprimir "Aprovado"

determina se a condição "grau do aluno for maior que ou igual a 60" é verdadeira ou falsa. Se a condição for verdadeira, a palavra "**Aprovado**" é impressa e a instrução que vem logo após o pseudocódigo é "realizada" (lembre-se de que o pseudocódigo não é uma linguagem real de programação). Se a condição for falsa, a impressão é ignorada e a instrução que vem logo após o pseudocódigo é realizada. Observe que a segunda linha dessa estrutura de seleção está recuada (indentada). Tal recuo é opcional, mas altamente recomendado por destacar a estrutura inerente dos programas estruturados. Aplicaremos criteriosamente as convenções para os recuos (indentações) ao longo deste texto. O compilador java ignora caracteres em branco (whitespace

characters) como os caracteres de espaço, tabulação e nova linha usados em recuos e no espaçamento vertical.

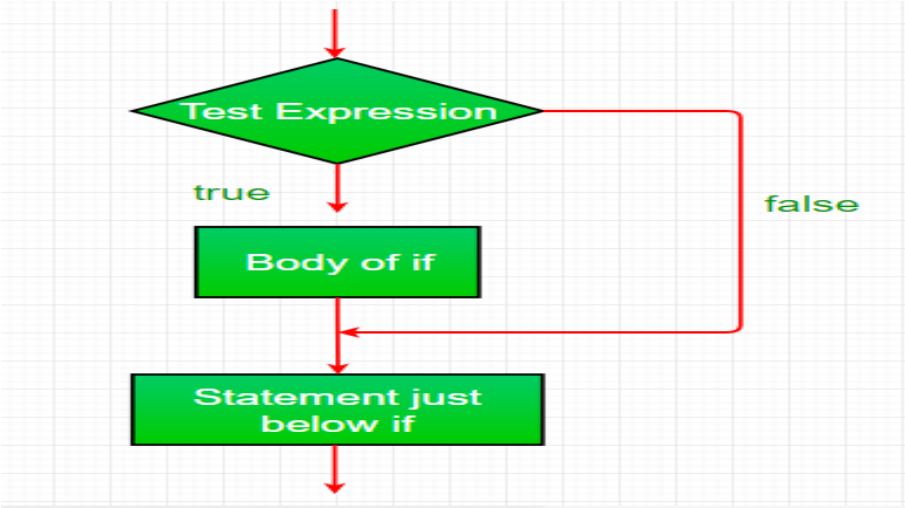
A instrução If do pseudocódigo anterior pode ser escrita em java como

```
if (grau >= 60)

System.out.printf("Aprovado\n");
```

Observe que o código em java tem muita semelhança com a versão do pseudocódigo em inglês. Esta é uma das propriedades do pseudocódigo que o torna uma grande ferramenta para o desenvolvimento de programas.

Fluxograma if declaração:




**Operação:** A condição após a avaliação da instrução if será true ou false. A instrução if em Java aceita valores booleanos e se o valor for true, então ele executará o bloco de instruções abaixo dele.

```
if(condição)

   declaração1;

   declaração2;
```

// Aqui, se a condição for verdadeira, o bloco if considerará a declaração

// abaixo dela, ou seja, declaração1, e a declaração2 não será considerada no bloco if, ela ainda será executada

// pois não é afetada por nenhuma condição if.

**Observação:**

| |
|---|
| se não fornecermos as chaves '{' e '}' depois de if( condition ), por padrão a instrução if considerará a instrução imediata como estando dentro de seu bloco. |

```
// Java program to illustrate the if else statement
public class IfElseExample {
    public static void main(String[] args) {
    boolean a = true;
    boolean b = false;

    if (a) {
        System.out.println("a is true");
    } else {
        System.out.println("a is false");
    } if (b) {
        System.out.println("b is true");
    } else {
        System.out.println("b is false");
       }
    }
}
```

Saída

a é verdadeiro

b é falso

Explicação:

- O código começa com a declaração de duas variáveis booleanas a e b, com a definida como verdadeira e b definida como falsa.

- A primeira instrução if-else verifica o valor de a. Se o valor de a for true, o código dentro do primeiro conjunto de chaves {} é executado e a mensagem “a is true” é impressa no console. Se o valor de a for false, o código dentro do segundo conjunto de chaves {} é executado e a mensagem “a is false” é impressa no console.

- A segunda instrução if-else verifica o valor de b da mesma forma. Se o valor de b for true, a mensagem “b is true” é impressa no console. Se o valor de b for false, a mensagem “b is false” é impressa no console.

- Este código demonstra como usar uma instrução if-else para tomar decisões com base em valores booleanos. Ao usar uma instrução if-else, você pode controlar o fluxo do seu programa e executar o código somente sob certas condições. O uso de valores booleanos em uma instrução if-else fornece uma maneira simples e flexível de tomar essas decisões.

## Vantagens da instrução If else

1. **Execução condicional**: A declaração if-else permite que o código seja executado condicionalmente com base no resultado de uma expressão booleana. Isso fornece uma maneira de tomar decisões e controlar o fluxo de um programa com base em diferentes entradas e condições.

2. **Legibilidade**: A instrução if-else torna o código mais legível ao indicar claramente quando um bloco de código específico deve ser executado. Isso torna mais fácil para outros entenderem e manterem o código.

3. **Reusabilidade**: Ao usar instruções if-else, os desenvolvedores podem escrever código que pode ser reutilizado em diferentes partes do programa. Isso reduz a quantidade de código que precisa ser escrito e mantido, tornando o processo de desenvolvimento mais eficiente.

4. **Depuração:** A instrução if-else pode ajudar a simplificar o processo de depuração, tornando mais fácil rastrear problemas no código. Ao indicar claramente quando um bloco de código específico deve ser executado, fica mais fácil determinar por que um pedaço específico de código não está funcionando como esperado.

5. **Flexibilidade:** A instrução if-else fornece uma maneira flexível de controlar o fluxo de um programa. Ela permite que os desenvolvedores lidem com diferentes cenários e respondem dinamicamente a mudanças nas entradas do programa.

No geral, a declaração if-else é uma ferramenta fundamental na programação que fornece uma maneira de controlar o fluxo de um programa com base em condições. Ela ajuda a melhorar a legibilidade, a reutilização, a depuração e a flexibilidade do código.

**Boa prática de programação:**

| |
|---|
| Uma instrução longa pode ser dividida em várias linhas. Se uma instrução deve ocupar mais de uma linha, escolha locais de divisão que façam sentido (como após uma vírgula em uma lista separada por vírgulas). Se uma instrução for dividida em duas ou mais linhas, recue todas as linhas subsequentes. |

**Erro comum de programação:**

| |
|---|
| Colocar um ponto-e-vírgula imediatamente à direita do parêntese direito depois de uma condição em uma estrutura if. |

# If-else em Java

A tomada de decisão em Java ajuda a escrever declarações orientadas a decisões e executar um conjunto particular de código com base em certas condições. A declaração if sozinha nos diz que se uma condição for verdadeira, ela executará um bloco de declarações e se a condição for falsa, ela não executará. Neste capítulo, aprenderemos sobre Java if-else. If-else juntos representam o conjunto de instruções condicionais em Java que são executadas de acordo com a condição que é verdadeira.

Sintaxe da declaração if-else

```
if (condição) {
   // Executa este bloco se
  // a condição for verdadeira
} else {
    // Executa este bloco se
   // a condição for falsa
}
```

A estrutura de seleção if realiza uma ação indicada somente quando a condição for verdadeira; caso contrário a ação é ignorada. A estrutura de seleção if/else permite ao programador especificar que sejam realizadas ações diferentes conforme a condição seja verdadeira ou falsa. Por exemplo, a instrução em pseudocódigo

Se o grau do aluno for maior que ou igual a 60

Imprimir "Aprovado " senão

Imprimir "Reprovado"

Imprimir **Aprovado** se o grau do aluno for maior que ou igual a 60 e Imprime **Reprovado** se o grau do aluno for menor que 60. Em qualquer um dos casos, depois de a impressão acontecer, a instrução que vem após o 3 pseudocódigo na sequência é realizada. Observe que o corpo de senão (else) também está recuado. A convenção de recuo escolhida, qualquer que seja ela, deve ser aplicada criteriosamente em todos os seus programas. É difícil ler um programa que não obedeça a convenções de espaçamento uniforme.

A estrutura If/else do pseudocódigo anterior pode ser escrita em java como

```
if (grau >= 60)
System.out.printf("Aprovado \n"); else
System.out.printf("Reprovado \n");
```

Relembrando o operador ternário, a linguagem Java fornece o operador condicional (?:) que está intimamente relacionado com a estrutura if/else. O operador condicional é o único operador ternário do java — ele utiliza três operandos. Os operandos e o operador condicional formam uma expressão condicional. O primeiro operando é uma condição, o segundo, o valor de toda a expressão condicional se a condição for verdadeira, e o terceiro, o valor de toda a expressão condicional se a condição for falsa. Por exemplo, a instrução System.out.printf()

System.out.printf("%s\n", grau >= 60 ? "Aprovado" : "Reprovado");

contém uma expressão condicional que assume o valor da string literal **"Aprovado"** se a condição **grau >= 60** for verdadeira e assume o valor " Reprovado" se a condição for falsa.

A string de controle de formato de `System.out.printf()` contém a especificação de conversão %s para imprimir uma string de caracteres.

Assim, a instrução `System.out.printf()` anterior realiza essencialmente o mesmo que a instrução if/else precedente.

Os valores em uma expressão condicional também podem ser ações a serem executadas. Por exemplo, a expressão condicional

```
grau >= 60 ? System.out.printf("Aprovado\n") : System.out.printf("Reprovado\n");
```

é lida desta forma, "Se grau for maior ou igual a 60 então execute printf (" Aprovado\n"), caso contrário execute printf ("Reprovado\n"). Isso também faz o mesmo que a estrutura if/else anterior. Veremos que os operadores condicionais podem ser usados em algumas situações em que instruções if/else não podem.

## Boa prática de programação

Consulte a tabela de precedência dos operadores ao escrever expressões que contêm muitos operadores. Certifique-se de que os operadores da expressão são executados na ordem adequada. Se você não tiver certeza quanto à ordem de cálculo de uma expressão complexa, use parênteses para impor aquela ordem, exatamente do modo como efeito em expressões algébricas. Não se esqueça de levar em consideração que alguns operadores em java, como o operador de atribuição (=), são associados da direita para a esquerda e não da esquerda para a direita.

# Instruções de Decisão Encadeadas

Em Java, podemos usar instruções if aninhadas para criar lógica condicional mais complexa. Instruções if aninhadas são instruções if dentro de outras instruções if.

Sintaxe:

```
if (condição1) {
    // bloco de código 1
    if (condição2) {
    // bloco de código 2
    }
}
```

Estruturas if/else aninhadas verificam vários casos inserindo umas estruturas if/else em outras. Por exemplo, a instrução em pseudocódigo a seguir imprimirá A para graus de exame maiores que 90,B para graus maiores que ou iguais a 80,C para graus maiores que ou iguais a 70,D para graus maiores que ou iguais a 60 e F para todos os outros graus.

```
Se o grau do aluno for maior que ou igual a 90
            Imprimir "A " senão
Se o grau do aluno for maior que ou igual a 80
            Imprimir "B" senão
Se o grau do aluno for maior que ou igual a 70
            Imprimir "C" senão
Se o grau do aluno for maior que ou igual a 60
            Imprimir "D" senão
            Imprimir "F"
```

Esse pseudocódigo pode ser escrito em java como

```
if (grau >= 90)
        System.out.printf("A\n");
        else
if (grau >= 80)
    System.out.printf("B\n");
        else
if (grau >= 70)
    System.out.printf("C\n");
        else
if (grau >= 60)
    System.out.printf("D\n");
        else
        System.out.printf("F\n");
```

Se a variável grau for maior que ou igual a 90, as quatro primeiras condições serão verdadeiras, mas somente a instrução printf colocada antes do primeiro teste será executada. Depois de aquele printf ser executado, a porção else do if/else "externo" é ignorada. Muitos programadores em java preferem escrever a estrutura i f anterior como

```
if (grau >= 90)
        System.out.printf("A\n");
else if (grau >= 80)
        System.out.printf("B\n");
else if (grau >= 70)
        System.out.printf("C\n");
else if (grau >= 60)
        System.out.printf("D\n");
```

```
else
        System.out.printf("F\n");
```

Para o compilador, ambas as formas são equivalentes. A última forma é mais popular porque evita recuos muito grandes para a direita. Frequentemente, tais recuos deixam um espaço muito pequeno para uma linha, obrigando a que as linhas sejam divididas e prejudicando a legibilidade do programa.

A estrutura de seleção if deve conter apenas uma instrução em seu corpo. Para incluir várias instruções no corpo de um if, coloque o conjunto de instruções entre chaves ( { e }). Um conjunto de instruções dentro de um par de chaves é chamado uma instrução composta.

## O que é if-else em Java?

If-else são instruções condicionais que são executadas de acordo com a instrução correta executada. Se a condição if for verdadeira, então o bloco de código dentro da instrução if é executado, senão ele executa o bloco de instruções else.

O exemplo a seguir inclui uma instrução composta na porção else de uma estrutura if/else.

```
if (grau >= 60)
    System.out.printf("Aprovado.\n");
else {
    System.out.printf("Reprovado.\n");
System.out.printf("Voce deve fazer este curso
    novamente.\n");
}
```

Nesse caso, se o grau for menor que 60, o programa executa ambas as instruções `System.out.printf()` no corpo de else e imprime

```
Reprovado.
Voce deve fazer este curso novamente.
```

Observe as chaves em torno das duas instruções na cláusula else. Elas são importantes. Sem elas, a instrução

`System.out.printf(`"Voce deve fazer este curso novamente.\n");

estaria no lado de fora da porção else do if e seria executada independentemente de o grau ser menor que 60.

Os erros de sintaxe são acusados pelo compilador. Os erros lógicos produzem seus efeitos durante o tempo de execução. Um erro lógico fatal faz com que o programa falhe e termine prematuramente. Um erro lógico não-fatal permite que o programa continue a ser executado mas produz resultados incorretos.

- **Boa prática de programação**

| |
|---|
| Coloque uma linha em branco antes e após todas as estruturas de controle em um programa para melhorar sua legibilidade. |

- **Boa prática de programação**

| |
|---|
| Aplicar consistentemente as convenções para os recuos aumenta bastante a legibilidade do programa. Sugerimos valores fixos de aproximadamente ¼ da polegada ou três espaços em branco em cada nível de recuo. |

# Jogo Pedra Papel Tesoura

Pedra Tesoura Papel é um jogo que é jogado nos pátios das escolas e até eletronicamente nos casinos. A versão que se segue opõe humanos a computadores. Para jogar o jogo, introduz um número: 0 (pedra), 1 (tesoura) ou 2 (papel).

O computador seleciona aleatoriamente a sua jogada, também 0, 1 ou 2.

O jogo resulta numa vitória, na derrota ou num empate com base nas seguintes regras:

- Pedra quebra Tesoura (Pedra ganha).
- O papel cobre a pedra (o papel ganha).
- A Tesoura corta o Papel (a Tesoura ganha).
- Se ambos os jogadores escolherem a mesma letra, há um empate.

Por exemplo, se escolheres Pedra e o computador Papel, o computador ganha porque “O papel cobre a pedra”. Por outro lado, se escolheres Pedra e o computador Tesoura, então ganhas porque “A Pedra quebra a Tesoura”.

```
import java.util.Scanner;

public class PedraPapelTesoura {

      public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        final int PEDRA = 0, TESOURA = 1, PAPEL = 2;

        int jogador, computador; // humano vs computador

       System.out.println("Pedra:0; Tesoura:1; Papel:2 --
Escolha: ");
```

```
      jogador = input.nextInt();
      computador = (int)(3*Math.random());
       System.out.println("Escolha do Computador " +
computador);

System.out.println("**********************************
***");

     if (jogador == computador) {
        System.out.println("É um empate!");
     } else {
        if (jogador == PEDRA) {
           if (computador == TESOURA) {

System.out.println("Jogador:Pedra\nComputador:Tesoura\nVi
toria do Jogador");
           } else {

System.out.println("Jogador:Pedra\nComputador:Papel\nVito
ria do Computador.");
        }
          } else {
              if (jogador == TESOURA) {
                  if (computador == PEDRA) {

System.out.println("Jogador:Tesoura\nComputador:Pedra\nVi
toria do Computador.");
                  } else {

System.out.println("Jogador:Tesoura\nComputador:Papel\nVi
toria do Jogador.");
```

```
        }
           } else {
              if (computador == PEDRA) {

System.out.println("Jogador:Papel\nComputador:Pedra\nVito
ria do Jogador.");
              } else {

System.out.println("Jogador:Papel\nComputador:Tesoura\nVi
toria do Computador.");
                          }
                      }
                  }
              }
        }
}
```

Este código implementa um jogo simples de Pedra, Papel e Tesoura no qual um jogador humano joga contra o computador.

Aqui está uma explicação detalhada de como ele funciona:

1. Importação de Classes Necessárias

```
import java.util.Scanner;
```

A classe Scanner é usada para ler a entrada do jogador no console.

2. Definição da Classe e Método Principal

```
public class PedraPapelTesoura {
    public static void main(String[] args) {
```

Define a classe PedraPapelTesoura e seu método principal main, que é o ponto de entrada do programa.

3. Definição de Constantes

```
final int PEDRA = 0, TESOURA = 1, PAPEL = 2;
```

Constantes são definidas para representar os valores das jogadas:

- PEDRA = 0
- TESOURA = 1
- PAPEL = 2

Isso torna o código mais legível e fácil de entender.

4. Entrada do Jogador

```
System.out.println("Pedra:0; Tesoura:1;
Papel:2 -- Escolha: ");
jogador = input.nextInt();
```

O programa exibe as opções e lê a escolha do jogador como um número inteiro (0, 1 ou 2).

5. Escolha Aleatória do Computador

```
computador = (int)(3 * Math.random());
```

O computador escolhe sua jogada de forma aleatória:

Math.random() gera um número aleatório entre 0 (inclusive) e 1 (exclusivo).

Multiplicando por 3 e convertendo para inteiro (int), obtemos valores 0, 1 ou 2.

6. Exibição da Escolha do Computador

```
System.out.println("Escolha do Computador "
+ computador);
```

Mostra a escolha do computador.

7. Determinação do Resultado

O programa usa uma série de estruturas condicionais (if e else) para comparar as escolhas do jogador e do computador:

Caso de Empate:

```
if (jogador == computador)
    System.out.println("É um empate!");
```

Se a escolha do jogador for igual à do computador, o jogo termina empatado.

Caso do Jogador Escolher Pedra:

```
if (jogador == PEDRA)
    if (computador == TESOURA)
```

```
System.out.println("Jogador:Pedra\nComputador:T
esoura\nVitoria do Jogador");

else
System.out.println("Jogador:Pedra\nComputador:P
apel\nVitoria do Computador.");
```

Pedra vence Tesoura.

Pedra perde para Papel.

Caso do Jogador Escolher Tesoura:

```
if (jogador == TESOURA)

    if (computador == PEDRA)

System.out.println("Jogador:Tesoura\nComputador:Pedr
a\nVitoria do Computador.");

    else

System.out.println("Jogador:Tesoura\nComputador:Pape
l\nVitoria do Jogador.");
```

Tesoura vence Papel.

Tesoura perde para Pedra.

Caso do Jogador Escolher Papel:

```
if (computador == PEDRA)

System.out.println("Jogador:Papel\nComputador:P
edra\nVitoria do Jogador.");
```

```
else
System.out.println("Jogador:Papel\nComputador:T
esoura\nVitoria do Computador.");
```

Papel vence Pedra.

Papel perde para Tesoura.

8. Fluxo do Programa

- 1. O programa solicita a escolha do jogador.
- 2. Gera aleatoriamente a jogada do computador.
- 3. Exibe ambas as escolhas.
- 4. Usa as regras do jogo para determinar o vencedor.
- 5. Mostra o resultado no console.

## Pontos Importantes

1. Simplicidade: O código utiliza if-else de forma direta para avaliar as condições de vitória, derrota ou empate.

2. Aleatoriedade: A jogada do computador é gerada de forma imprevisível.

3. Interatividade: O programa pede a entrada do jogador e exibe o resultado em formato legível.

## Melhorias Possíveis

1. Validação da Entrada: Garantir que o jogador insira apenas 0, 1 ou 2.

2. Loop para Repetição: Permitir que o jogo seja repetido sem precisar reiniciar o programa.(ainda neste capítulo vamos ver sobre loops)

3. Melhor Interface: Mapear os números (0, 1, 2) para os nomes ("Pedra", "Papel", "Tesoura") ao exibir as escolhas.

# Estrutura de Seleção Múltipla Switch

O switch case Java é uma estrutura muito importante para testar condições de uma forma simples e intuitiva, reduzindo a necessidade de criar blocos de código complexo usando vários if else encadeados. No entanto, é preciso cuidado ao utilizá-la, pois qualquer pequeno erro na lógica empregada para definir as condições de teste pode causar diversos erros no programa. Para evitar que isso aconteça, é necessário entender como de fato o switch case funciona e quando é melhor utilizar essa estrutura no lugar do if else.

## O que é o Switch case Java e para que serve?

O switch case é uma estrutura de decisão usada quando precisamos testar condições para determinar qual função será executada em seguida. Assim, essa expressão nos permite substituir múltiplos “if else if”, tornando o código criado mais simples, legível e fácil de manter. Além disso, a estrutura switch case no Java trabalha com vários tipos de dados, como byte, short, long, string, int, enum, entre outros.

## Como funciona o Switch case Java?

O funcionamento da estrutura switch é bastante simples. Inicialmente, o valor da variável passada no switch é comparado com os valores fornecidos em cada case. Se um desses valores for igual ao valor da variável, o bloco de código do case em questão será executado.

As comparações da estrutura switch são sempre feitas de forma sequencial e não há limite para a quantidade de cases que cada switch pode ter. Além disso, a estrutura switch também pode conter as declarações opcionais **“break”** e **“default”**, que explicaremos melhor nos próximos tópicos.

## Qual a Sintaxe do Switch case Java?

Para utilizar o Switch case em java, usamos a seguinte sintaxe:

```
public class SwitchFallThroughExample {
    public static void main(String[] args) {
    int day = 3;
    switch (day) {
        case 1:
        case 2:
        case 3:
            System.out.println("Weekday");
            break;
        case 4:
        case 5:
            System.out.println("Weekend");
            break;
        default:
            System.out.println("Invalid day");
    }
    }
}
```

## Sendo que:

- **switch (expressão):** é onde passamos a variável de teste que servirá de referência para as comparações que o programa deve fazer;
- **case:** é onde definimos o valor que será comparado com a variável de teste e o código que será executado caso sejam compatíveis;

- **break:** é a declaração opcional de quebra;

- **default:** é a declaração opcional padrão, na qual definimos o código que deve ser executado, se nenhum dos valores declarados nos cases for compatível com o valor passado na variável de teste.

No entanto, ao usar a estrutura switch case é importante lembrar que:

- cada switch pode conter um ou N cases;
- os valores dos cases não podem ser repetidos, caso contrário ocorrerá um erro durante a compilação do código;
- o valor de cada case deve ser do mesmo tipo da expressão switch;
- os valores dos cases não podem ser variáveis;
- A variável passada no switch deve ser de um dos tipos aceitos pela estrutura (short, byte, long, int, enum, string).

## Switch case VS If else: quais as diferenças?

Assim como o if else, a declaração switch case funciona como uma estrutura de decisão condicional. Nesse caso, como podemos diferenciá-las e saber em quais casos usar cada uma delas? Para responder essa questão, devemos observar alguns pontos importantes no funcionamento dessas estruturas, como:

- **Velocidade:** dependendo da quantidade de casos que precisam ser testados no código, a velocidade de execução pode ser afetada. Por isso, se o número de casos for maior que 5, é interessante considerar utilizar a estrutura switch e não o if else;
- **Valores fixos ou valores booleanos:** o if else é melhor para testar condições booleanas, já a estrutura switch é melhor para testar valores fixos;
- **Legibilidade do código:** em muitos casos, o switch case ajuda a manter o código limpo, pois evita que uma quantidade muito grande de if else if sejam usados no programa;

- **Expressão de teste:** Outro ponto importante que deve ser analisado é a expressão que será usada como base para o teste condicional. Isso porque o switch case aceita apenas valores fixos para teste. Por outro lado, o if else também é capaz de testar faixas de valores e condições além de valores fixos.

Agora você já sabe o que é e como funciona a estrutura switch case Java. Trata-se de um elemento simples e prático de usar, que pode ser muito útil quando precisamos substituir blocos de if else para melhorar a clareza e legibilidade de um código!

## Melhores Práticas

- **Use break Statements:** Sempre inclua break declarações, a menos que seja desejado fall-through intencional. Isso previne a execução não intencional de casos subsequentes.

- **Caso padrão:** inclua um default para lidar com valores inesperados e melhorar a robustez do código.

- **Tipos de expressão:** a switch expressão pode ser do tipo byte, short, int, char, String, ou uma enumeração.

- **Evite lógica complexa:** use switch para cenários simples com valores discretos. Para condições complexas, considere usar if-else instruções.

- **Melhorias no Java 12+:** No Java 12 e versões posteriores, as expressões switch permitem retornar valores e usar a yield palavra-chave para obter um código mais conciso.

# Laços de Repetição

Looping em linguagens de programação é um recurso que facilita a execução de um conjunto de **instruções/funções** repetidamente enquanto alguma condição é avaliada como verdadeira. Java fornece três maneiras de executar os loops. Embora todas as maneiras forneçam funcionalidade básica semelhante, elas diferem em sua sintaxe e tempo de verificação de condição. Java fornece três tipos de instruções condicionais: o segundo tipo é a instrução de loop.

**while loop:** Um while loop é uma declaração de fluxo de controle que permite que o código seja executado repetidamente com base em uma dada condição booleana. O while loop pode ser pensado como uma declaração if repetitiva.

Sintaxe:

```
enquanto (condição booleana) {
   instruções de loop...
}
```

**for loop:** for loop fornece uma maneira concisa de escrever a estrutura do loop. Diferentemente de um while loop, uma instrução for consome a inicialização, condição e incremento/decremento em uma linha, fornecendo assim uma estrutura de loop mais curta e fácil de depurar.

```
for (condição de inicialização; condição de teste; incremento/decremento){
    declarações)
}
```

**do while:** o loop do while é semelhante ao loop while com a única diferença de que ele verifica a condição após executar as instruções e, portanto, é um exemplo de loop de controle de saída.

```
do {
    declarações..
} while (condição);
```

# Estrutura de Repetição While

Uma estrutura de repetição permite ao programador especificar que uma ação deve ser repetida enquanto uma determinada condição for verdadeira.

A instrução em pseudocódigo:

```
Enquanto houver mais itens em minha lista de compras
Comprar o próximo item e riscá-lo de minha lista
```

descreve uma repetição que ocorre durante a realização das compras. A condição "houver mais itens em minha lista de compras" pode ser verdadeira ou falsa. Se for verdadeira, a ação "Compre o próximo item e risque-o da minha lista" é realizada. Essa ação será realizada repetidamente enquanto a condição permanecer verdadeira. A instrução (ou instruções) contida na estrutura de repetição while constitui o corpo do while. O corpo da estrutura while pode ser uma instrução simples ou composta. Posteriormente, a condição se tornará falsa (quando o último item da lista for comprado e riscado na lista). Nesta ocasião, a repetição termina, e é executada a primeira instrução do pseudocódigo colocada logo após a estrutura de repetição.

Como exemplo de um while real, considere o trecho de programa destinado a encontrar a primeira potência de 2 maior que 1000. Suponha que a variável inteira do produto foi inicializada com o valor 2. Quando a estrutura de repetição a seguir terminar sua execução, produto conterá a resposta procurada:

```
produto = 2;
while (produto <= 1000) produto = 2 * produto;
```

O loop while começa com a verificação da condição booleana. Se for avaliado como verdadeiro, as instruções do corpo do loop são executadas, caso contrário, a primeira instrução após o loop é executada. Por esse motivo, também é chamado de loop de controle de

entrada, uma vez que a condição é avaliada como verdadeira, as instruções no corpo do loop são executadas. Normalmente, as instruções contêm um valor de atualização para a variável que está sendo processada para a próxima iteração. Quando a condição se torna falsa, o loop termina, marcando o fim do seu ciclo de vida.

## Formulando Algoritmos: Estudo de Caso 1 (Repetição Controlada por Contador)

Para ilustrar como os algoritmos são desenvolvidos, resolvemos várias formas de um problema de média de notas de uma turma de alunos. Considere o seguinte enunciado de um problema:

Uma turma de dez alunos fez um teste. Os graus, (inteiros variando de 0 a 100) do teste estão disponíveis para você. Determine a média da turma no teste.

A média da turma é igual à divisão da soma dos graus pelo número de alunos. O algoritmo para resolver esse problema em um computador deve receber cada um dos graus, realizar o cálculo da média e imprimir o resultado. Vamos usar o pseudocódigo, listar as ações a serem executadas e especificar a ordem em que elas devem ser realizadas. Usamos a repetição controlada por contador para obter um grau de cada vez. Essa técnica usa uma variável chamada contador para especificar o número de vezes que um conjunto de instruções deve ser executado. Nesse exemplo, a repetição termina quando o contador supera 10.  Na próxima seção, mostramos como os algoritmos de pseudocódigo são desenvolvidos.

A repetição controlada por contador é chamada normalmente repetição definida porque o número de repetições é conhecido antes de o loop começar a ser executado.

Observe as referências no algoritmo a um total e a um contador. Um total é uma variável usada para acumular uma série de valores. Um

contador é uma variável usada para contar — neste caso, para contar o número de graus fornecidos. Normalmente, as variáveis usadas para armazenar totais devem ser inicializadas com o valor zero antes de serem utilizadas em um programa; caso contrário a soma incluiria os valores anteriores armazenados no local de memória daquele total. Normalmente as variáveis que servem para contadores são inicializadas com o valor zero ou um, dependendo de seu uso (apresentaremos exemplos mostrando cada um desses usos).

Uma variável não-inicializada contém um valor "lixo" — o último valor armazenado no local da memória reservado para aquela variável.

Definir o total com valor 0

Definir contador com valor 1

Enquanto o contador de graus for menor que ou igual a dez

Obter próximo grau

Adicionar o grau ao total

Adicionar um ao contador de graus

Definir a média da turma com o valor total dividido por dez, imprimir a média da turma. Algoritmo do pseudocódigo que usa repetição controlada por contador para resolver o problema da média da turma.

```
public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

            int contador, grau, total, media;
            /* fase de inicialização */
            total = 0;
            contador = 1;
```

```
                /* fase de processamento */
                while (contador <= 10) {
                    System.out.println("Entre com o grau:
");
                    grau = input.nextInt();

                    total = total + grau;
                    contador = contador + 1;
                }
                /* fase de terminação */
                media = total / 10;
            System.out.printf("A media da turma e %d\n",
media);
    }
}
```

saída:

Entre com o grau: 98

Entre com o grau: 7 6

Entre com o grau: 71

Entre com o grau: 87

Entre com o grau: 83

Entre com o grau: 90

Entre com o grau: 57

Entre com o grau: 79

Entre com o grau: 82

Entre com o grau: 94

A media da turma e 81

Observe que o cálculo da média no programa produziu um resultado inteiro. Na realidade, a soma dos graus nesse exemplo é 817 que ao ser dividido por 10 deveria levar a 81,7, i.e., um número com ponto decimal.

Veremos como lidar com tais números (chamados números de ponto flutuante).

- **Erros Comuns de Programação**

  - Tentar usar o operador de incremento ou decremento em uma expressão que não seja um nome simples de variável, e.g., escrever ++ (x+1) é um erro de sintaxe.

  - Usar números de ponto flutuante de uma maneira que os considere representados precisamente pode levar a resultados incorretos. Os números de ponto flutuante são representados apenas aproximadamente na maioria dos computadores.

  - Se um contador ou total não for inicializado, provavelmente os resultados de seu programa estarão incorretos. Esse é um exemplo de erro lógico.

  - Tentar usar o operador de incremento ou decremento em uma expressão que não seja um nome simples de variável, e.g., escrever ++ (x+1) é um erro de sintaxe.

  - Fornecer no corpo de uma estrutura while uma ação que posteriormente não torna falsa a condição no while. Normalmente, tal estrutura de repetição nunca terminará — este erro é chamado "loop infinito".

  - Colocar um ponto e vírgula (;) depois da condição em uma estrutura if leva a um erro lógico em estruturas if de seleção simples e um erro de sintaxe em estruturas if de seleção dupla.

  - Esquecer de uma ou ambas as chaves que delimitam uma instrução composta.

# Estrutura de Repetição Do/While

A estrutura de repetição **do/while** é similar à estrutura while. Na estrutura while, a condição de continuação do loop é testada em seu início antes de o corpo da estrutura ser executado. A estrutura **do/ while** testa a condição de continuação depois de o corpo do loop ser executado, portanto ele será executado pelo menos uma vez. Quando um **do/while** termina, a execução continua com a instrução após a cláusula while. Observe que não é necessário usar chaves na estrutura **do/while** se houver apenas uma instrução no corpo. Entretanto, normalmente as chaves são incluídas para evitar confusão entre as Estruturas while e do/while. Por exemplo,

**while (condição)** é considerado normalmente o cabeçalho para uma estrutura while. Uma estrutura do/while sem chaves em torno de um corpo com uma única instrução teria o seguinte aspeto.

e poderia causar alguma confusão. A última linha — while (condição); pode ser interpretada de maneira errada pelo leitor como uma estrutura while contendo uma instrução vazia. Desta forma, a instrução do/while com uma instrução é escrita frequentemente como se segue para evitar confusão:

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

- **Boa prática de programação**

| |
|---|
| Alguns programadores sempre incluem chaves em uma estrutura do/while mesmo que elas não sejam necessárias. Isso ajuda a eliminar a ambiguidade entre a estrutura do/while contendo uma instrução e a estrutura while. |

- **Erro comum de programação**

Acontecem loops infinitos quando a condição de continuação do loop em uma estrutura while, for ou do/while nunca se torna falsa. Para evitar isso, certifique-se de que não há um ponto-e-vírgula imediatamente após o cabeçalho de uma estrutura while ou for. Em um loop controlado por contador, certifique-se de que a variável de controle esteja sendo incrementada (ou decrementada) no corpo do loop. Em um loop controlado por sentinela, certifique-se de que o valor sentinela seja fornecido posteriormente.

O programa de Exemplo 9 usa uma estrutura do/while para imprimir os números de 1 a 10. Observe que a variável de controle contador é pré-incrementada no teste da condição de continuação do loop. Observe também o uso das chaves para encerrar um corpo com uma única instrução do do/while.

Exemplo 9: /* Usando a estrutura de repetição do/while */

```
public class Main {
    public static void main(String[] args) {
            int contador = 1;

            do {

                System.out.printf("%d ", contador);

            } while (++contador <= 10);
    }
}
```

Saída:

1 2 3 4 5 6 7 8 9 10

# Os Fundamentos da Repetição

Muitos programas envolvem repetições ou loops. Um loop é um grupo de instruções que o computador executa repetidamente enquanto alguma condição de continuação de loop permanecer verdadeira. Vimos duas maneiras de realizar uma repetição:

1. Repetição controlada por contador
2. Repetição controlada por sentinela

Algumas vezes, a repetição controlada por contador é chamada repetição definida porque sabemos antecipadamente quantas vezes o loop será executado. A repetição controlada por sentinela é chamada algumas vezes repetição indefinida porque não se sabe antecipadamente quantas vezes o loop será executado.

Em uma repetição controlada por contador, uma variável de controle é usada para contar o número de repetições. A variável de controle é incrementada (normalmente de 1) cada vez que o grupo de instruções é realizado. Quando o valor da variável de controle indicar que o número correto de repetições foi realizado, o loop é encerrado e o computador continua a execução do programa a partir da instrução imediatamente após o loop.

Os valores sentinelas são usados para controlar repetições quando:

1. O número exato de repetições não é conhecido de antemão e
2. O loop inclui instruções que obtêm dados cada vez que o loop é realizado.

O valor sentinela indica o "fim dos dados". A sentinela é entrada depois que todos os itens regulares de dados tiverem sido fornecidos ao

programa. As sentinelas devem ter valores diferentes daqueles dos itens regulares de dados.

# Repetição Controlada por Contador

A repetição controlada por contador exige:

1. O nome de uma variável de controle (ou contador do loop).
2. O valor inicial da variável de controle.
3. O incremento (ou decremento) pelo qual a variável de controle é
4. modificada cada vez que o loop é realizado.
4. A condição que testa o valor final da variável de controle (i.e., se o loop deve continuar).

A declaração

```
int contador = 1;
```

fornece o nome da variável de controle (contador), declara-a como sendo um inteiro, reserva espaço para ela e define seu valor inicial como 1. Esta declaração não é uma instrução executável.

A declaração e a inicialização de contador também poderiam ter sido realizadas com as instruções

```
int contador;
contador = 1;
```

A declaração não é executável, mas a atribuição é. Usamos ambos os métodos de inicializar variáveis. A instrução

```
++contador;
```

```
/* Repetição controlada por contador */
int contador = 1; /* inicialização */
while (contador <= 10) { /* condição de repetição */
       System.out.printf("%d\n", contador);
       ++contador; /* incremento */
```

```
}
```

incrementa o contador do loop de 1 cada vez que o loop é realizado. A condição de continuação do loop na estrutura while examina se o valor da variável de controle é menor que ou igual a 10 (o último valor para o qual a condição é verdadeira). Observe que o corpo do while é realizado mesmo quando a variável de controle é igual a 10. O loop termina quando a variável de controle se torna maior que 10 (i.e., quando contador se torna 11).

Normalmente, os programadores da linguagem java fariam o programa do exemplo acima, de uma forma mais concisa inicializando contador com o valor 0 e substituindo a estrutura while por

```
while (++contador <= 10)
printf ("%d\n", contador);
```

Este código economiza uma instrução porque o incremento é realizado diretamente na condição de while antes de ela ser examinada. Além disso, esse código elimina as chaves em torno do corpo do while porque agora o while contém apenas uma instrução. Fazer códigos de uma maneira condensada como essa exige alguma prática.

- **Erro comum de programação**

| |
|---|
| Como os valores em ponto flutuante podem ser valores aproximados, controlar a contagem de loops com variáveis de ponto flutuante pode resultar em valores imprecisos de contadores e exames incorretos da condição de terminação. |

- **<u>Boas prática de programação</u>**

  - Controle a contagem das repetições (loops) com valores inteiros
  - Faça um recuo nas instruções do corpo de cada estrutura de controle.

  - Coloque uma linha em branco antes e depois de cada uma das principais estruturas de controle para fazer com que ela se destaque no programa.
  - Muitos níveis de aninhamento podem fazer com que um programa fique difícil de entender. Como regra geral, tente evitar o uso de mais de três níveis de recuos

  - A combinação de espaçamento vertical antes e após as estruturas de controle e dos recuos do corpo daquelas estruturas dá aos programas um aspecto bidimensional que aumenta muito a sua legibilidade.

# Estrutura de Repetição For

O Loop For, em Java, é um dos três tipos de laço de repetição for como também é chamado. Essas estruturas são utilizadas para executar diversas vezes um mesmo bloco de instruções e a sequência de comandos é programada para voltar ao seu ponto de origem assim que completada, resumidamente.

É bem comum que muitos de nós tenhamos dificuldades para compreender como usar e manipular os **laços for**, devido à necessidade de alta abstração pelo usuário. É comum que iniciantes no mundo da programação tenham muitas dúvidas de como criar e em que momentos utilizar esses recursos.

Existem diversos motivos para uso de um Loop For em Java. Talvez o principal deles seja dar a capacidade ao desenvolvedor de realizar iterações em blocos de códigos sem precisar recorrer a um modelo manual, repetindo esses blocos cada vez que for necessário.

## O que é o Loop For Java e para que serve?

Quando executamos uma sequência de comandos na linguagem Java e precisamos que ela se repita até determinada condição seja contemplada, chamamos isso de “loop”. O Loop For, nesse caso, é responsável por iterar alguma parte de nosso código quando precisamos, não sendo necessário reescrevê-lo, automatizando, dessa maneira, qualquer incremento ou decremento de uma variável.

O laço for possibilitará a execução de determinada variável em nosso programa, a qual é iniciada pelo loop. Conforme nossos sistemas vão ficando mais complexos, a alteração de algumas dessas variáveis pode causar complicações durante seu desenvolvimento por um programador não muito experiente.

# Como funciona o Loop For Java?

O funcionamento do Loop For pode ter até quatro processos independentes entre si. O primeiro deles é a inicialização da variável contida no laço de iteração que pode ou não já estar inicializada. Depois da variável executada, temos a segunda parte de nosso loop, quando temos uma “verificação de condição”, a qual é realizada pelo loop de maneira contínua.

Essa verificação é o que define se o nosso laço for continuará ou será interrompido. Para facilitar seu entendimento, imagine que ao programarmos um sistema com variáveis entre 0 e 10 definidas randomicamente, e um laço for para conferir o valor de cada uma delas, queremos que nosso pequeno sistema observe se determinada variável é menor que 5, e adicione o valor de +1, em caso afirmativo.

Quando a variável for executada (e for igual ou maior que 5), ele passará para uma próxima, sem realizar alterações. Após verificar se a condição é satisfeita (ter um valor menor que 5), o loop terá uma ação a ser realizada (adicionar +1 ao valor daquela variável) essa é a terceira parte de nosso laço For. A verificação da primeira variável realizada pelo loop declarando se é menor que 5 tem duas possibilidades: **verdadeiro ou falso**.

Será a partir dela que nosso sistema será capaz de incrementar ou decrementar o valor daquela variável identificada como “menor do que 5”, ou terminará ali o laço de iteração e partirá para checagem da próxima variável no loop essa é a quarta e última parte do processo. Abordaremos essas questões com mais detalhes e outros exemplos mais adiante, no conteúdo.

# Sintaxe do For em Java

É possível encontrarmos diversas formas de iterar os elementos com laço for em Java, mas, basicamente, ela é escrita da seguinte forma:

```
for (int x = 0; x < y; x++) {

    ...

}
```

Podemos, também, representar a sintaxe de outra maneira:

```
for (inicialização da variável; verificação da
condição; incremento/decremento do valor da variável

{

    comando a ser executado/declaração

}
```

A estrutura de repetição for manipula automaticamente todos os detalhes da repetição controlada por contador. O programa funciona da seguinte maneira. Quando a estrutura for começa a ser executada, a variável de controle contador é inicializada com o **valor 1**. A seguir, a condição de continuação do loop **contador <= 10** é examinada. Como o valor inicial de contador é 1, a condição é satisfeita, portanto a instrução printf imprime o valor de contador, ou seja, 1.

A variável de controle contador é então incrementada pela expressão **contador++** e o loop começa novamente com seu teste de continuação. Como a variável de controle é agora igual a 2, o valor final não é excedido, e o programa realiza a instrução printf mais uma vez. O processo continua até que a variável de controle contador seja incrementada até o seu valor final de 11, isto faz com que o teste de continuação do loop não seja verdadeiro e a repetição termine. O programa continua com a execução da primeira instrução depois da estrutura for (neste caso, a instrução return no final do programa).

```
public class Main {

    public static void main(String[] args) {

     int contador;

 // inicialização, condição de repetição e
incremento

for (contador = 1; contador <= 10; contador++)

         System.out.printf("%d\n", contador);

   }

}
```

Exemplo: Repetição controlada por contador com a estrutura for.

Observe que a estrutura for "faz tudo" — ela especifica todos os itens necessários para a repetição controlada por contador com uma variável de controle. Se houver mais de uma instrução no corpo do for, devem ser usadas chaves para definir o corpo do loop. Observe que no exemplo acima, usa a condição de continuação do loop contador <= 10.

Se o programador escrever incorretamente no contador < 10, o loop só seria executado 9 vezes. Esse é um erro comum de lógica chamado off-by-one error (erro por falta de uma repetição).

- **Boa prática de programação**

| |
|---|
| Usar o valor final na condição de uma estrutura while ou for e usar o operador relacional <= evitará erros por falta de uma repetição (off-by one). Para um loop usado para imprimir os valores 1 a 10, por exemplo, a condição de continuação do loop deve ser contador <= 10 em vez de contador < 11 ou contador < 10. |

O formato geral da estrutura for é for (expressão1; expressão2; expressão 3) instrução onde expressão 1 inicializa a variável de controle do loop, expressão 2 é a condição de continuação do loop e expressão 3 incrementa a variável de controle. Na maioria dos casos, a estrutura for pode ser representada por uma estrutura while

equivalente da seguinte maneira:

```
for ( contador = 1; contador <= 10; contador ++)


    expressão1;
    while (expressão2) {
        instrução expressão 3;
    }
```

Frequentemente, a expressão 1 e a expressão 3 são listas de expressões separadas por vírgulas. As vírgulas são usadas aqui na

forma de operadores de vírgulas que garantem que a lista de expressões seja avaliada da esquerda para a direita. O valor e o tipo de uma lista de expressões que pode ser separada por vírgulas é o valor e o tipo da expressão mais à direita da lista. O operador de vírgula é usado mais frequentemente em uma estrutura for. Sua utilidade principal é permitir que o programador use expressões múltiplas para inicialização e/ou incremento.

- **Boa prática de programação**

| |
|---|
| Nas seções de inicialização e incremento de uma estrutura for, coloque apenas expressões que utilizam variáveis de controle. A manipulação de outras variáveis deve aparecer antes do loop (se elas devem ser executadas apenas uma vez como instruções de inicialização) ou no corpo do loop (se elas devem ser executadas uma vez. para cada repetição como as instruções para incrementar ou decrementar). |

As três expressões na estrutura for são opcionais. Se expressão 2 for omitida, a linguagem java presume que a condição é verdadeira, criando assim um loop infinito. A expressão 1 pode ser omitida se a variável de controle for inicializada em outro lugar do programa. A expressão 3 pode ser omitida se o incremento é calculado por instruções no corpo da estrutura for ou se nenhum incremento se faz necessário. A expressão de incremento na estrutura for age como uma instrução independente do java no final do corpo do for. Desta forma, as expressões

```
contador = contador + 1

contador += 1

++contador

contador++
```

são equivalentes na parte incremental da estrutura for. Muitos programadores em java preferem a forma contador++ porque o incremento ocorre depois de o corpo do loop ser executado. Assim, pós-incrementar parece mais natural. Como a variável que está sendo pós-incrementada ou pré-incrementada aqui não aparece na expressão, ambas as formas de incrementar produzem o mesmo efeito. São exigidos os dois ponto-e-vírgula na estrutura for.

- **Erros comum de programação**

  - Usar vírgulas em vez de ponto-e-vírgula em um cabeçalho de uma estrutura for.
  - Colocar um ponto-e-vírgula imediatamente à direita do cabeçalho de uma estrutura for faz com que o corpo dessa estrutura seja uma instrução vazia. Normalmente isto é um erro de lógica.

## A Estrutura For: Notas e Observações

1. A inicialização, a condição de continuação do loop e o incremento podem conter expressões aritméticas. Por exemplo, admita que x = 2 e y = 10, a instrução

```
for (j = x; j <= 4 * x * y; j += y / x)
```

é equivalente à instrução

```
for (j = 2; j <= 80; j += 5)
```

2. O "incremento" pode ser negativo (caso em que na realidade ele é um decremento e o loop faz.. uma contagem regressiva).

3. Se a condição de continuação do loop for inicialmente falsa, a parte do corpo do loop não é realizada. Desta forma, a

execução prossegue com a instrução imediatamente após a estrutura for.

4. Frequentemente, a variável de controle é impressa ou usada em cálculos no corpo de um loop, mas não é exigido que isto aconteça. É comum usar a variável de controle para controlar a repetição sem nunca a mencionar no corpo do loop.

5. A estrutura for é representada em um fluxograma da mesma forma que a estrutura while.

Por exemplo, o fluxograma da instrução for

```
for (contador = 1; contador <= 10; contador++)
printf("%d", contador);
```

é mostrada no exemplo acima. Esse fluxograma torna claro que a inicialização ocorre apenas uma vez e que o incremento acontece depois de o corpo da instrução ter sido executado.

- **Boa prática de programação**

| |
|---|
| Embora o valor da variável de controle possa ser modificado no corpo de um loop for, isto pode levar a erros difíceis de perceber. |

# Loop for simples

O laço for é recomendado para ser usado quando temos um número fixo de iterações necessárias para o loop. Com isso, é possível iterar partes de nosso programa, assim como mostramos anteriormente, quando explicamos o que é o laço For.

Sintaxe, aqui, também não temos surpresa. Como esse é o laço mais comum de ser visto, o seu funcionamento é igual ao que já trazemos:

```
for (inicialização da variável; condição; alteração da variável) {
    [comando]
}
```

Exemplo de uso

O Loop For mais simples também tem usos variados no dia a dia — confira um exemplo de como fazer uma contagem regressiva em uma variável declarada usando o recurso:

```
public class ForSimples{
    public static void main(String[] args) {
    for(int count=10; count >= 1; count--){
        System.out.println(count);
        }
    }
}
```

Aqui, programamos que a alteração da variável — diminuir 1 do valor total — seja realizada a cada vez que seja identificada com o valor de 1, ou maior. Nesse caso, a declaramos com o valor de 10.

# Diferenças entre While e For na linguagem Java?

“**While**” é uma palavra inglesa que significa “**enquanto**”, em português. Dessa forma, queremos que haja (ou não) alguma alteração em determinada variável “**enquanto**” ela é definida por determinado valor, explicando da forma mais simples possível.

Às vezes, não desejamos que nosso programa opere “enquanto” determinada variável apresenta certo valor, mas, sim, que ela se modifique por um número predeterminado de vezes.

Por exemplo, quando temos um array de 100 elementos e queremos contabilizá-los, usaremos um laço For para incrementar +1 em nossa variável que usamos como contador de cada um dos loops. Já o While será usado para que determinado comando seja executado enquanto um requisito está ou não sendo atendido.

## Manipulando Loops For, 4 exemplos práticos

Para ajudar a compreender ainda mais sobre o funcionamento do Loop For em Java, trazemos 4 exemplos práticos sobre como implementar esses recursos nos seus códigos. Confira, a seguir!

## 1. Loop Infinito

```
public class LOOPINF {
    public static void main(String args[]){
            int x;
            for( x = 10 ; ; x-- )
System.out.println(x);
    }
}
```

Neste exemplo, veremos uma série infinita de números iniciando por 10, subtraindo o valor de 1 a cada loop, resultando em uma série sem fim de números negativos depois que o contador passou por 0. Se sua máquina estiver com pouca memória, aconselhamos não testá-lo, ok?

## 2. Atualização e inicialização múltipla de variáveis

Podemos tanto inicializar a execução de uma variável quanto atualizá-la no próprio loop. Por exemplo:

```
public class Exemplo1 {
  public static void main(String args[]) {
    int x1 = 1;
    for(long x2=0, y = 0; x1 < 10 && x2 < 10; x1++,
x2++) {
    System.out.println("O valor de x2 é: " +x2);
    }
    System.out.println("O valor de x1 é: " +x1);
  }
}
```

Repare, no código, que as variáveis são atualizadas e modificadas conforme são executadas pelo loop — o que resulta na seguinte saída:

## 3. Loop For vazio (Empty Loop)

Quando não temos comandos ou alterações nas variáveis em nosso Loop For, chamamos de um loop vazio. Por exemplo:

```
    for(x = 0; x <= 10; x++);
```

Repare que o loop apresentado não contém nenhuma declaração, apenas a alteração da variável x enquanto ela for menor ou igual a zero, adicionando o valor de +1 para x.

Esse tipo de loop é utilizado em aplicações em que precisamos apenas alterar o valor de alguma variável, deixando o restante do programa como está. Também pode ser usado em casos específicos em que temos a necessidade de incluir algum atraso durante o processamento de nosso código.

## 4. Loop com múltiplas declarações e instruções

Podemos escrever múltiplas instruções ou declarar múltiplas variáveis em nossos laços For. É importante saber que cada uma é realizada na sequência em que é executada pelo código.

Veja, a seguir, um exemplo para melhor entendimento:

```
public class nossoMultLoop {
   public static void main(String args[]) {
     int a,sum;
     for(a = 1, sum = 0; a <= 5; ++a) {
     //repare que o loop tem várias instruções
System.out.println("O valor de a agora é: "+a);
          sum = sum + a;
     }
     System.out.println("A soma dos números é "
+sum);
   }
}
```

# Declaração Break

Um operador break em Java é usado de três maneiras. Primeiro, como você já viu, ele termina a sequência de instruções em branches de instruções switch. Segundo, ele pode ser usado para sair de um loop. Terceiro, ele pode ser usado como uma forma "civilizada" do operador de branch incondicional goto. Usando para sair de um loop Ao usar, você pode forçar o término imediato do loop, ignorando a expressão condicional e qualquer código restante no corpo do loop.

Quando uma instrução é encontrada dentro de um loop, a segunda termina e o controle do programa é transferido para a instrução que a segue. Exemplo simples:

```
public class BreakLoop {
  public static void main(String[] args) {
    for (int i = 0; i < 100; i++) {
    if (i == 4) {
    break;
    }
    System.out.println("i: " + i);
    }
    System.out.println("O ciclo está completo.");
  }
}
```

Este programa gera a seguinte saída:

Embora o loop for seja projetado aqui para executar suas instruções de 0 a 99 vezes, a instrução break faz com que ele termine mais cedo quando i é igual a 4. A instrução break pode ser usada com qualquer um dos loops do Java, incluindo loops intencionalmente infinitos. Por exemplo, o programa anterior é mostrado abaixo, codificado com um loop while. A saída deste programa é a mesma que a de seu predecessor.

## Operador continue

Às vezes, é útil iniciar a próxima iteração do loop mais cedo. Ou seja, você precisa continuar a execução do loop, mas parar de processar o restante do código em seu corpo para esta iteração específica. Na verdade, esta é uma goto transição das próximas operações do corpo para o final do bloco do loop. Esta ação é realizada pelo operador continue. Em loops, a instrução while and faz com que o controle seja transferido diretamente para a expressão condicional que controla o loop. No loop, o controle passa primeiro para a parte iterativa do operador e depois para a expressão condicional.

Para todos os três loops, qualquer código intermediário é ignorado. Um programa de exemplo que usa para imprimir dois números em cada linha é fornecido abaixo:

```
public class Continue {

  public static void main(String[] args) {

    for (int i = 0; i < 10; i++) {

    System.out.print(i + " ");

    if (i % 2 == 0) {

    continue;

    }

    System.out.println("");

    }
```

```
  }
}
```

Este código usa a operação %(resto do módulo) para verificar se é i par. Se for, o loop continua sem imprimir o caractere de nova linha.

Assim como na declaração break, você pode definir um rótulo em continue indicando qual loop envolvente continuar. Um programa de exemplo que usa continue para imprimir uma tabela de multiplicação triangular de 0 a 9.

# Operador return

A última declaração de controle é return. Ela é usada para retornar explicitamente de um método, ou seja, ela transfere o controle programático de volta para o programa de chamada. O operador return pertence à categoria de operadores de transição. Embora uma discussão completa sobre ele deva esperar até uma discussão sobre métodos, uma breve visão geral é fornecida aqui return. O operador return pode ser usado em qualquer lugar em um método para saltar de volta para o programa que chamou o método. O operador return encerra imediatamente a execução do método no qual ele reside. Isso é ilustrado pelo exemplo a seguir:

```
public class Return {
  public static void main(String[] args) {
    boolean t = true;
    System.out.println("Antes da declaração de retorno.");
    if (t) {
    return;
    }
    System.out.println("Esta declaração nunca será executada.");
```

```
  }
}
```

Aqui, return retorna ao sistema Java em tempo de execução, já que esse sistema é o que chama o main().

A saída deste programa:

```
Para o operador return.
```

Você pode notar que a declaração final println()não é executada. No momento da execução, return o controle é transferido de volta para o programa de chamada. E a última observação. No programa anterior, o operador if(t) é necessário. Sem ele, o compilador Java lançaria um erro de "código inalcançável" porque saberia que a última declaração println() nunca seria executada. Para evitar esse erro, o operador (t) é usado, if ele engana o compilador para fins desta demonstração.

# Lista de Exercícios Propostos

1. Faça um programa que receba dois números e mostre qual deles é o maior.

2. Leia um número fornecido pelo usuário. Se esse número for positivo, calcule a raiz quadrada do número, Se o número for negativo, mostre uma mensagem dizendo que o número é inválido.

3. Leia um número real. Se o número for positivo imprima a raiz quadrada. Do contrário, imprima o número ao quadrado.

4. Faça um programa que leia um número e, caso ele seja positivo, calcule e mostre:

    O número digitado ao quadrado

    A raiz quadrada do número digitado

5. Faça um programa que receba um número inteiro e verifique se este número é par ou ímpar.

6. Escreva um programa que, dados dois números inteiros, mostre na tela o maior deles, assim como a diferença existente entre ambos.

7. Faça um programa que receba dois números e mostre o maior. Se por acaso, os dois se os números forem iguais, imprima a mensagem Números iguais.

8. Faça um programa que leia 2 notas de um aluno, verifique se as notas são válidas e exiba na tela a média destas notas. Uma nota válida deve ser, obrigatoriamente, um valor entre 0.0 e 10.0, onde caso a nota não possua um valor válido, este fato deve ser informado ao usuário e o programa termina.

9. Leia o salário de um trabalhador e o valor da prestação de um empréstimo. Se a prestação for maior que 20% do salário imprima:

Empréstimo não concedido, caso contrário imprima: Empréstimo concedido.

10. Faça um programa que receba a altura e o sexo de uma pessoa e calcule e mostre seu peso ideal, utilizando as seguintes fórmulas (onde h corresponde á altura):

Homens: (72.7 * h) - 58

Mulheres: (62.1 * h) - 44.7

11. Escreva um programa que leia um número inteiro maior do que zero e devolva, na tela, a soma de todos os seus algarismos. Por exemplo, ao número 251 corresponderá o valor 8 (2 + 5 + 1). Se o número lido não for maior do que zero, o programa terminará com a mensagem “Número invalido”.

12. Ler um número inteiro. Se o número lido for negativo, escreva a mensagem “Número inválido”. Se o número for positivo, calcular o logaritmo desse número.

13. Faça um algoritmo que calcule a média ponderada das notas de 3 provas. A primeira e a segunda prova tem peso 1 e a terceira tem peso 2. Ao final, mostrar a média do aluno e indicar se o aluno foi aprovado ou reprovado. A nota para aprovação deve ser igual ou superior a 60 pontos.

13. Faça uma prova de matemática para crianças que estão aprendendo a somar números inteiros menores do que 100. Escolha números aleatórios entre 1 e 100, e mostre na tela a pergunta: qual é a soma de a + b, onde a e b são os números aleatórios. Peça a resposta. Faça cinco perguntas ao aluno, e mostre para ele as perguntas e as respostas corretas, além de quantas vezes o aluno acertou.

14. Faça um programa que receba três números e mostre-os em ordem crescente.

15. Calcule as raízes da equação de 2o grau. Lembrando que:

- Se DELTA < 0, não existe real. Imprima a mensagem Não existe raiz.
- Se DELTA = 0, existe uma raiz real. Imprima a raiz e a mensagem Raiz única.

- Se DELTA >= 0, imprima as duas raízes reais.

16. Uma empresa vende o mesmo produto para quatro diferentes estados. Cada estado possui uma taxa diferente de imposto sobre o produto (MG 7%; SP 12%; RJ 15%; MS 8%). Faça um programa em que o usuário entre com o valor e o estado destino do produto e o programa retorne o preço final do produto acrescido do imposto do estado em que ele será vendido. Se o estado digitado não for válido, mostrar uma mensagem de erro.

17. Determine se um determinado ano lido é bissexto. Sendo que um ano é bissexto se for divisível por 400 ou se for divisível por 4 e não for divisível por 100. Por exemplo:  1988, 1992, 1996

18. Escreva o menu de opções abaixo. Leia a opção do usuário e execute a operação escolhida. Escreva uma mensagem de erro se a opção for inválida.

Escolha a opção:

1- Soma de 2 números.

2- Diferença entre 2 números (maior pelo menor).

3- Produto entre 2 números.

4- Divisão entre 2 números (o denominador não pode ser zero).

19. Faça um programa que mostre ao usuário um menu com 4 opções de operações matemáticas (as básicas, por exemplo). O usuário escolhe uma das opções e o seu programa então pede dois valores numéricos e realiza a operação, mostrando o resultado e  saindo.

20. Faça um programa para verificar se um determinado número inteiro é divisível por 3 ou 5, mas não simultaneamente pelos dois.

21. Dados três valores, A, B, C, verificar se eles podem ser valores dos lados de um triângulo e, se forem, se é um triângulo escaleno, equilátero ou isóscele, considerando os seguintes conceitos:

O comprimento de cada lado de um triângulo é menor do que a soma dos outros dois lados.

Chama-se equilátero o triângulo que tem três lados iguais.

Denominam-se isósceles o triângulo que tem o comprimento de dois lados iguais. Recebe o nome de escaleno o triângulo que tem os três lados diferentes.

22. Usando switch, escreva um programa que leia um inteiro entre 1 e 7 e imprima o dia da semana correspondente a este número. Isto é, domingo se 1, segunda-feira se 2, e assim por diante.

23. Usando switch, escreva um programa que leia um inteiro entre 1 e 12 e imprima o mês correspondente a este número. Isto é, janeiro se 1, fevereiro se 2, e assim por diante.

24. Faça um Programa que peça dois números e imprima o maior deles.

25. Faça um Programa que peça um valor e mostre na tela se o valor é

positivo ou negativo.

26. Faça um Programa que verifique se uma letra digitada é "F" ou "M".

Conforme a letra escrever: F - Feminino, M - Masculino, Sexo Inválido.

27. Faça um Programa que verifique se uma letra digitada é vogal ou

Consoante.

28. Faça um programa que peça uma nota, entre zero e dez. Mostre uma mensagem caso o valor seja inválido e continue pedindo até que o usuário informe um valor válido.

29. Faça um programa que leia um nome de usuário e a sua senha e não aceite a senha igual ao nome do usuário, mostrando uma mensagem de erro e voltando a pedir as informações.

30. Faça um programa que leia e valide as seguintes informações:

a. Nome: maior que 3 caracteres;

b. Idade: entre 0 e 150;

c. Salário: maior que zero;

d. Sexo: 'f' ou 'm';

e. Estado Civil: 's', 'c', 'v', 'd';

31. Supondo que a população de um país A seja da ordem de 80000 habitantes com uma taxa anual de crescimento de 3% e que a

população de B seja 200000 habitantes com uma taxa de crescimento de 1.5%. Faça um programa que calcule e escreva o número de anos necessários para que a população do país A ultrapasse ou iguale a população do país B, mantidas as taxas de crescimento.

32. Altere o programa anterior permitindo ao usuário informar as populações e as taxas de crescimento iniciais. Valide a entrada e permita repetir a operação.

33. Faça um programa que imprima na tela os números de 1 a 20, um abaixo do outro. Depois modifique o programa para que ele mostre os números um ao lado do outro.

34. Faça um programa que leia 5 números e informe o maior número.

35. Faça um programa que leia 5 números e informe a soma e a média dos números.

36. Faça um programa que imprima na tela apenas os números ímpares entre 1 e 50.

37. Faça um programa que receba dois números inteiros e gere os números inteiros que estão no intervalo compreendido por eles.

38. Altere o programa anterior para mostrar no final a soma dos números.

39. Faça um programa que peça dois números, base e expoente, calcule e mostre o primeiro número elevado ao segundo número. Não utilize a função de potência da linguagem.

40. Faça um programa que peça 10 números inteiros, calcule e mostre a quantidade de números pares e a quantidade de números ímpares.

41. A série de Fibonacci é formada pela sequência 1,1,2,3,5,8,13,21,34,55,... Faça um programa capaz de gerar a série até o n−ésimo termo.

42. A série de Fibonacci é formada pela sequência 0,1,1,2,3,5,8,13,21,34,55,... Faça um programa que gere a série até que o valor seja maior que 500.

43. Faça um programa que calcule o fatorial de um número inteiro fornecido pelo usuário. Ex.: 5!=5.4.3.2.1=120

44. Faça um programa que, dado um conjunto de N números, determine o menor valor, o maior valor e a soma dos valores.

45. Altere o programa anterior para que ele aceite apenas números entre 0 e 1000.

46. Altere o programa de cálculo do fatorial, permitindo ao usuário calcular o fatorial várias vezes e limitando o fatorial a números inteiros positivos e menores que 16.

47. Faça um programa que peça um número inteiro e determine se ele é ou não um número primo. Um número primo é aquele que é divisível somente por ele mesmo e por 1.

48. Altere o programa de cálculo dos números primos, informando, caso o número não seja primo, por qual número ele é divisível.

49. Faça um programa que mostre todos os primos entre 1 e N sendo N um número inteiro fornecido pelo usuário. O programa deverá mostrar também o número de divisões que ele executou para encontrar os números primos. Serão avaliados o funcionamento, o estilo e o número de testes (divisões) executados.

50. Faça um programa que calcule e mostre a média aritmética de N notas.

51. Faça um programa que peça para n pessoas a sua idade, ao final o programa deverá verificar se a média de idade da turma varia entre 0 e 25,26 e 60 e maior que 60; e então, dizer se a turma é jovem, adulta ou idosa, conforme a média calculada.

52. Numa eleição existem três candidatos. Faça um programa que peça o número total de eleitores. Peça para cada eleitor votar e ao final mostrar o número de votos de cada candidato.

53. Faça um programa que calcule o número médio de alunos por turma. Para isto, peça a quantidade de turmas e a quantidade de alunos para cada turma. As turmas não podem ter mais de 40 alunos.

54. Faça um programa que calcule o valor total investido por um colecionador em sua coleção de CDs e o valor médio gasto em cada um deles. O usuário deverá informar a quantidade de CDs e o valor para cada um.

55. O Departamento Estadual de Meteorologia lhe contratou para desenvolver um programa que leia as um conjunto indeterminado de temperaturas, e informe ao final a menor e a maior temperaturas informadas, bem como a média das temperaturas.

56. Os números primos possuem várias aplicações dentro da Computação, por exemplo na criptografia. Um número primo é aquele que é divisível apenas por um e por ele mesmo. Faça um programa que peça um número inteiro e determine se ele é ou não um número primo.

57. Encontrar números primos é uma tarefa difícil. Faça um programa que gera uma lista dos números primos existentes entre 1 e um número inteiro informado pelo usuário.

58. Faça um programa que leia dez conjuntos de dois valores, o primeiro representando o número do aluno e o segundo representando a sua altura em centímetros. Encontre o aluno mais alto e o mais baixo. Mostre o número do aluno mais alto e o número do aluno mais baixo, junto com suas alturas.

59. Foi feita uma estatística em cinco cidades brasileiras para coletar dados sobre acidentes de trânsito. Foram obtidos os seguintes dados: Código da cidade;

a. Número de veículos de passeio (em 1999);

b. Número de acidentes de trânsito com vítimas (em 1999). Deseja-se saber:

c. Qual o maior e menor índice de acidentes de trânsito e a que cidade pertence;

d. Qual a média de veículos nas cinco cidades juntas;

e. Qual a média de acidentes de trânsito nas cidades com menos de 2.000 veículos de passeio.

60. Faça um programa que leia uma quantidade indeterminada de números positivos e conte quantos deles estão nos seguintes intervalos: [0-25], [26-50], [51-75] e [76-100]. A entrada de dados deverá terminar quando for lido um número negativo.

61. Sendo H= 1 + 1/2 + 1/3 + 1/4 + ... + 1/N, Faça um programa que calcule o valor de H com N termos.

# MÓDULO 8: STRINGS

**Programação é um processo sem fim de tentativa e erro, de tentar colocar o comando certo no lugar certo.**

Reshma Saujani

- PROCESSAMENTO DE STRINGS
- MÉTODOS
- EXPRESSÕES REGULARES
- LENGTH, GET CHARS
- COMPARANDO STRINGS
- OPERADOR IGUALDADE VS MÉTODO EQUALS
- VALUE OF
- STRINGBUILDER

# Objetivos

Ao final desse capítulo o leitor deverá ser capaz de:

- Manipular strings e seus métodos.
- Saber a diferença entre o operador == e o método equals.
- Saberá a usar diversos métodos de strings.
- Poder comparar strings.
- Poderá saber usar o melhor método de acordo com a necessidade.

# STRINGS E SUAS MANIPULAÇÕES EM JAVA

Em Java, a **String** é um objeto que representa uma sequência de caracteres. Java fornece uma API robusta e flexível para lidar com strings, permitindo várias operações, como concatenação, comparação e manipulação. Neste capítulo, abordaremos o conceito de String Java em detalhes.

## O que são Strings em Java?

Strings são o tipo de objeto que pode armazenar o caractere de valores e, em Java, cada caractere é armazenado em 16 bits, ou seja, usando codificação UTF de 16 bits. Uma string age da mesma forma que um array de caracteres em Java, mais pra frente vamos falar sobre arrays.

Strings são amplamente utilizadas na programação Java e representam uma sequência de caracteres. Na linguagem Java, as strings são tratadas como objetos. A plataforma Java fornece a classe String para criar e manipular strings.

## Criando Strings

Para o compilador, qualquer texto entre aspas duplas é uma String.
Por esse motivo, a criação de um objeto desse tipo não requer a utilização do operador new. Assim, uma String é criada de forma semelhante a um tipo primitivo, utilizando-se a sintaxe [tipo] [nome] = [valor], apesar de se tratar de um tipo por referência, A maneira mais direta de criar uma string é escrever:

Exemplo:

```
//criação do String
String str = “Hello”;
```

```
int inteiro = 4;
String novo = str + "valor do inteiro:" + inteiro;
//Concatenação de strings
```

## Formas de criar uma string

Existem duas maneiras de se criar uma string em Java, de forma: Literal de sequência de caracteres, e de forma a usar a palavra-chave **new**

1) Literal de string (cadeia de caracteres)
2) Usando a palavra-chave new

### 1. Literal de string (cadeia de caracteres)

O literal String de Java é criado utilizando aspas duplas.

Por exemplo:

```
String s = "bem-vindo";
```

Sempre que cria um literal de cadeia, a JVM verifica primeiro o "conjunto de constantes de cadeia". Se a cadeia já existir no conjunto, é devolvida uma referência à instância do conjunto. Se a string não existir no pool, uma nova instância de string é criada e colocada no pool.

Por exemplo:

```
String s1 = "Bem-vindo";
String s2 = "Bem-vindo"; //Não cria uma nova instância
```

No exemplo acima, apenas será criado um objeto. Em primeiro lugar, a JVM não encontrará qualquer objeto de cadeia de caracteres com o valor "Bem-vindo" no conjunto de constantes de cadeia de caracteres, razão pela qual criará um novo objeto. Depois de encontrar a cadeia de

caracteres com o valor “Bem-vindo” no conjunto, não criará um novo objeto, mas devolverá a referência à mesma instância.

Mais um Exemplo:

```
String demonStrating = “eu gosto de java”;
```

2. Usando a palavra-chave new

```
String s = new String(“Bem-vindo”);
```

Nesse caso, a JVM criará um novo objeto string na memória heap normal (não pool) e o literal “Bem-vindo” será colocado no pool de constantes de string.

A variável se refere ao objeto no heap (não pool) No exemplo dado, apenas um objeto será criado. Primeiramente, a JVM não encontrará nenhum objeto string com o valor “Bem-vindo” no pool de constantes string, então ela criará um novo objeto. Depois disso, ela encontrará a string com o valor “Bem-vindo” no pool, ela não criará um novo objeto, mas retornará à referência para a mesma instância.

Exemplo:

```
String demoString = new String (“Mais uma string”);
```

Exemplo usando a forma literal: StringDemo.java

```
public class StringDemo {
    public static void main(String[] args) {
        String saudacao = “olá mundo!”;
    }
}
```

Sempre que o compilador encontra uma string literal no código, ele cria um objeto String com o valor correspondente, neste caso,

Exemplo usando a palavra-chave new: StringDemo1.java

```
public class StringDemo1 {
public static void main(String[] args) {

char[] helloArray = {'o','l','á','!'};
String helloString = new String(helloArray);
System.out.println(helloString);
    }
}
```

## Imutabilidade

Strings em Java são imutáveis. Isso significa que uma vez criada, uma String não pode ser alterada. Por exemplo, ao fazer uma operação como str += "abc", você está, na verdade, criando um novo objeto String ao invés de modificar o objeto original.

Essa característica de imutabilidade traz benefícios como segurança (já que o valor não pode ser alterado de forma inesperada), além de otimizar o uso de memória através do String Pool.

## Comprimento da String(length)

O método String length retorna o comprimento da string em Java,
ou seja, o tamanho da string. Aqui veremos como encontrar o comprimento de uma string usando o método String length.

A classe String do Java compreende muitos métodos para executar várias operações em strings, como compare, concat, equals, split, length, replace, compareTo, substring, etc. Destes métodos, vamos nos concentrar no método length da classe String do Java.

O comprimento ou tamanho da string significa o número total de caracteres presentes nela.
Em String, todos os elementos são armazenados na forma de caracteres, ou seja, “1”, ” “, “_”, etc. todos são considerados caracteres.
O método length() conta o número total de caracteres em uma String.

Exemplo:

```
public class StringDemo {
   public static void main(String args[]) {

      String frase = "Eu vi o ceu!";
    int comprimento = frase.length();
    System.out.println("O comprimento da string
e: " + comprimento);
   }
}
```

## Concatenando Strings

Existem duas formas de unir duas ou mais sequências de caracteres.

A mais comum dentre elas é utilizando o operador de adição, como exemplo concatenação de Strings:

```
String nome = "Mauro ";
String sobrenome = "Talino";
String nomeCompleto = nome + sobrenome;
```

### Método String concat

O método String concat concatena (acrescenta) uma string ao final de outra string. Ele retorna a string combinada, ele é usado para concatenação de strings, Ele retorna NullPointerException se qualquer uma das strings for Null.

```
// Declarar a variável string original.
String original = "angola";

// Declara outra variável string com o nome
//e inicializa-a com uma string vazia.
String invertido = "";

// Iterar por cada carácter da string "original"
do último índice para o primeiro.
for (int i = original.length() - 1; i >= 0; i--) {
// Extrai o caractere atual no índice "i" da string
"original".
    char currentChar = original.charAt(i);

    // Converte o caractere para um objeto String
    //utilizando o método "Character.toString".
    String charAsString =
Character.toString(currentChar);


    // Concatenar o carácter convertido String
    //convertido ao final da string "invertida".
    invertido = invertido.concat(charAsString);
}

// Imprime as strings Original e Invertido.
System.out.println("string original: "+ original);
System.out.println("string invertida: "+ invertido);
```

# Onde fica o armazenamento

É útil visualizar alguns aspectos de como as coisas estão dispostas enquanto o programa está a ser executado, em particular como a memória é organizada. Existem cinco locais diferentes para armazenar dados:

- 1. Registos. Este é o armazenamento mais rápido porque existe num local diferente do outro tipo de armazenamento: dentro do processador. No entanto, o número de registos é severamente limitado, pelo que os registos são atribuídos à medida que são necessários. O utilizador não tem controlo direto, nem vê qualquer evidência nos seus programas de que os registos existem (C & C++, por outro lado, permitem sugerir por outro lado, permitem-lhe sugerir ao compilador a atribuição de registos).

- 2. A stack. Esta vive na área geral da memória de acesso aleatório (RAM), mas tem suporte direto do processador através do seu ponteiro de stack. O ponteiro da stack é movido para baixo para criar nova memória e para cima para libertar essa memória. Esta é uma forma Esta é uma forma extremamente rápida e eficiente de atribuir armazenamento, perdendo apenas para os registos. O sistema Java deve saber, enquanto está a criar o programa, o tempo de vida exato de todos os itens que estão armazenados na stack. Esta restrição impõe limites à flexibilidade dos seus programas, portanto, embora exista algum armazenamento Java na stack - em particular, referências a objetos - os objetos Java em si não são colocados na stack.

- 3. O heap. Este é um pool de memória de uso geral (também na área de RAM) onde todos os objetos Java. O bom do heap é que, ao contrário da stack, o compilador não precisa não precisa de saber quanto tempo esse armazenamento deve permanecer na stack. Assim, há uma grande flexibilidade no

uso do armazenamento no heap. Sempre que você precisar de um objeto, você simplesmente escreve o código para criá-lo usando new, e o armazenamento é alocado no heap quando esse código é executado. É claro que há um preço a pagar por esta flexibilidade: pode levar mais tempo para alocar e limpar o armazenamento da heap do que o armazenamento da stack (se você fosse possível criar objectos na stack em Java, como é possível em C++).

- 4. Armazenamento constante. Os valores constantes são frequentemente colocados diretamente no código do programa, o que é seguro, uma vez que nunca podem mudar. Por vezes, as constantes são isoladas por para que possam ser opcionalmente colocadas em memória só de leitura (ROM), em sistemas incorporados.

- 5. Armazenamento não-RAM. Se os dados estiverem completamente fora de um programa, podem existir enquanto o programa não está a ser executado, fora do controlo do programa. Os dois principais exemplos disso são os objetos de fluxo, nos quais os objetos são transformados em fluxos de bytes, geralmente para serem enviados para outra máquina, e os objetos persistentes, em que os objetos são colocados no disco para que eles mantenham seu estado mesmo quando o programa é terminado. O truque com estes tipos de armazenamento é transformar os objetos em algo que possa existir no outro meio e, no entanto, possa ser ressuscitado num objeto regular baseado em RAM quando necessário. O Java oferece suporte para persistência leve, e mecanismos como o JDBC e o Hibernate fornecem um suporte mais sofisticado para armazenamento e recuperação de informações de objetos em bases de dados.

## Criando Strings Formatadas(format)

Você pode usar os métodos printf e format para criar strings com números ou textos formatados. O método estático format da classe String permite reutilizar a string formatada em vez de apenas imprimi-la diretamente.

Exemplo

Em vez de usar:

```
System.out.printf("O valor do float e %f, o inteiro e %d, e a string é %s", floatVar, intVar, stringVar);
```

Você pode usar:

```
String fs;
fs = String.format("O valor do float e %f, o inteiro e %d, e a string e %s", floatVar, intVar, stringVar);
System.out.println(fs);
```

```
public class Main {
    public static void main(String[] args) {
  // Cadeia de entrada personalizada a ser formatada
  String str = "Motorola";

  // Concatenação de duas cadeias de caracteres
   String s = String.format("Nome da Empresa %s", str);

   // O resultado é dado até 8 casas decimais
      String str2 = String.format("Minha resposta %.8f", 47.65734);
```

```
// Aqui a resposta é suposto ser %15.8f” e
// “47.65734000” tem 15 espaços
String str3 = String.format("Minha resposta %15.8f",
47.65734);

 // Imprimir e apresentar cadeias de caracteres
    System.out.println(s);
    System.out.println(str2);
    System.out.println(str3);
    }
}
```

# Principais Métodos de Manipulação de Strings

A classe String fornece muitos métodos úteis para a manipulação de strings, Abaixo estão alguns dos mais usados, acompanhados de exemplos.

## Acessando Caracteres Individualmente (charAt)

O método String charAt() retorna o caractere no índice especificado em uma string. O índice do primeiro caractere em uma string é 0, o do segundo caractere é 1, e assim por diante. O valor do índice deve estar entre 0 e length() – 1.

Se o valor do índice for maior ou igual ao comprimento da string ou a um número negativo, ele retornará stringIndexOutOfBoundsException

```
public class Main {
    public static void main(String[] args) {
    // Definir uma cadeia de caracteres
    String s = "Java String charAt() example";

    // Recuperar e imprimir o carácter no índice 8
    char ch = s.charAt(8);
    System.out.println(ch);

    // Recuperar e imprimir o carácter no índice 24
    ch = s.charAt(24);
    System.out.println(ch);
    }
}
```

# Comparando Strings (equals e equalsIgnoreCase)

Em Java, você não deve usar o operador `==` para comparar strings, pois isso compara referências de objetos e não o conteúdo.

No Java, para comparar Strings com precisão, usamos os métodos equals e  equalsIgnoreCase, e entender como cada um funciona é essencial para evitar erros e garantir que a comparação seja feita conforme desejado.

Vamos ver como e quando utilizar cada um:

## Equals

O método equals é utilizado para comparar o conteúdo de duas Strings levando em conta letras maiusculas e minúsculas.

Retorna true se ambas as Strings têm o mesmo conteúdo e o mesmo tamanho.

Exemplo de uso:

```
String str1 = "Java";
String str2 = "java";

System.out.println(str1.equals(str2));
// false, pois "Java" e "java" têm diferenças de maiusculas e minúsculas

System.out.println(str1.equals("Java"));
// true, pois ambos os conteúdos são idênticos
```

## EqualsIgnoreCase

O método equalsIgnoreCase também compara o conteúdo das Strings, mas ignora as diferenças de letras maiusculas e minúsculas.
Esse método é útil quando a sensibilidade de caso (maiusculas e minúsculas) não importa, por exemplo, ao comparar nomes de usuários ou e-mails.

Exemplo de uso:

```
 String str1 = "Java";
 String str2 = "java";
System.out.println(str1.equalsIgnoreCase(str2));
// true, ignora as diferenças de maiusculas e minúsculas
```

## Diferenças Práticas entre  equals e equalsIgnoreCase

**equals**: Sensível ao caso. Útil quando é necessário que o conteúdo seja exatamente igual, incluindo o formato das letras.

**equalsIgnoreCase**: Ignora o caso. Útil em situações onde o formato (maiusculas/minúsculas) não é relevante para a comparação.

Exemplo Comparativo

Imagine que estamos desenvolvendo um sistema de login onde o nome de usuário não distingue letras maiusculas de minúsculas, mas a senha deve ser exata:

```
public class Main {
    public static void main(String[] args) {
    String username = "User123";
    String password = "MySecretPass";

    // Entradas do usuário
    String inputUsername = "user123";
    String inputPassword = "mysecretpass";

    // Comparação de nome de usuário
    boolean isUsernameValid =
username.equalsIgnoreCase(inputUsername);
// true, pois ignora o caso

    // Comparação de senha
    boolean isPasswordValid =
password.equals(inputPassword);
// false, pois é sensível ao caso

    if (isUsernameValid && isPasswordValid) {
        System.out.println("Login bem-sucedido!");
        } else {
        System.out.println("Nome de usuário ou senha
incorretos.");
    }
    }
}
```

## Observações de Performance e Boas Práticas

Ambos os métodos equals e equalsIgnoreCase são otimizados para comparar strings, mas em casos onde você sabe que a String é curta e tem um padrão consistente (como um código de produto ou sigla), usar equals pode ser um pouco mais eficiente devido à ausência de manipulação de caso.

Evite usar `==` para comparar Strings. Esse operador compara a referência dos objetos (endereços na memória), o que geralmente não é o desejado ao comparar conteúdos de Strings.

Esses métodos tornam a manipulação de Strings em Java mais versátil e precisa, ajudando a lidar com diferentes necessidades de comparação em programas e sistemas.

## Método contains

O método contains da classe String, do português contêm, pesquisa a sequência de caracteres desta string.

Ele retorna true se a sequência de valores char for encontrada nesta string, caso contrário, retorna false.

Exemplo: Main02.java

```
public class Main02 {
    public static void main(String[] args) {
    String frase = "Java é divertido!";
    boolean contem = frase.contains("Java");
    System.out.println(contem);
    }
}
```

Exemplo: Main03.java

```
String str = "Para aprender Java visita:
JavaPontoOrg.com";

if (str.contains("JavaPontoOrg.com")) {
System.out.println("Esta JavaPontoOrg.com");
} else {
  System.out.println("Resultado nao encontrado");
}
```

## Método Substrings (substring)

Em Java, uma Substring é uma parte de uma String ou um subconjunto da String. Há duas variantes do método substring.

Aqui, exploraremos essas duas variantes do método Java String substring(), sua sintaxe, casos de uso, exemplos e possíveis aplicações em cenários do mundo real.

O método substring pode ser utilizado para fazer uma extração de prefixos ou sufixos. Por exemplo, podemos ter uma lista de nomes e é necessário filtrar os nomes cujo apelido é “singh”.

O programa seguinte mostra o mesmo.

```
public class Main {
    public static void main(String[] args) {
    String str[] =
            {
                "Praveen Kumar",
                "Yuvraj Singh",
                "Harbhajan Singh",
                "Gurjit Singh",
                "Virat Kohli",
                "Rohit Sharma",
                "Sandeep Singh",
                "Milkha Singh"
            };

    String surName = "Singh";
    int surNameSize = surName.length();

    int size = str.length;

    for(int j = 0; j < size; j++) {
        int length = str[j].length();

// extracting the surname
```

```
String subStr = str[j].substring(length -
surNameSize);
// checks whether the surname is equal to "Singh" or
not
         if(subStr.equals(surName)){
              System.out.println(str[j]);
         }
        }
    }
}
```

## Substituindo Caracteres ou Sequências (replace)

O método replace retorna um novo objeto contendo a string original com um trecho especificado substituído por outra expressão indicada.
Esse método deixa a string original inalterada.

A versão sobrecarregada do método replace permite substituir substrings em vez de caracteres individuais.
Exemplo: Main01.java

```
public class Main01 {
    public static void main(String[] args) {

    String nome = "mesquita";
    String nomeAlterado = nome.replace('e', 'o');

    System.out.println(nomeAlterado);

    }
}
```

Saída:

```
Mosquita
```

```
// Inicializando a String
String Str = new String("Bem-vindo ao conceitos
iniciais");

// Utilizar replace para substituir caracteres
System.out.print("Depois de substituir todos os o
por T: ");
System.out.println(Str.replace('o', 'T'));
```

```
// Utilizar replace para substituir caracteres
System.out.print("Depois de substituir todos os e
por D: ");
System.out.println(Str.replace('e', 'D'));
```

## Dividindo uma String (split)

O método split divide uma string em partes, com base em um delimitador especificado.  Após dividir contra a expressão regular dada, esse método retorna um array de strings.

```
String str = "apple,orange,banana";
String[] fruits = str.split(",");
System.out.println(Arrays.toString(fruits));
// Saída: [apple, orange, banana]
```

No exemplo acima, o nosso delimitador são os espaços, ou seja, divisão ocorre depois sempre que existir um espaço.

## Verificando Início e Fim de Strings (startsWith, endsWith)

Os métodos startsWith e endsWith aceitam uma string e um número inteiro como argumentos, retornando um valor booleano que indica se a string inicia ou termina, respectivamente, com o texto informado a partir da posição dada.

Você pode usar os métodos startsWith e endsWith para verificar se uma string começa ou termina com uma determinada sequência de caracteres.

```
 // Declarar e inicializar uma String
 String Str = new String("Sandeep Jain");

 // Testando o prefixo usando o método startsWith()
 System.out.print("Check Whether It Starts With
Letter 'S': ");

 // Impressão do valor booleano correspondente
 System.out.println(Str.startsWith("S"));
```

Este exemplo resume: startsWith, endsWith

```
String[] nomes = {"iniciado", "iniciando",
"finalizado", "finalizando"};

for (String recebeNomes : nomes) {
    if (recebeNomes.startsWith("in"))
    System.out.printf("\"%s\"inicia com \"in\" \n",
recebeNomes);
}

System.out.println();

for (String recebeNomes : nomes) {
    if (recebeNomes.startsWith("ici", 2))
    System.out.printf("\"%s\" inicia com \"ici\" na
posição 2 \n", recebeNomes);
}

System.out.println();

for (String recebeNomes : nomes) {
    if (recebeNomes.endsWith("ado"))
    System.out.printf("\"%s\" termina com \"ado\"
\n", recebeNomes);
}
```

## Procurando Posições de Caracteres ou Substrings (indexOf, lastIndexOf)

Permitem a localização de caracteres e substrings especificadas em strings.

## IndexOf

Localiza a primeira ocorrência de um caractere em uma string.
Se o método localizar o caractere, é retornado o índice do caractere na String, caso contrário retorna.

Existem duas versões do indexOf que procuram caracteres em uma String.

1ª versão – aceita um inteiro que é conhecido como o número do índice na String.
2ª versão – aceita dois argumentos inteiros – o caractere e o índice inicial em que a pesquisa da String deve iniciar.

```
String letras = "abcadefghijklmcopqrsdeftuvz";

//IndexOf PARA LOCALIZAR UM CARACTERE EM UM STRING
System.out.printf("Último 'c' está localizado no index %d\n", letras.indexOf('c'));

System.out.printf("Último 'a' está localizado no index %d \n", letras.indexOf('a', 1));


//-1 NÃO EXISTE
System.out.printf("'$' está localizado no index %d\n\n", letras.indexOf('$'));
```

## LastIndexOf

Localiza a última ocorrência de um caractere em uma string.
O método procura do fim da string em direção ao começo, se encontrar o caractere é retornado o seu índice na string, caso contrário retorna -1.
Existem duas versões do lastIndexOf que pesquisam por caracteres em uma string.

1ª versão – utiliza um inteiro do caractere.
2ª versão – aceita 2 argumentos – um inteiro de caractere e o índice a partir do qual iniciar a pesquisa de trás para frente.

```
String letras = "abcadefghijklmcopqrsdeftuvz";

//lastIndexOf PARA LOCALIZAR UM CARACTERE EM UMA
STRING
System.out.printf("Último 'c' está localizado no
index %d\n", letras.lastIndexOf('c'));
System.out.printf("Último 'a' está localizado no
index %d\n", letras.lastIndexOf('a', 5));
System.out.printf("Último '$' está localizado no
index %d\n", letras.lastIndexOf('$'));

//indexOf PARA LOCALIZAR UMA SUBSTRING EM UMA STRING
System.out.printf("\"def\" está localizado no index
%d\n", letras.indexOf("def"));

//2 argumento string e outro o ponto inicial que
começará a pesquisa
System.out.printf("\"def\" está localizado no index
%d\n", letras.indexOf("def", 7));
System.out.printf("\"hello\" está localizado no
index %d\n\n", letras.indexOf("hello"));
```

## Convertendo Outros Tipos para String (valueOf)

valueOf é um método estático da classe String, que não precisa de uma instância para ser invocado.
Ele converte um tipo primitivo em um objeto do tipo String.

```
double numero = 102939939.939;
boolean booleano = true;

System.out.println("Retorna Valor: " +
String.valueOf(numero));

System.out.println("Retorna Valor: " +
String.valueOf(booleano));
```

## Método compareTo

Strings em Java são objetos que são suportados internamente por um array somente, o que significa alocação contígua de memória para caracteres. Observe que strings são imutáveis em Java, o que significa que, uma vez que criamos um objeto String e atribuímos alguns valores a ele, não podemos alterar o conteúdo. No entanto, podemos criar outro objeto String com as modificações que queremos.

A classe String do Java compreende muitos métodos para executar várias operações em strings e vamos nos concentrar no método compareTo.

O método compareTo compara a string fornecida com a string atual lexicograficamente. Ele retorna um número positivo, um número negativo ou 0. Ele compara strings com base no valor Unicode de cada caractere nas strings.

```
String str1 = "Nerds";
String str2 = "Nerdos";

int comparacao = str1.compareTo(str2);

if (comparacao < 0) {
    System.out.println(str1 + " vem antes de ‘+ str2
+’ lexicograficamente.");
} else if (comparacao > 0) {
    System.out.println(str1 + " vem depois de ‘+
str2 +’ lexicograficamente.");
} else {
    System.out.println(str1 + " e ‘+ str2 +’ são
lexicograficamente iguais.");
}
```

## Método compareToIgnoreCase

Compara lexicograficamente duas strings, ignorando as diferenças entre maiusculas e minúsculas.

```
public class Main01 {
    public static void main(String[] args) {
    String str1 = "apple";
    String str2 = "Apple";

System.out.println(str1.compareToIgnoreCase(str2));
    }
}
```

## Método contentEquals

Semelhante ao método equals, o método contentEquals também é usado para comparar o conteúdo da String. No entanto, diferentemente do método equals, contentEquals recebe qualquer implementação da interface CharSequence como um argumento.
Isso significa que String, StringBuffer, StringBuilder, CharBuffer ou Segment podem ser comparados.

Portanto, o método contentEquals se preocupa apenas com o conteúdo da string. Se o argumento for um objeto String, o método equals é chamado para comparação. Por outro lado, se uma sequência de caracteres genérica for fornecida, o método compara caracteres individuais em posições semelhantes.

O método retorna true se a sequência de caracteres no argumento fornecido corresponder à String original. Ao contrário do método equals, se um argumento nulo for passado para o método contentEquals, ele lançará uma NullPointerException.

```
8        String str = "Hello";
9        StringBuilder sb = new StringBuilder("Hello");
10       System.out.println(str.contentEquals(sb));  // saida: true
```

Retorna true se e somente se esta String representar a mesma sequência de caracteres que o StringBuffer especificado.

## Observação: Boas Práticas

Devemos levar em conta, que usar contentEquals quando estamos preocupados apenas com o conteúdo do objeto. Por outro lado, às vezes pode ser importante verificar o tipo do objeto.

Nesse caso, devemos usar o método equals que nos dá condições de verificação mais rigorosas.

## Método copyValueOf

O método copyValueOf converte um array de caracteres em uma String com o mesmo conteúdo. Este método é equivalente a valueOf(char[]).

O deslocamento representa o índice do primeiro elemento a partir do qual a cópia será iniciada, e a contagem representa o número de elementos a serem copiados.

Retorna uma String que representa a sequência de caracteres na matriz especificada.

```
8       char[] data = {'J', 'a', 'v', 'a'};
9       String str = String.copyValueOf(data);
10      System.out.println(str);  // saida: Java
```

## Método matches

O método matches da classe Matcher é usado para obter o resultado se esse padrão corresponde ou não a esse matcher. Ele retorna um valor booleano mostrando o mesmo.

Informa se essa string corresponde ou não à expressão regular fornecida.

```
8       String str = "abc123";
9       System.out.println(str.matches( regex: "\\w+\\d+"));
10      // saida: true
```

## Método regionMatches

boolean regionMatches(int toffset, String other, int ooffset, int len)
Essa versão compara uma parte da string que chama o método (this) com uma parte de outra string (other).

Os parâmetros funcionam da seguinte forma:

toffset: Posição inicial na string this a partir da qual começa a comparação.

other: A outra string com a qual será feita a comparação.

ooffset: Posição inicial na string other a partir da qual começa a comparação.

len: Número de caracteres a serem comparados.

Testa se duas regiões de string são iguais.

```
public class Main {
  public static void main(String[] args) {
      String str1 = "HelloWorld";
      String str2 = "WorldHello";

      boolean result = str1.regionMatches(5,
str2, 0, 5);
      System.out.println(result); // true
  }
}
```

### Quando usar regionMatches

O método regionMatches é útil quando você quer comparar substrings diretamente sem extraí-las antes, e também em situações onde o desempenho é importante, já que evita a criação de objetos temporários.

## Método replaceAll

O método replaceAll nos permite procurar padrões dentro de uma String e substituí-los por um valor desejado. É mais do que uma simples ferramenta de busca e substituição, pois utiliza expressões regulares (regex) para identificar padrões, tornando-o altamente flexível para uma variedade de casos de uso.
Substitui cada substring da string que corresponde à expressão regular fornecida pela substituição fornecida.

```
7
8       String str = "123abc456abc";
9       System.out.println(str.replaceAll( regex: "abc",  replacement: "XYZ"));
10      // saida: 123XYZ456XYZ
```

## Replace VS ReplaceAll

A classe String fornece replace e replaceAll. Os dois métodos parecem similares e podem produzir os mesmos resultados algumas vezes:

```
String input = "hello.java.hello.world";
String replaceResult = input.replace("hello", "hi");
assertEquals("hi.java.hi.world", replaceResult);

String replaceAllResult = input.replaceAll("hello",
"hi");
assertEquals("hi.java.hi.world", replaceAllResult);
```

Neste exemplo, tanto replace(“hello”, “hi”) quanto replaceAll(“hello”, “hi”) substituem todos os “ hello “s em input por “ hi “. Então, alguns de nós podemos perguntar, qual é a diferença entre eles?

A principal diferença entre replace e replaceAll é que replace sempre executa a substituição literal de String. Em contraste, replaceAll usa regex para corresponder a padrões, tornando-se uma ferramenta mais avançada para manipulação de strings.

Os dois métodos produziram o mesmo resultado no exemplo acima, pois os caracteres no padrão regex “ hello ” não têm significados especiais. Agora, digamos que queremos substituir todos os pontos “ . ” por dois pontos “ : “. Se ainda passarmos os mesmos parâmetros para replace e replaceAll desta vez, eles produzirão resultados diferentes

## Método subSequence

O método subSequence obtém uma parte de uma String dado o índice inicial e o comprimento do resultado. O método SubSequence se comporta da mesma forma que substring.

A única diferença é que ele retorna um CharSequence em vez de uma String.

```
8           String str = "Programming";
9           System.out.println(str.subSequence(0, 4));
10          // saida: Prog
11
```

## Método toLowerCase

O método toLowerCase é usado e operado sobre uma string onde queremos converter todas as letras para minúsculas em uma string.

Converte todos os caracteres desta String para letras minúsculas usando as regras da localidade padrão.

```
7
8           String str = "JAVA";
9           System.out.println(str.toLowerCase());  // Output: java
0
```

## Segundo Exemplo do método toLowerCase(Locale locale)

Este método nos permite passar locale também para vários idiomas. Vamos ver um exemplo abaixo onde estamos obtendo string em inglês e turco.

```
import java.util.Locale;

public class Main {
 public static void main(String[] args) {
     String s = "JAVA IS LANG HELLO stRIng";
     String eng = s.toLowerCase(Locale.ENGLISH);
     System.out.println(eng);
     String turkish =
s.toLowerCase(Locale.forLanguageTag("tr"));
     System.out.println(turkish);
     }
}
```

## Método toUpperCase

O método toUpperCase da classe String converteu todos os caracteres da string em uma letra maiuscula. Há duas variantes do método toUpperCase. A principal coisa a ser levada em consideração é que o método toUpperCase funciona da mesma forma que o método UpperCase(Locale.getDefault()), pois internamente o local padrão é usado.

Converte todos os caracteres desta String em maiusculas usando as regras do Locale fornecido.

```
public class Main {
 public static void main(String[] args) {
     String s1="hello string";
     String s1upper=s1.toUpperCase();
     System.out.println(s1upper);
 }
}
```

## Método trim

O método trim remove qualquer espaço em branco no início e no fim de uma String. Se a String contiver apenas espaços, o método retornará uma String vazia.

Retorna uma cópia da string, com espaços em branco à esquerda e à direita omitidos.

```
8          String str = "  Hello World  ";
9          System.out.println(str.trim());
10         // saida: "Hello World"
```

## Manipulando Strings Mutáveis com StringBuilder e StringBuffer

Se precisar modificar uma string várias vezes, é mais eficiente usar StringBuilder ou StringBuffer.
Diferente de String, essas classes criam objetos mutáveis, permitindo a alteração direta do conteúdo sem a criação de novos objetos.

## Usando StringBuilder

```
public class Main {
    public static void main(String[] args) {
    StringBuilder sb = new StringBuilder("Java");

    sb.append(" é poderoso!");

    System.out.println(sb.toString());
      // Saída: Java é poderoso!
   }
}
```

## Usando StringBuffer

O StringBuffer funciona de maneira semelhante ao StringBuilder, mas é seguro para threads, o que significa que ele pode ser usado em ambientes multithreaded com segurança.

```
      StringBuffer sbr = new StringBuffer("Geeks");
      String str = sbr.toString();
      StringBuilder sbl = new StringBuilder(str);

      System.out.println(sbl);
```

# Diferenças entre String, StringBuilder e StringBuffer

A função de StringBuilder é muito semelhante à classe StringBuffer, pois ambas fornecem uma alternativa à classe String ao criar uma sequência mutável de caracteres. No entanto, a classe StringBuilder difere da classe StringBuffer com base na sincronização.

A classe StringBuilder não fornece nenhuma garantia de sincronização, enquanto a classe StringBuffer fornece. Portanto, esta classe foi projetada para uso como um substituto imediato para StringBuffer em locais onde o StringBuffer estava sendo usado por um único thread (como geralmente é o caso).

Sempre que possível, é recomendável que esta classe seja usada em preferência ao StringBuffer, pois será mais rápida na maioria das implementações. Instâncias de StringBuilder não são seguras para uso por vários threads.

## Imutabilidade

A classe String é imutável, enquanto StringBuilder e StringBuffer são mutáveis.

## Sincronização

StringBuffer é sincronizado (thread-safe), o que significa que ele pode ser usado com segurança em ambientes multithread.
StringBuilder não é sincronizado, o que o torna mais rápido para uso em ambientes de thread única.

## Desempenho

Devido à imutabilidade de String, toda vez que você faz uma modificação, um novo objeto é criado, o que pode afetar o desempenho se houver muitas operações de concatenação.
StringBuilder é geralmente preferido quando há muitas operações de manipulação de strings devido à sua eficiência.

# Resumo do módulo

Strings e Criação de Strings

Strings em Java representam sequências de caracteres e são imutáveis. Elas podem ser criadas diretamente usando aspas duplas (String nome = "Java";) ou instanciadas usando o operador new (String saudacao = new String("Olá") ;).
Essas formas são amplamente utilizadas para armazenar e manipular texto.

Propriedades Básicas e Manipulação Simples de Strings

length: Retorna o comprimento da String (minhaString.length()).
Concatenando Strings: Strings podem ser concatenadas com o operador + ou com o método .concat() ("Hello".concat(" World")).
String Formatada (format): Permite criar Strings com variáveis e formatações específicas, similar ao printf em C (String.format("Hello, %s!", "Java")).

Principais Métodos de Manipulação de Strings

Acessando Caracteres: charAt() retorna o caractere em um índice específico (minhaString.charAt(2)).
Comparando Strings: O método equals() compara o conteúdo de duas Strings,
enquanto equalsIgnoreCase() faz a comparação ignorando diferenças de maiusculas/minúsculas.

Diferença entre equals() e equalsIgnoreCase(): equals() é sensível a maiusculas e minúsculas, útil para casos em que a exatidão do conteúdo é essencial.
Já equalsIgnoreCase() é insensível a isso, sendo útil em comparações que ignoram diferenças de capitalização.
contains: Verifica se uma sequência específica de caracteres está presente na String (minhaString.contains("abc")).

## Métodos para Manipulação Avançada de Strings

Substrings: substring() extrai uma parte da String, com base em índices definidos (minhaString.substring(2, 5)).
replace e replaceAll: replace() substitui todas as ocorrências de um caractere ou substring; replaceAll() usa expressões regulares para substituição.
split: Divide uma String em várias partes, retornando um array de Strings (minhaString.split(" ")).
startWith e endsWith: Verificam se a String começa ou termina com uma sequência específica (minhaString.startsWith("Java")).

## Procurando Posições e Comparações Avançadas

indexOf e lastIndexOf: Retornam a primeira ou última posição de um caractere ou substring específica na String (minhaString.indexOf('a')).
valueOf: Converte valores de outros tipos para uma representação de String (String.valueOf(123)).
compareTo e compareToIgnoreCase: Comparam duas Strings lexicograficamente, retornando um valor inteiro que indica a ordem.
contentEquals: Verifica se o conteúdo de uma String corresponde ao de um StringBuilder ou outra sequência de caracteres.

## Outros Métodos Avançados

copyValueOf: Cria uma String a partir de um array de caracteres (String.copyValueOf(charArray)).
matches: Avalia se uma String corresponde a uma expressão regular (minhaString.matches("\\d+")).
regionMatches: Compara regiões específicas de duas Strings, útil para comparações parciais.
replaceAll: Substitui substrings que correspondem a uma expressão regular.

Transformação e Construção de Strings

toLowerCase e toUpperCase: Convertem a String para minúsculas ou maiusculas (minhaString.toUpperCase()).
trim: Remove espaços em branco no início e no fim de uma String (minhaString.trim()).
StringBuilder e StringBuffer: Classes para manipulação eficiente de Strings mutáveis.
StringBuilder é mais rápido, mas não é seguro para threads; StringBuffer é seguro para uso em threads.

Esses métodos fornecem uma ampla gama de ferramentas para criar, manipular e processar texto em Java, de forma a atender a diferentes necessidades de formatação, busca, comparação e modificação de dados textuais.

# Lista de Exercícios Propostos

1. Digite um nome, calcule e retorne quantas letras tem esse nome.
2. Faça um programa que leia um nome e imprima as 4 primeiras letras do nome.
3. Formate uma frase que inclua nome, idade e data de nascimento usando

String.format.

4. Ordene e compare várias Strings "apple", "banana", "Apple" e "Banana".
5. Verifique se uma String está no formato de e-mail válido usando matches.
6. Substitua todas as letras por "*" na frase "Security123!".
7. Compare duas Strings ignorando maiusculas e minúsculas em uma região específica.
8. Crie uma frase e remova palavras ou caracteres específicos com delete e replace.
9. Encontre e exiba todas as posições da palavra "Java" em uma frase longa.
10. Use StringBuffer para construir e substituir partes de uma frase em tempo de execução.
11. Escreva um programa que ordene Strings lexicograficamente usando compareTo.
12. Extraia diversas partes de uma frase longa e combine-as em uma nova frase.
13. Divida a frase "ID123: Name-Java| Country-AO" em partes usando split.
14. Use trim, replace e concatene uma frase eliminando espaços e substituindo símbolos.
15. Crie uma função que verifique se uma String é um palíndromo, ignorando maiusculas e minúsculas.
16. Encontre e remova todas as instâncias de uma palavra em uma frase com StringBuilder.
17. Valide um número de telefone no formato (XX) XXXX-XXXX usando matches.

18. Faça um programa que, a partir de uma string digitada pelo usuário, imprima:

a) O número de caracteres da string.
b) A string com todas suas letras em maiusculo.
c) O número de vogais da string.
d) Se a string digitada começa com “UNI” (ignorando maiusculas/minúsculas).
e) Se a string digitada termina com “RIO” (ignorando maiusculas/minúsculas).
f) O número de dígitos (0 a 9) da string.
g) Se a string é um palíndromo ou não.

19. Escreva um programa que dado um valor numérico digitado pelo usuário, imprima cada um dos seus dígitos por extenso.

Exemplo:
Entre o número: 4571 Resultado: quatro, cinco, sete, um

20. Escreva um programa que, a partir de um nome informado pelo usuário, exiba suas iniciais. As iniciais são formadas pela primeira letra de cada nome, sendo que todas deverão aparecer em maiusculas na saída do programa.

Note que os conectores e, do, da, dos, das, de, di, du não são considerados nomes e, portanto, não devem ser considerados para a obtenção das iniciais.
As iniciais devem ser impressas em maiusculas, ainda que o nome seja entrado todo em minúsculas.

Exemplos:
Maria das Graças Pimenta => MGP
João Carlos dos Santos => JCS
José da Silva => JS
Pedro Pereira Teixeira => PPT

21. Faça um programa que, a partir de um texto digitado pelo usuário, conte o número de caracteres total e o número de palavras (palavra é definida por qualquer sequência de caracteres delimitada por espaços em branco) e exiba o resultado.

22. Faça um programa que, a partir de um texto digitado pelo usuário, Imprima o texto removendo todos os espaços em branco adicionais encontrados, de modo que haja, no máximo, um espaço em branco separando as palavras presentes nesse texto.

23. Faça um programa que contenha um menu com as seguintes opções:

(a) Ler uma string S1 (tamanho máximo 20 caracteres);

(b) Imprimir o tamanho da string S1;

(c) Comparar a string S1 com uma nova string S2 fornecida pelo usuário e
imprimir o resultado da comparação;

(d) Concatenar a string S1 com uma nova string S2 e imprimir na tela o

resultado da concatenação;

(e) Imprimir a string S1 de forma reversa;

(f) Contar quantas vezes um dado caractere aparece na string S1. Esse caractere desse ser informado pelo usuário;

(g) Substituir a primeira ocorrência do caractere C1 da string S1 pelo

caractere C2. Os caracteres C1 e C2 serão lidos pelo usuário;

(h) Verificar se uma string S2 e substring de S1. A string S2 deve ser informada
pelo usuário;

(i) Retornar uma substring da string S1. Para isso o usuário deve informar a
partir de qual posição deve ser criada a
substring e qual é o tamanho da substring.

24. O código de César é uma das mais simples e conhecidas técnicas de criptografia. É um tipo de substituição na qual cada letra do texto é substituída por outra, que se apresenta no alfabeto abaixo dela um número fixo de vezes. Por exemplo, com uma troca de três posições, ‘A’ seria substituído por ‘D’, ‘B’ se tornaria ‘E’, e assim por diante.

Implemente um programa que faça uso desse Código de César (3 posições), entre com uma string e retorne a string codificada.

Exemplo:

String: a ligeira raposa marrom saltou sobre o cachorro cansado
Nova string: D OLJHLUD UDSRVD PDUURP VDOWRX VREUH R FDFKRUUR FDQVDGR

25. Faça um programa que receba duas frases distintas e imprima de maneira invertida, trocando as letras A por *.

26. Escreva um programa que substitui as ocorrências de um caractere ‘0’ em uma string por outro caractere ‘1’.

27. Faça um programa que receba uma palavra e a imprima de trás-para-frente.

28. Faça um programa que receba do usuário uma string. O programa imprime a string sem suas vogais.

29. Faça um programa que receba uma palavra e calcule quantas vogais (a, e, i, o, u) possui essa palavra. Entre com um caractere (vogal ou consoante) e substitua todas as vogais da palavra dada por esse caractere.

30. Escreva um programa que leia a idade e o primeiro nome de 10 pessoas.

Seu programa deve terminar quando uma idade negativa for digitada. Ao terminar, seu programa deve escrever o nome e a idade das pessoas mais jovens e mais velhas.

# MÓDULO 9: ARRAYS

**Na programação, nem o mais experiente dos pilotos chegou no ápice do aprendizado.**

Vitthin

# OBJECTIVOS

Ao final desse capítulo o leitor deverá ser capaz de:

- Criar, declarar e inicializar um array da melhor forma, bem como todas as formas possíveis de se inicializar um array.
- Formas de percorrer um array.
- Manipular Array Literal
- Inserindo e alterando os valores do array

# Introdução

Arrays são estruturas fundamentais em Java que nos permitem armazenar vários valores do mesmo tipo em uma única variável. Eles são úteis para gerenciar coleções de dados de forma eficiente. Arrays em Java funcionam de forma diferente do que em C/C++. Este capítulo aborda os conceitos básicos e explicações aprofundadas com exemplos de declaração, criação e manipulação de arrays em Java.

Arrays são fundamentais para a programação Java. Eles permitem que você armazene e manipule coleções de dados de forma eficiente.

Um array é um grupo de variáveis (chamados elementos ou componentes) que contém valores todos do mesmo tipo. Os arrays são objetos; portanto, são considerados tipos por referência. Como você logo verá, o que em geral consideramos um array é, na verdade, uma referência a um objeto array na memória. Os elementos de um array podem ser tipos primitivos ou tipos por referência, para referenciar um elemento particular em um array, especificamos o nome da referência para o array e o número de posição do elemento no array. O número de posição do elemento é chamado de índice ou subscrito.

Exemplo:

| 40 | 55 | 23 | 87 | 21 | 11 | 94 |
|---|---|---|---|---|---|---|

Num array existem elementos, e cada elemento encontra-se numa posição(índice), O primeiro elemento em cada array tem índice zero e às vezes é chamado de zero-ésimo elemento. Assim,

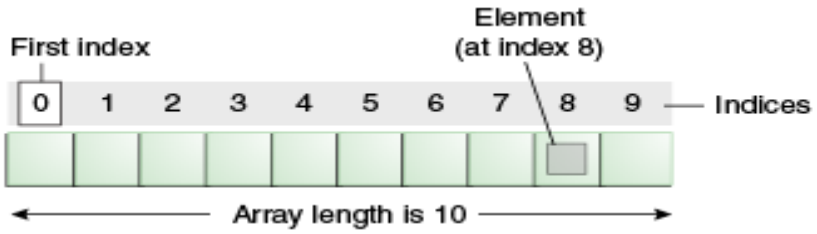
no exemplo acima: elemento 40 na posição 0, elemento 55 na posição 1, elemento 94 na posição 6.

Há três características principais de um array:

1. Alocação dinâmica: em matrizes, a memória é criada dinamicamente, o que reduz a quantidade de armazenamento necessária para o código.

2. Elementos armazenados sob um único nome: Todos os elementos são armazenados sob um nome. Esse nome é usado sempre que usamos um array.

3. Ocupa localização contígua: Os elementos nos arrays são armazenados em posições adjacentes. Isso torna fácil para o usuário encontrar as localizações de seus elementos.




# Declarando e criando arrays

Os objetos array ocupam espaço na memória. Como outros objetos, arrays são criados com a palavra-chave new. Para um objeto array, especifique o tipo dos elementos do array e o número de elementos como parte de uma expressão de criação de array que utiliza a palavra-chave new. Tal expressão retorna uma referência que pode ser armazenada em uma variável de array. A declaração e a expressão de criação de arrays a seguir cria um objeto array que contém 12 elementos int e armazenam a referência do array no variável c do array:

```
int[] c = new int[12];
```

Essa expressão pode ser utilizada para criar o array no exemplo acima, quando um array é criado, cada um de seus elementos recebe um valor padrão, zero para os elementos de tipo primitivo numéricos, false para elementos boolean e null para referências.

Como veremos mais adiante, pode-se fornecer valores não padrão de elementos ao criar um array.

Criar o array no exemplo abaixo, também pode ser realizado em dois passos, como a seguir:

```
int[] c; // declara a variável de array
c = new int[12]; // atribui à variável de array
```

Na declaração, os colchetes que seguem o tipo indicam que c é uma variável que referencia um array (isto é, a variável armazenará uma referência de array). Na instrução de atribuição, a variável de array c recebe a referência para um novo array de 12

- **<u>Erro Comum de Programação:</u>**

| |
|---|
| Em uma declaração de array, especificar o número de elementos entre os colchetes da declaração (por exemplo, int[12] c;) é um erro de sintaxe. |

Um programa pode criar vários arrays em uma única declaração. A declaração seguinte reserva 100 elementos para b e 27 elementos para x:

```
String[] b = new String[100], x = new String[27];
```

Quando o tipo de array e os colchetes são combinados no início da declaração, todos os identificadores na declaração são variáveis de array. Nesse caso, as variáveis b e x referem-se ao arrays String. Para maior legibilidade, preferimos declarar apenas uma variável por declaração. A declaração anterior é equivalente a:

```
String[] b = new String[100]; // cria array b
String[] x = new String[27]; // cria array x
```

- **<u>Uma Boa Prática de Programação:</u>**

| |
|---|
| Para legibilidade, declare apenas uma variável por declaração. Mantenha cada declaração em uma linha separada e inclua um comentário que descreva a variável sendo declarada. |

Quando apenas uma variável é declarada em cada declaração, os colchetes podem ser colocados após o tipo ou após o nome da variável de array, como em:

```
String b[] = new String[100]; // cria array b
String x[] = new String[27]; // cria array x
```

Um programa pode declarar arrays de qualquer tipo. Cada elemento de um array do tipo primitivo contém um valor do tipo de elemento declarado do array. De maneira semelhante, em um array de um tipo por referência, todo elemento é uma referência a um objeto do tipo de elemento declarado do array. Por exemplo, todo elemento de um array int é um valor int e todo elemento de um array String é uma referência a um objeto String.

## Vantagens dos Arrays em Java

- Os arrays Java permitem que você acesse qualquer elemento aleatoriamente com a ajuda de índices
- É fácil armazenar e manipular grandes conjuntos de dados

## Desvantagens dos Arrays em Java

- O tamanho do array não pode ser aumentado ou diminuído depois de declarado — os arrays têm um tamanho fixo
- Java não pode armazenar dados heterogêneos. Ele só pode armazenar um único tipo de primitivos

# Inicialização

Para inicializar um array em Java, você precisa alocar espaço para o array na memória e atribuir valores aos seus elementos. Existem algumas maneiras diferentes de inicializar um array em Java.

A melhor forma de inicializar um array, de acordo com as melhores práticas, é utilizar o operador new, como já apresentamos:

```
int[] novo_array = new int[100];
```

Precisamos detalhar, primeiramente, o tipo de dado (inteiros, caracteres, booleanos ou decimais), seguido dos colchetes (“[ ]”). Após isso, damos um nome para nosso array e usamos o sinal de igual para associá-lo a um novo conjunto de elementos, de determinado tipo de dado.

Precisamos dar a quantidade de posições disponíveis nesse conjunto entre os colchetes, que, no caso do exemplo acima, seria o valor de 100. Há outras formas de inicializar o array, e já descrevemos as principais delas, na seção “Qual a sintaxe do Java array” — se você perdeu essa parte, vale a pena voltar e conferir!

Casos de inicialização:

```
int[] numbers1 = new int[3];
int[] numbers2 = { 1, 2, 3 };
float[] boats = new float[5];
String[] theory = new String[] {"a", "b", "c"};
int[][] numbers5 = new int[5][];
int[][] numbers6 = new int[5][4];
```

## Inicialização por tamanho

Uma forma comum de inicializar um array é especificar seu tamanho na declaração e depois atribuir valores aos seus elementos. Por exemplo, para criar um array de inteiros com quatro elementos, você pode fazer o seguinte:

```
int[] numeros = new int[3];
numeros[0] = 1;
numeros[1] = 2;
numeros[2] = 3;
```

## Inicialização por valores

Outra maneira de inicializar um array em Java é listar seus valores na declaração. Por exemplo, para criar um array de inteiros com os valores 1, 2, 3 e 4, você pode fazer o seguinte:

```
int[] numeros = {1, 2, 3, 4};
```

## **Saiba. Que:**

Note ao criamos um array em java não existe diferença entre:
`String[] nomes` **ou** `String nomes[]`

Cria um array de cinco elementos com valores de índice de 0 a 4. O elemento n[0] é inicializado como 10, n[1] é inicializado como 20 etc. Quando o compilador encontrar uma declaração de array que inclua uma lista de inicializador, ele conta o número de inicializadores na lista para determinar o tamanho do array, depois configura a operação new apropriada “nos bastidores”.

# Acesso aos elementos de um array

Para acessar um elemento em um array em Java, você precisa usar seu índice. Os índices começam em zero para o primeiro elemento do array e aumentam em uma unidade para cada elemento subsequente. Aqui está um exemplo de como acessar o segundo elemento em um array de inteiros:

```
int[] numeros = {1, 2, 3, 4};
int segundoNumero = numeros[1];
// segundoNumero contém o valor 2
```

De lembrar que, para acessar o conteúdo de um array, utilizamos o índice de cada elemento. Lembre-se de que os índices em Java começam em 0, então o primeiro elemento está na posição `0`.

Exemplo:

```
int[] numeros = {10, 20, 30, 40, 50};
System.out.println(numeros[2]);
// Exibe 30, que está no índice 2
```

Demostração de acesso ao conteúdo de um array:

```
 public class Array {

     public static void main(String args[]) {

          int month_days[];

          month_days = new int[12];

          month_days[0]  = 31;

          month_days[1]  = 28;

          month_days[2]  = 31;

          month_days[3]  = 30;

          month_days[4]  = 31;

          month_days[5]  = 30;

          month_days[6]  = 31;

          month_days[7]  = 31;

          month_days[8]  = 30;

          month_days[9]  = 31;

          month_days[10] = 30;

          month_days[11] = 31;


        System.out.println("Abril  tem  "  +  month_days[3]  +  "
dias.");

     }

}
```

Funcionamento do exercício resumido

O programa cria um array de inteiros chamado **month_days** com **12** elementos, cada posição do array é inicializada com a quantidade de dias de um mês específico, o programa imprime o número de dias de abril usando **month_days[3**], que retorna **30**.

Este é um exemplo simples de como arrays podem ser usados para armazenar uma sequência de valores relacionados e como acessar valores específicos pelo índice.

# Inserindo valores em um array

Existem diversas formas de inserirmos valores em um array, no exemplo anterior dos dias do mês, já vimos esta operação que é a seguinte:

```
int[] a = new int[3];
a[0] = 124;
a[1] = 43;
a[2] = 1023;
```

# Alterando o Conteúdo de um Array

É possível modificar os elementos de um array acessando-os pelo índice e atribuindo um novo valor.

Exemplo:

```
int[] numeros = {10, 20, 30, 40, 50};
numeros[0] = 15;
// Altera o primeiro elemento para 15
System.out.println(numeros[0]); // Exibe 15
```

# Percorrendo um array

Uma maneira comum de percorrer todos os elementos de um array em Java é usar um loop for. Aqui está um exemplo de como imprimir todos os elementos de um array de inteiros:

```
int[] numeros = {1, 2, 3, 4};
for (int i = 0; i < numeros.length; i++) {
    System.out.println(numeros[i]);
}
```

Nesse exemplo, usamos a propriedade “length” do array para obter o número de elementos no array. Em seguida, usamos um loop for para iterar por todos os elementos do array, imprimindo cada um deles na tela.

- **Recomenda-se:**

| |
|---|
| Recomenda-se sempre utilizar o tamanho do array(length) como condição de parada do for para evitar o acesso a posições inválidas. |

Demonstração Array:

```
int[] arr = {1,2,3,4,5};
int n = arr.length;

// percorrendo array
for(int i = 0; i < n; i++) {
   System.out.print(arr[i]+" ");
}
```

Exercício1: Fazer um programa para ler 5 valores e, em seguida, mostrar todos os valores lidos, juntamente com o maior e a média dos valores.

```
public class Exercicio1 {
  public static void main(String[] args) {

 Scanner input = new Scanner(System.in);
    int[] numeros = new int[5];
    int somaDosValores = 0;
    int maxValor = 0, mediaDosValores = 0;

    for (int j = 0; j < numeros.length; j++) {

    System.out.printf("Digite um valor para a posicao: [%d] ",j);
         numeros[j] = input.nextInt();
    }

    System.out.printf("\nOs valores lidos são:  ");

    for (int i = 0; i < numeros.length; i++) {
         if (numeros[i] > maxValor) {
            maxValor = numeros[i];
         }
        somaDosValores += numeros[i];
        System.out.printf(" %d ", numeros[i]);
    }

     mediaDosValores = somaDosValores / 5;
     System.out.println("\nO maior valor eh: "+maxValor);
     System.out.println("A media dos valores eh: "+mediaDosValores);
         }
}
```

Esse código é um programa Java que lê cinco números inteiros inseridos pelo usuário, exibe esses números, calcula o maior número inserido e a média dos valores. A seguir, explico cada parte do código:

Estrutura do Código

```
public class Exercicio1 {
    public static void main(String[] args) {
```

Essa linha define uma classe chamada Exercicio1 e um método main, que é o ponto de entrada do programa.

```
Scanner input = new Scanner(System.in);
```

A linha acima cria um objeto Scanner chamado input, que permite ler dados inseridos pelo usuário no console.

Declaração de Variáveis

```
int[] numeros = new int[5];
int somaDosValores = 0;
int maxValor = 0, mediaDosValores = 0;
```

- `int[] numeros = new int[5];`
  Declara e inicializa um array numeros com 5 posições, onde serão armazenados os cinco valores inseridos pelo usuário.

- `int somaDosValores = 0;`
  Inicializa a variável **somaDosValores** para armazenar a soma dos valores inseridos.

- `int maxValor = 0;`
  Inicializa a variável **maxValor** com zero, que será usada para armazenar o maior valor entre os números inseridos.

- `int mediaDosValores = 0;`

  Inicializa **mediaDosValores**, que armazenará a média dos valores inseridos.

Leitura dos Valores do Usuário

```
for (int j = 0; j < numeros.length; j++) {
   System.out.printf("Digite um valor para a posicao: [%d] ", j);
     numeros[j] = input.nextInt();
}
```

Esse for percorre cada posição do array numeros (de 0 a 4) e pede ao usuário que insira um valor para cada posição. O valor digitado é então armazenado no índice correspondente de numeros.

Exibição dos Valores e Cálculo do Maior Valor e da Soma

```
System.out.printf("\nOs valores lidos são:  ");
for (int i = 0; i < numeros.length; i++) {
     if (numeros[i] > maxValor) {
     maxValor = numeros[i];
     }
     somaDosValores += numeros[i];
     System.out.printf(" %d ", numeros[i]);
}
```

Aqui, temos um segundo loop for que percorre o array numeros para:

1. Verificar se o valor atual (numeros[i]) é maior que o valor armazenado em maxValor. Se for, maxValor é atualizado com o valor atual.

2. Somar o valor atual a somaDosValores.
3. Exibir os valores lidos na tela.

Cálculo da Média dos Valores

```
mediaDosValores = somaDosValores / 5;
```

Aqui, a média dos valores é calculada dividindo a soma total (somaDosValores) pelo número de elementos (5) e o resultado é armazenado em mediaDosValores.

Exibição do Maior Valor e da Média

```
System.out.println("\nO maior valor eh: " + maxValor);
```

```
System.out.println("A media dos valores eh: " + mediaDosValores);
```

Essas duas linhas exibem o maior valor (maxValor) e a média dos valores (mediaDosValores) no console.

Funcionamento Completo do Programa

O programa solicita cinco números do usuário, calcula e exibe os seguintes resultados:

- Os valores lidos.
- O maior valor entre eles.
- A média dos valores.

Exemplo de Entrada e Saída

Exemplo de entrada do usuário:

Digite um valor para a posicao: [0] 10

Digite um valor para a posicao: [1] 15

Digite um valor para a posicao: [2] 20

Digite um valor para a posicao: [3] 5

Digite um valor para a posicao: [4] 25

Saída esperada:

Os valores lidos são:  10 15 20 5 25

O maior valor eh: 25

A media dos valores eh: 15

- **<u>Observações:</u>**

| |
|---|
| O programa assume que o usuário sempre insere inteiros válidos.<br>O cálculo da média utiliza divisão inteira. Se **somaDosValores** não for divisível exatamente por 5, o resultado será arredondado para baixo. |

# ForEach

As estruturas de repetição Java for Each ou, como também são conhecidas, enhanced-for loop, foram adicionadas a partir da 5º (quinta) versão do Java. Trouxeram consigo facilidades para as pessoas desenvolvedoras durante a manipulação de arrays e coleções, possibilitando uma escrita de código mais limpa ao realizar a iteração desses elementos.

A expressão de língua inglesa “for each” pode ser traduzida como “para cada”. Em Java, os métodos iteradores for-each e forEach(), explicando de maneira bem resumida, são usados para iteração de listas e coleções. Sua principal característica é a simplificação do loop for deixando o código mais leve e fácil de ser lido pelas pessoas responsáveis pelo desenvolvimento e pela manutenção.

Não há alteração quanto à performance do programa em que um laço forEach é aplicado na maioria das vezes. Isso se deve ao fato de que seu funcionamento ocorre da mesma maneira da estrutura for, aquela mais convencional, durante o processamento do código.

Em alguns casos, no entanto, precisamos acessar coleções de estruturas de dados específicas, por exemplo. O uso do forEach(), como iterador, pode providenciar um processamento significativamente mais rápido para as operações. Normalmente, usamos a estrutura for-each para que o laço for percorra todo o array,  essa é uma das suas principais funcionalidades. Fazendo isso, nos precavemos de problemas relacionados à manipulação das posições ocupadas pelos elementos de um array (como iniciar sua contagem pelo número 1, e não pelo 0, como fazem algumas pessoas) ou ocupar, sem querer, todas as posições durante o processo de iteração e bugar nossa aplicação.

# A Sintaxe do forEach

A sintaxe do loop forEach consiste na identificação do tipo de estrutura dos dados utilizados, seguidos por dois pontos (:) posteriormente, usamos a identificação do array ou da coleção que queremos utilizar, assim:

```
for(<Tipo> <identificação> : <array ou coleção>) {
    <comando>
}
```

Repare que os dois pontos (:) podem ser lidos como “em”. No caso do exemplo acima, poderíamos ler o código da seguinte maneira: para cada elemento contido em um array, uma instrução é dada. Ela seria equivalente ao uso do laço for normal, como no seguinte exemplo:

```
for (int i = 0; i<arr.length; i++) {
    type var = arr[i];
    declarações da variável;
}
```

Se o array utilizado (no caso do exemplo a seguir, “meuArray”) for um número inteiro, ficaria assim com o uso de um loop for-each:

```
for (int count : meuArray){
    <comando>
}
```

A partir do Java 8, é possível a utilização do método forEach() — o qual serve melhor para situações específicas, como veremos durante nosso guia:

```
list.forEach(meuArray -> meuArray.operação);
```

Para facilitar ainda mais seu entendimento, acompanhe, a seguir, o mesmo código sendo escrito com o laço for-each e, em seguida, usando o método forEach().

## Usando o for-each loop clássico

```
import java.util.*;

public class ExemploForEach {

    public static void main(String[] args) {

List<Integer> i = Arrays.asList(15, 11, 13, 9, 35);

    for(Integer item : i){
        System.out.print(item);
    }
  }
}
```

Usando o método forEach()

```
import java.util.*;

public class ExemploForEach {
    public static void main(String[] args) {

List<Integer> itens = Arrays.asList(15, 11, 13, 9, 35);
    itens.forEach(System.out::println);
    }
}
```

# As vantagens e desvantagens do método Java forEach() em relação ao for-each loop na linguagem Java

Os modelos de loop que levantamos durante este guia, o forEach clássico e o método forEach(), são usados para casos específicos, com cada um deles oferecendo uma vantagem ou desvantagem para determinada situação. Agora, abordaremos as principais delas.

## Otimização de desempenho

Embora sejam raros os casos em que a troca do iterador forEach() pelo for-each convencional oferece uma melhoria significativa no desempenho da aplicação, é algo a ser avaliado. Pacotes e bibliotecas usados em Java, certamente, terão algumas otimizações possíveis no quesito desempenho ao optarem por esse método.

## Código mais sustentável e explícito durante a manutenção

Como vimos no exemplo acima, durante a apresentação das sintaxes utilizadas pelo forEach, o uso do método forEach() se torna bastante viável devido à facilitação dos processos de depuração do código. Ao mesmo tempo, simplificam também a adaptação e a extensibilidade da aplicação por outras pessoas desenvolvedoras.

## Opção de suporte para execução paralela de recursos

O uso do iterador forEach() permite que o código tenha a possibilidade de ser otimizado — nos casos em que a implementação padrão do fluxo de dados não atenda aos requerimentos da aplicação desenvolvida. É possível que forneçamos nossa própria implementação, como configurações e preferências internas do sistema operacional — e sem prejudicar o código como um todo.

## Iterando os nomes e imprimindo uma lista de convidados

Aqui, trazemos um exemplo bem simples, mas muito efetivo, para o uso dos loops forEach. Perceba que pedimos para o sistema que criamos imprimir um título para a lista e, em seguida, iterar os valores contidos no vetor “convidados”:

```
import java.util.ArrayList;
import java.util.List;


public class ExemploForEach {
  public static void main(String[] args) {
    String[] convidados = {"Maria", "Francisco", "Jair", "Benedita", "Joana", "Xavier"};


//iterando o Array com forEach:
```

```
    System.out.println("Os convidados para a festa são:");
    for(String festa : convidados) {
    System.out.println(festa);
    }
  }
}
```

### As diferenças principais entre o for-each loop e o método forEach()

Como já havíamos mostrado no começo de nosso guia, se buscarmos a explicação mais simples para esse tema, podemos entender que tanto o loop for quanto o ForEach loop têm a mesma funcionalidade eles são responsáveis por iterar os elementos em dada coleção, como os ArrayLists, que abordamos durante nossos exemplos.

Se olharmos de maneira um pouco mais aprofundada para os dois, veremos que são iteradores diferentes e que se comportam de maneira bastante diversa entre si. Não é à toa que o forEach (ou for-each loop) também é chamado de enhanced loop. Basicamente, é uma estrutura de controle for um pouco mais avançada e mais limpa de ser implementada do que a convencional.

## O loop for-each como um iterador externo

Ao trabalhar com iteradores externos, precisamos especificar para eles como as ações serão realizadas durante o processo. Isso acontece por conta de como a iteração é processada durante o loop. Dessa forma, o for-each loop alterna o “o quê” e “como” ele fará a iteração.

Considere o seguinte exemplo de loop:

```
for (String nome : nomes) {
    System.out.println(nome);
}
```

Embora não seja explícito às pessoas que programam esses loops, saiba que, por exemplo, os métodos next() e hasnext(), usados para invocar o próximo elemento da coleção e para saber se o loop chegou ao fim, respectivamente, não está exposto no código, mas é acessado durante a execução.

## O método forEach() como iterador interno

Já quando falamos de um iterador interno, como o método forEach(), a própria coleção acessada é responsável por fazer a iteração — precisamos apenas dos comandos, para que contabilizem cada elemento da coleção em que trabalhamos.

O método forEach() é capaz de gerenciar a iteração em plano de fundo, permitindo à pessoa programadora apenas escrever as ações que devem ser realizadas nos elementos de determinada coleção, resumidamente. Confira mais esse exemplo:

```
names.forEach(name -> System.out.println(name));
```

No método forEach() que usamos como exemplo logo acima, observamos que é utilizada uma declaração lambda — que é um pequeno bloco de código que recebe parâmetros e retorna valores. Isso quer dizer que apenas o próprio método precisa saber o que deve ser feito, passando toda a ação de iteração para a parte interna de processamento.

Concluindo, vimos que o Java forEach é um elemento que pode ser implementado a partir da versão do Java 8 e vem sendo atualizado desde então. Na maior parte das vezes, ele pode ser mais conveniente para a implementação do que o loop for convencional.

# Array Literal em Java

Um Java Array Literal é uma maneira concisa de declarar e criar um array em uma única instrução. Ele usa uma notação abreviada onde os elementos no array são separados por vírgulas e colocados entre colchetes [ ]. Esses literais podem armazenar todos os tipos de objetos e qualquer número de dimensões. Por exemplo, você pode declarar um array com qualquer número de dimensões, como unidimensional, bidimensional ou tridimensional.

Literais de array Java são uma maneira conveniente de criar e inicializar um array em uma linha de código. Eles também são úteis para criar arrays de objetos, como strings, números e booleanos. Além disso, eles podem ser usados para criar arrays multidimensionais, que podem ser usados para armazenar dados de uma forma mais organizada.

## Sintaxe e formatação de um literal de array

Sintaxe-wise, um literal de array é escrito com colchetes [ ] em ambos os lados, com elementos dentro do array separados por vírgulas. Cada elemento pode armazenar qualquer tipo de objeto, como strings, inteiros, longs, floats e assim por diante. Aqui está um exemplo de um literal de array unidimensional com cinco elementos:

```
["Olá", "Isto", "é", "um", "Java Array Literal"]
```

- **NOTA:**

| |
|---|
| Esses arrays são ideais para listas de constantes ou valores conhecidos. |

# Exemplos de criação de um literal de array

Para ilustrar a criação de um literal de array em Java, aqui está um exemplo de criação de um array unidimensional com cinco elementos:

String[] shoppingList = {“Leite”, “Pão”, “Queijo”, “Ovos”, “Manteiga”};

E aqui está um exemplo de criação de uma matriz bidimensional:

int[][] meuArray = {{1, 2}, {3, 4}, {5, 6}};

Por fim, aqui está um exemplo de criação de uma matriz tridimensional:

char[][][] meuArray = {{{'a', 'b'}, {'c', 'd'}}, {{'e', 'f'}, {'g', 'h'}}};

É importante notar que o número de elementos em cada dimensão do array deve ser especificado ao criar o array. Por exemplo, se você quisesse criar um array bidimensional com três elementos na primeira dimensão e quatro elementos na segunda dimensão, você precisaria especificar isso ao criar o array.

## Melhores práticas para trabalhar com literais de array

- Declare literais de array apenas como estruturas de dados temporárias ou intermediárias, pois elas não são dinâmicas e não podem ser redimensionadas facilmente.
- Certifique-se de que os literais da matriz sejam concisos e fáceis de ler; não deve haver muitos elementos declarados em uma linha.
- Use nomes de variáveis significativas que reflitam com precisão o que está sendo armazenado na matriz.
- Sempre que possível, use blocos de inicialização em vez de literais ao trabalhar com grandes quantidades de dados.

E isso cobre tudo para criar um Array Literal.

# Array Multidimensional

Uma matriz é um conjunto ordenado de elementos do mesmo tipo, primitivo ou referência. Informações gerais sobre arrays (principalmente unidimensionais), aqui falaremos sobre arrays cujos elementos são outros arrays. Essas matrizes são chamadas multidimensionais. Uma matriz cujos elementos são outras matrizes, ou seja, uma matriz de matrizes, é chamada de bidimensional. Nem todas as linguagens possuem arrays multidimensionais estruturados dessa forma, mas em Java é esse o caso.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}.$$

Talvez tudo isso parece muito abstrato, então agora vamos passar para as manifestações concretas de matrizes multidimensionais - bidimensionais e tridimensionais. O fato é que os desenvolvedores Java às vezes usam matrizes bidimensionais, com muito menos frequência - matrizes tridimensionais, e matrizes ainda maiores são extremamente raras. Há uma grande probabilidade de você não os encontrar.

- **<u>Esclarecer:</u>**

  - O termo array é o nome técnico mais correto em Java, pois define qualquer coleção sequencial de dados do mesmo tipo.
  - Matriz é comumente usado para arrays bidimensionais, mas tecnicamente continua sendo um array.

# Array bidimensional

Um array bidimensional em Java é um array de arrays, ou seja, cada célula contém uma referência a um array. Mas é muito mais fácil apresentá-lo na forma de uma tabela que possui um determinado número de linhas (primeira dimensão) e número de colunas (segunda dimensão). Uma matriz bidimensional em que todas as linhas possuem um número igual de elementos é chamada de retangular.

```
int[][] matriz = {{1, 2}, {3, 4}};
```

## Declarando, criando e inicializando arrays bidimensionais

O procedimento para declarar e criar um array bidimensional é quase o mesmo que no caso de um array unidimensional:

```
int[][] twoDimArray = new int[3][4];
```

Esta matriz possui 3 linhas e 4 colunas. O tamanho de um array bidimensional retangular (podem não ser retangulares, mais sobre isso abaixo), ou seja, o número total de elementos pode ser determinado multiplicando o número de linhas pelo número de colunas. Agora está inicializado (preenchido) com valores padrões. Ou seja, zeros. Vamos preenchê-lo com os valores que precisamos.

```
twoDimArray[0][0] =  5;
twoDimArray[0][1] =  7;
twoDimArray[0][2] =  3;
twoDimArray[0][3] = 17;
twoDimArray[1][0] =  7;
twoDimArray[1][1] =  0;
twoDimArray[1][2] =  1;
twoDimArray[1][3] = 12;
twoDimArray[2][0] =  8;
twoDimArray[2][1] =  1;
twoDimArray[2][2] =  2;
twoDimArray[2][3] =  3;
```

Tal como acontece com matrizes unidimensionais, você pode executar o procedimento de inicialização mais rapidamente:

## Exibindo uma matriz bidimensional na tela

A maneira mais lógica de fazer esta operação é primeiro gerar a linha zero elemento por elemento, depois o segundo e assim por diante. A maneira mais comum de gerar um array bidimensional em Java é usar dois loops aninhados.

```
int [][] twoDimArray = {{5,7,3,17}, {7,0,1,12}, {8,1,2,3}};
for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
                System.out.print(" " + twoDimArray[i][j] + " ");
        }
      System.out.println();
}
```

## Saída rápida de uma matriz bidimensional

A maneira mais curta de exibir uma lista de elementos de um array bidimensional na tela é usar o método deepToStringde classe Arrays. Exemplo:

```
int[][] myArray = {{18,28,18},{28,45,90},{45,3,14}};
System.out.printLn(Arrays.deepToString(myArray));
```

O resultado do programa é a seguinte saída:

[[18, 28, 18], [28, 45, 90], [45, 3, 14]]

## “**Comprimentos**” de uma matriz bidimensional

Para obter o comprimento de um array unidimensional (ou seja, o número de elementos nele), você pode usar a variável length. Ou seja, se definirmos um array int a[] = {1,2,3}, então a operação a.length retornará 3. Mas e se aplicarmos o mesmo procedimento ao nosso array bidimensional.

```
int [][] twoDimArray = {{5,7,3,17}, {7,0,1,12}, {8,1,2,3}};
System.out.println(twoDimArray.length);
```

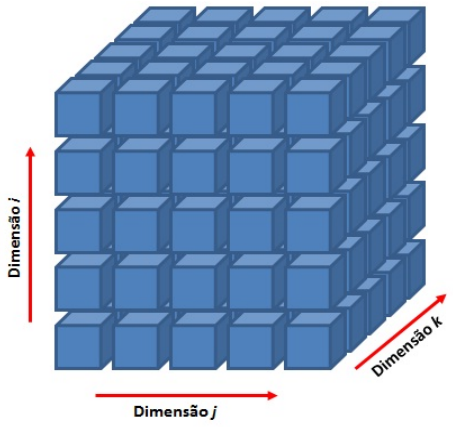
**Saída:** 3 Portanto, esta operação gera o número de linhas na matriz. Como obter o número de colunas? Se estamos lidando com matrizes bidimensionais retangulares (ou seja, aquelas em que todas as linhas têm o mesmo comprimento), podemos aplicar a operação twoDimArray[0].length ou em vez do elemento zero (essencialmente a linha zero) - qualquer outro existente. Podemos fazer isso porque em Java, um array bidimensional é um array de arrays e o elemento zero twoDimArray[0] é um array de comprimento 4. Você mesmo pode verificar isso.

# Matrizes tridimensionais em Java

Seguindo o bom senso e a lógica da linguagem Java, um array tridimensional pode ser chamado de “array de arrays de arrays” ou “um array cujo cada elemento é um array bidimensional”. Além disso, essas matrizes bidimensionais podem ser diferentes. Exemplo:

```
int[][][] threeDimArr = new int[2][][];
    threeDimArr[0] = new int[5][2];
    threeDimArr[1] = new int[1][1];
```

Porém, mais frequentemente, na prática, existem matrizes tridimensionais nas quais todas as três quantidades são definidas ao mesmo tempo, um análogo das matrizes bidimensionais retangulares.




Como já mencionei, arrays tridimensionais ou mais raramente são usados. No entanto, você pode programar algo interessante com um array 3D. Por exemplo, um estacionamento de vários andares. Cada andar pode ser considerado uma matriz bidimensional, e uma vaga de estacionamento pode ser considerada um elemento específico de uma matriz tridimensional. Um elemento de tal array pode ser representado por um tipo boolean com valor falso se o espaço estiver livre e verdadeiro se o espaço estiver ocupado.

```
boolean[][][] parkingLot = new boolean[3][2][5];

        parkingLot[0][1][0] = true;
        parkingLot[0][1][3] = true;

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 5; k++) {
              System.out.print("arr[" + i + "][" + j + "][" + k + "] = " +
parkingLot[i][j][k] + "\t");
                }
           System.out.println();
        }
    }
```

## Matrizes multidimensionais no trabalho real de um programador Java

Na realidade, a maioria dos desenvolvedores Java não encontra arrays multidimensionais com muita frequência. No entanto, há uma série de tarefas para as quais esta estrutura de dados é muito adequada.

- Para testes e configuração de matrizes como constantes para verificar um algoritmo específico.
- Às vezes, matrizes multidimensionais são usadas para redes neurais.
- Matrizes multidimensionais são adequadas para arquivadores.
- Trabalhando com imagens.

# Copiando arrays

A classe System em Java oferece um método chamado arraycopy, que permite copiar dados de um array (matriz) para outro de maneira eficiente e rápida. Este método é muito útil quando você precisa duplicar ou mover elementos entre arrays sem usar loops explícitos, o que torna o código mais simples e geralmente mais rápido.

## Estrutura do método arraycopy

A chamada ao método arraycopy tem a seguinte estrutura:

```
System.arraycopy(Object src, int srcPos, Object
dest, int destPos, int length);
```

### Cada parâmetro tem um papel específico:

- src: o array de origem, ou seja, o array de onde os dados serão copiados.
- srcPos: a posição inicial no array de origem (índice) a partir da qual a cópia começará.
- dest: o array de destino, para onde os dados serão copiados.
- destPos: a posição inicial no array de destino (índice) onde os dados serão armazenados.
- length: o número de elementos a serem copiados do array de origem para o array de destino.

Exemplo de uso

Aqui está um exemplo de como utilizar o método System.arraycopy para copiar elementos de um array para outro:

```
public class ExemploArrayCopy {
        public static void main(String[] args) {
        int[] origem = {1, 2, 3, 4, 5};   // Array de origem
        int[] destino = new int[5];
        // Array de destino com tamanho suficiente

        // Copiando todos os elementos de 'origem' para 'destino'
        System.arraycopy(origem, 0, destino, 0, origem.length);

        // Exibindo o array de destino para verificar a cópia
        for (int i : destino) {
            System.out.print(i + " ");      // Saída: 1 2 3 4 5
        }
    }
}
```

No exemplo acima:

- O array de origem possui cinco elementos: {1, 2, 3, 4, 5}.
- O array destino é criado com o mesmo tamanho de origem, mas inicia vazio {0, 0, 0, 0, 0}.
- O método System.arraycopy(origem, 0, destino, 0, origem.length) copia todos os elementos do array origem para o array destino, começando no índice 0 de ambos os arrays e copiando o total de origem.length elementos.

Depois da cópia, o array destino conterá os mesmos valores que origem, ou seja, {1, 2, 3, 4, 5}.

Vantagens de usar System.arraycopy

- Eficiência: System.arraycopy é mais rápido do que um loop for manual para copiar elementos, pois é otimizado em nível de sistema.
- Simplicidade: evita a necessidade de um código mais complexo com loops para copiar elementos, tornando o código mais legível.

- Flexibilidade: você pode especificar o ponto de início e o número de elementos para copiar, o que permite copiar apenas uma parte do array.

## ● Observações importantes

- O tipo de dados dos arrays de origem e destino deve ser compatível; caso contrário, ocorrerá um erro ArrayStoreException.

- O método não aumenta automaticamente o tamanho do array de destino. O array de destino deve ter tamanho suficiente para receber os elementos copiados; caso contrário, ocorre uma exceção IndexOutOfBoundsException.

# Arrays no nosso dia a dia

É bastante complicado explorarmos o uso de array no dia a dia das pessoas programadoras. Isso se deve ao fato de a aplicação desse recurso está intimamente ligada a uma imensa variedade de recursos utilizados por diversas das funções, métodos e classes disponíveis nos inúmeros pacotes que atendem à linguagem Java.

A cada atualização da linguagem, temos vários desses elementos como novidade. É verdade que veremos muito sua aplicação quando trabalhamos no desenvolvimento de jogos e coisas mais simples, o princípio didático ainda se torna bastante expoente para o array.

Como já temos estruturas e frameworks que lidam mais facilmente com os arrays, optamos por eles na maioria das vezes. Isso porque trarão maior dinamicidade, confiança e agilidade para o andamento de nossa aplicação que está sendo desenvolvida, aqui está uma lista de alguns exemplos práticos de onde usamos o conceito de arrays no cotidiano:

1. Lista de Compras ou Itens

Imagine uma lista de compras em um supermercado. Cada item, como "arroz", "feijão", "leite", etc., é armazenado em uma sequência organizada. Podemos pensar nessa lista como um array onde cada posição contém um item que queremos comprar.

Da mesma forma, se quisermos organizar os itens por ordem de importância, ou separá-los por categoria, estamos manipulando dados de uma maneira que seria muito próxima de arrays.

2. Agenda Telefônica

Em uma agenda de contatos, temos uma lista de pessoas, cada uma com informações como nome, telefone e endereço. Essa lista pode ser representada como um array onde cada posição armazena os dados de um contato.

Esse “array” de contatos pode ser útil quando queremos encontrar rapidamente uma pessoa específica ou armazenar novos contatos na próxima posição disponível.

3. Calendário

Um calendário é uma boa representação de um array bidimensional (um array dentro de outro). Cada linha do calendário representa uma semana, e cada coluna representa um dia.

Isso também é aplicável a horários de aula ou compromissos. Se temos uma tabela de horários para organizar o dia, podemos usar um array para armazenar as atividades de cada hora do dia.

6. Playlist de Músicas ou Vídeos

Em aplicativos de música ou vídeo, as playlists são basicamente arrays de faixas de música ou vídeos, onde cada faixa ocupa uma posição específica na sequência. Isso permite a reprodução organizada das músicas e facilita a seleção de uma música específica.

Similarmente, você pode reorganizar, adicionar ou remover músicas da playlist, o que é basicamente uma operação de inserção ou exclusão em um array.

7. Dados de Sensores em um Dispositivo Eletrônico

Dispositivos como termostatos ou monitores de batimentos cardíacos coletam dados em intervalos regulares e os armazenam em arrays. Por exemplo, um sensor de temperatura pode armazenar as leituras de temperatura de cada minuto em um array para análise posterior.

Isso permite uma fácil análise de tendências ao acessar os dados históricos, simplesmente acessando as posições do array.

8. Organização de Notas em uma Turma

Imagine as notas dos alunos em uma disciplina. Essas notas podem ser armazenadas em um array, onde cada posição representa a nota de um aluno específico.

A utilização de arrays facilita operações como calcular a média da turma, identificar as melhores e piores notas, ou ainda listar todas as notas para análise.

Esses exemplos mostram que os arrays, ao organizarem dados de maneira sequencial e estruturada, ajudam a manter a informação organizada e facilitam o acesso rápido a qualquer dado, exatamente como fazemos ao organizar informações em listas, tabelas, ou sequências no dia a dia.

# Lista de Exercícios Propostos

1. Crie um array de inteiros com 5 elementos e imprima seus valores iniciais (sem inicialização).

2. Declare um array de Strings para armazenar os nomes dos dias da semana. Preencha-o com os nomes dos dias e imprima o conteúdo do array.

3. Declare e inicialize um array de double com valores de temperatura para cada mês do ano (12 posições). Atribua valores aleatórios e exiba todos os valores.

4. Crie um array de char de 10 elementos e inicialize-o com letras aleatórias. Em seguida, imprima todos os elementos.

5. Crie um array de int com 5 elementos, preencha com números de 1 a 5, e acesse o terceiro elemento para exibir seu valor no console.

6. Declare um array de float com 4 elementos, preencha-o com notas de uma disciplina e acesse o último elemento.

7. Crie um array de int com 4 posições, atribua valores e substitua o valor do primeiro e do último elemento por zero. Em seguida, exiba o array completo.

8. Declare um array de String com o nome de 5 frutas. Altere a terceira fruta para "Kiwi" e exiba todo o array.

9. Utilize a sintaxe de array literal para criar um array de int com os valores {10, 20, 30, 40, 50} e exiba os elementos.

10. Declare um array de boolean usando uma inicialização inline (true, false, true, false) e exiba os elementos.

11. Crie um array de String com 5 nomes. Use um loop for para imprimir todos os nomes em ordem.

12. Declare um array de int com 10 posições e preencha-o com valores de 1 a 10. Em seguida, use um loop for-each para imprimir cada valor multiplicado por 2.

13. Declare um array de String diretamente com nomes de cores ("vermelho", "azul", "verde", "amarelo") e exiba todas as cores usando um loop for.

14. Crie um array literal de double com temperaturas dos últimos 5 dias. Em seguida, percorra o array e exiba cada valor.

15. Crie um array bidimensional de int com 3x3 posições, inicialize com números de 1 a 9, e imprima todos os elementos na forma de uma matriz 3x3.

16. Crie um array bidimensional de String para armazenar o nome e o telefone de 3 pessoas. Exiba todos os nomes e telefones.

17. Declare e inicialize um array 2D de int com 2x4 posições e atribua valores. Exiba todos os valores em uma tabela.

18. Crie um array 2D de double para representar notas de 4 alunos em 3 disciplinas. Inicialize com valores e calcule a média de cada aluno.

19. Crie um array de inteiros com valores {5, 3, 9, 1, 4}. Use a classe java.util.Arrays para ordenar o array em ordem crescente e exiba o array ordenado.

20. Crie um array de String com nomes de 3 frutas usando a classe java.lang.reflect.Array. Exiba os elementos do array usando um loop for.

21. Criar um vetor A com 5 elementos inteiros. Construir um vetor B de mesmo tipo e tamanho e com os "mesmos" elementos do vetor A, ou seja, B[i] = A[i].

22. Criar um vetor A com 8 elementos inteiros. Construir um vetor B de mesmo tipo e tamanho e com os elementos do vetor A multiplicados por 2,

ou seja: B[i] = A[i] * 2.

23. Criar um vetor A com 15 elementos inteiros. Construir um vetor B de mesmo tipo e tamanho, sendo que cada elemento do vetor B deverá ser o quadrado do respectivo elemento de A, ou seja: B[i] = A[i] * A[I].

24. Criar um vetor A com 15 elementos inteiros. Construir um vetor B de mesmo tamanho, sendo que cada elemento do vetor B deverá ser a raiz quadrada do respectivo elemento de A, ou seja: B[i] = sqrt(A[i]).

25. Criar um vetor A com 10 elementos inteiros. Construir um vetor B de mesmo tipo e tamanho, sendo que cada elemento do vetor B deverá ser o respectivo elemento de A multiplicado por sua posição (ou índice),

ou seja: B[i] = A[i] * i.

26. Criar dois vetores A e B cada um com 10 elementos inteiros. Construir um vetor C, onde cada elemento de C é a soma dos respectivos elementos em A e B, ou seja: C[i] = A[i] + B[i].

27. Criar dois vetores A e B cada um com 10 elementos inteiros. Construir um vetor C, onde cada elemento de C é a subtração dos respectivos elementos em A e B, ou seja: C[i] = A[i] – B[i].

28. Criar dois vetores A e B cada um com 10 elementos inteiros. Construir um vetor C, onde cada elemento de C é a multiplicação dos respectivos elementos em A e B, ou seja: C[i] = A[i] * B[i].

29. Armazenar 10 nomes em um vetor NOME e imprimir uma listagem numerada.

30. Armazenar 15 números inteiros em um vetor NUM e imprimir uma listagem numerada contendo o número e uma das mensagens: par ou ímpar.

31. Armazenar 8 números em um vetor e imprimir todos os números. Ao final, teremos o total de números múltiplos de seis digitados.

32. Armazenar nomes e notas das PR1 e PR2 de 15 alunos. Calcular e armazenar a média arredondada. Armazenar também a situação do aluno: AP ou RP. Imprimir uma listagem contendo nome, notas médias e situação de cada aluno tabulando.

33. Armazenar nome e salário de 20 pessoas. Calcular e armazenar o novo salário sabendo-se que o reajuste foi de 8%. Imprimir uma listagem numerada com nome e novo salário.

34. Criar um algoritmo que leia o preço de compra e o preço de venda de 100 mercadorias,

O algoritmo deverá imprimir quantas mercadorias proporcionam:

* lucro < 10 %
* 10 % <= lucro <= 20 %
* lucro > 20 %

35. Criar o algoritmo que deixe entrar com nome e idade de 20 pessoas e armazene em um vetor todos os nomes que comecem pela letra do

intervalo L - S.

36. Armazenar código, nome, quantidade, valor de compra e valor de venda de 30 produtos. A listagem pode ser de todos os produtos ou somente de um ao se digitar o código.

37. Criar um algoritmo que leia dois conjuntos de números inteiros, tendo cada 284 um 10 e 20 elementos e apresente os elementos comuns aos conjuntos. Lembre-se de que os elementos podem se repetir, mas não podem aparecer repetidos na saída.

38. Criar um algoritmo que leia vários números inteiros e positivos. A leitura se encerra quando encontrar um número negativo ou quando o vetor ficar completo. Sabe-se que o vetor possui, no máximo, 10 elementos. Gerar e imprimir um vetor onde cada elemento é o inverso do correspondente do vetor original.

39. Ler um vetor vet de 10 elementos e obter um vetor w cujos componentes são os fatoriais dos respectivos componentes de v.

40. Uma pessoa muito organizada gostaria de fazer um algoritmo para armazenar os seguintes dados de um talonário após a utilização total do mesmo: n2 do cheque, valor, data e destino. Sabendo-se que o número de cheques pode ser variável e não ultrapassar 20, construa esse algoritmo de tal maneira que possa gerar um relatório no vídeo.

41. Criar um algoritmo para gerenciar um sistema de reservas de mesas em uma casa de espetáculo. A casa possui 30 mesas de 5 lugares cada. O algoritmo deverá permitir que o usuário escolha um código de uma mesa (100 a 129) e forneça a quantidade de lugares desejados. O algoritmo deverá informar se foi possível realizar a reserva e atualizar a reserva. Se não for possível, o algoritmo deverá emitir uma mensagem. O algoritmo deve terminar quando o usuário digitar o código O (zero) para uma mesa ou quando todos os 150 lugares estiverem ocupados.

42. Gere e imprima uma matriz M 4x4 com valores aleatórios entre 0-9. Após isso determine o maior número da matriz e a sua posição (linha, coluna).

43. Gere e imprima uma matriz M 10x10 com valores aleatórios entre 0-9. Após isso indique qual é o maior e o menor valor da linha 5 e qual é o maior e o menor valor da coluna 7.

44. Capture do teclado valor para preenchimento de uma matriz M 3x3. Após a captura imprima a matriz criada e encontre a quantidade de números pares, a quantidade de números ímpares.

45. Faça um programa para armazenar em uma matriz os compromissos de uma agenda pessoal. Cada dia do mês deve conter 24 horas, onde para cada uma destas 24 horas podemos associar uma tarefa específica (compromisso agendado). O programa deve ter um menu onde o usuário indica o dia do mês que deseja alterar e a hora, entrando em seguida com o compromisso, ou então, o usuário pode também consultar a agenda, fornecendo o dia e a hora para obter o compromisso armazenado.

46. Modifique o programa anterior de maneira a guardar os compromissos de todo o ano, organizados por mês, dia e hora (só 8 horas por dia).

47. Faça um programa para jogar o jogo da velha. O programa deve permitir que dois jogadores façam uma partida do jogo da velha, usando o computador para ver o tabuleiro. Cada jogador vai alternadamente informando a posição onde deseja colocar a sua peça (‘O’ ou ‘X’). O programa deve impedir jogadas inválidas e determinar automaticamente quando o jogo terminou e quem foi o vencedor (jogador1 ou jogador2). A cada nova jogada, o programa deve atualizar a situação do tabuleiro na tela.

48. Criar um vetor A com 10 elementos inteiros. Implementar um programa que defina e escreva a quantidade de elementos armazenados neste vetor que são pares.

49. Criar um vetor A com 10 elementos inteiros. Implementar um programa que defina e escreva a soma de todos os elementos armazenados neste vetor.

50. Criar um vetor A com 10 elementos inteiros. Implementar um programa que determine a soma dos elementos armazenados neste vetor que são múltiplos de 5.

# MÓDULO 10:  PROJETO S

**Na programação, o caminho mais curto entre A e B é o algoritmo.**
Daiana Araújo Martins

**A programação começa fora do computador.**
Ayrton Yagami

- ESTUDO DE CASO
- CÓDIGO LIMPO
- PRÓXIMOS PASSOS
- PROJETOS
- FEEDBACK DO AUTOR

# Objetivos

- Consolidar os conceitos aprendidos em projetos práticos.
- Demonstrar a importância de escrever código limpo e bem estruturado.
- Apresentar exemplos de projetos aplicáveis no mundo real.

# Estudo de caso

Antes de mergulhar em projetos práticos, examinaremos um estudo de caso. Vamos criar um programa simples de **Gestão de Tarefas Pessoais**.

## Descrição do Problema

Imagine que você deseja organizar melhor suas tarefas diárias. Para isso, o programa deverá:

1. Permitir o cadastro de tarefas com título, descrição e prazo.
2. Marcar tarefas como concluídas.
3. Listar todas as tarefas cadastradas.

Implementação em Java
Abaixo está uma abordagem básica para resolver o problema:

**OBS**: Para resolver problemas desse gênero usamos conceitos de programação Orientada a Objetos, assunto esse que não abordamos aqui neste livro, mas fica de lição para o leitor buscar informações sobre o assunto.

```
import java.util.ArrayList;
import java.util.Scanner;

class Tarefa {
    String titulo;
    String descricao;
    boolean concluida;

    Tarefa(String titulo, String descricao) {
    this.titulo = titulo;
    this.descricao = descricao;
    this.concluida = false;
    }

    void concluir() {
    this.concluida = true;
    }

    @Override
    public String toString() {
    return "Tarefa: " + titulo + " | Descrição: " +
descricao + " | Concluída: " + (concluida ? "Sim" :
"Não");
    }
}

public class GestorDeTarefas {
    public static void main(String[] args) {
    ArrayList<Tarefa> tarefas = new ArrayList<>();
    Scanner scanner = new Scanner(System.in);
```

```
    while (true) {
     System.out.println("\n1. Adicionar tarefa\n2.
Listar tarefas\n3. Concluir tarefa\n4. Sair");
        int opcao = scanner.nextInt();
        scanner.nextLine(); // Limpa o buffer

        if (opcao == 1) {
           System.out.println("Título da tarefa:");
            String titulo = scanner.nextLine();
                  System.out.println("Descrição  da
tarefa:");
            String descricao = scanner.nextLine();
                   tarefas.add(new  Tarefa(titulo,
descricao));
            System.out.println("Tarefa  adicionada
com sucesso!");
        } else if (opcao == 2) {
                    System.out.println("\nLista  de
Tarefas:");
            for  (int  i  =  0;  i  <  tarefas.size();
i++) {
                System.out.println(i + 1 + ". " +
tarefas.get(i));
            }
        } else if (opcao == 3) {
            System.out.println("Digite o número da
tarefa a concluir:");
            int indice = scanner.nextInt() - 1;
                  if  (indice  >=  0  &&  indice  <
tarefas.size()) {
                tarefas.get(indice).concluir();
```

```
            System.out.println("Tarefa concluída!");
              } else {
                              System.out.println("Tarefa
inválida.");
              }
          } else if (opcao == 4) {
              break;
          } else {
              System.out.println("Opção inválida.");
          }
      }
      scanner.close();
      }

}
```

# Código Limpo

Tu te tornas eternamente responsável por aquilo que... programas. Ok, a frase de O Pequeno Príncipe não é bem assim, mas se você é programador, sabe o quanto é importante cuidar dos códigos que escreve. Por meio do clean code, ou “código limpo”, é possível programar bons códigos para facilitar a sua leitura e torná-lo claro, simples e escalável, a fim de que alcance os objetivos do cliente. É claro que isso nem sempre é simples, ainda mais em um mercado onde tudo muda o tempo todo e novas maneiras de escrever códigos aparecem a cada dia. Para te ajudar, hoje vamos falar sobre clean code. Quais são as boas práticas que você deve se atentar ao programar? Confira no nosso artigo!

## O que é clean code?

Se você, como muitos desenvolvedores, têm o livro Código Limpo, de Robert Cecil Martin (o Uncle Bob), na cabeceira, sabe que o conceito de “clean code” é amplo e muitos especialistas têm suas próprias definições e até discordam entre si.

Grady Booch, Cientista-chefe de Engenharia de Software da IBM Research, por exemplo, é citado pelo autor por considerar que:

“O código limpo é simples e direto. Código limpo parece uma prosa bem escrita. Código limpo nunca obscurece a intenção do designer, mas é completo de abstrações nítidas e linhas simples de controle.”

Já Michael Feathers, autor de "Working Effectively with Legacy Code", coloca a palavra “cuidado” como central para um código limpo.

Técnica para facilitar a escrita e leitura de um código
Apesar das diferentes definições, é possível unir diferentes significados e dizer que clean code é aplicar determinadas técnicas para facilitar a escrita e leitura de um código.

Isso porque um software ou um sistema nunca está totalmente concluído, ele sempre sofrerá atualizações e receberá novas funcionalidades, o que torna importante que ele seja o mais limpo possível.

## Por que clean code é importante?

É preciso entender que, ao produzir um software para um negócio, a própria empresa passará por mudanças. Os requisitos do negócio mudam, as necessidades do usuário mudam e, eventualmente, o código que você escreveu irá precisar evoluir.

E outra pessoa precisará ler o seu código, entendê-lo e mudá-lo ou até corrigi-lo. Clean code pressupõe escrever códigos com alta legibilidade, o que leva a uma boa manutenção. Seguindo o livro de Uncle Bob (tradução livre):

> “Se você é programador há mais de dois ou três anos, provavelmente ficou lento devido a um código confuso. O grau de desaceleração pode ser significativo. [...] Com o tempo, a bagunça se torna tão grande, profunda e alta que eles não conseguem limpá-la.”

O que um clean code deve ser?
Para que isso não aconteça, é importante seguir alguns princípios ao desenvolver sistemas. O clean code deve ser:


- **Simples e direto:** seguindo o princípio de KISS (Keep It Simple Stupid), o código bem escrito deve ter o mínimo de complexidade possível, para que possa ser facilmente depurado e compreendido.

- **Seco:** o código DRY "Don't Repeat It" – conceito trazido por Andy Hunt, um dos autores do Manifesto Ágil -, é aquele sem ambiguidade, ou seja, se você já o inseriu em algum lugar no código-fonte, ele não deve ser implementado novamente.

- **Eficiente:** um código é programado para atender a algum objetivo específico, então procure se certificar que ele funcionará da forma que foi proposto.

- **Elegante:** Bjarne Stroustrup, inventor do C++, diz que gosta dos seus códigos elegantes e que, quando você o lê, deve se sentir feliz. A ideia da elegância é tornar o código diferente dos demais.

- **Atenção aos detalhes:** o programador deve sempre criar um código de forma cuidadosa, já que um código mal escrito desde o início impactará em sua manutenção, trazendo grandes prejuízos e lentidão para entendê-lo.

- **Atenção aos comentários:** comente o mínimo possível em seu código. O código já deve ser claro o suficiente para não necessitar de comentários. Caso necessite, tente comentar o mínimo possível.

Clean code: Testes de programação

Diante dessas regras, é sempre importante se lembrar que o código escrito por você passará, provavelmente, por manutenções e refatoração. O código só será considerado limpo se ele passar por testes.

Ao produzir um software, portanto, é preciso garantir que cada linha do código seja validada, para que se mantenha em funcionamento.

É interessante utilizar a metodologia de Desenvolvimento Orientado a Testes (TDD), e realizar testes de Integridade, Instalação, Configuração, Usabilidade, Segurança, Integração e outros.

Como escrever um bom código?

Diante disso, quais as práticas que devem ser seguidas para que o clean code seja possível?

- **Defina bons nomes:** o nome é essencial para o código. Deve ser direto e representar bem o que ele significa, mesmo que isso pressuponha um nome extenso.

- **Nomes de Classes:** evite palavras conflitantes com os componentes da linguagem e use substantivos em vez de verbos.

- **Nomes de Métodos:** aqui, sim, deve conter verbos e exprimir a intenção do desenvolvedor, como “ExcluirPagina”.

- **TDD:** como falamos, é preciso testar tudo, o tempo todo.

- **Notação húngara não agrega valor:** atualmente, entende-se que a notação húngara (uso do tipo da variável logo após o seu nome) polui a leitura e existem novas linguagens de programação mais adequadas.

- **Bons nomes dispensam comentários:** substitua-os por nomes claros, para que não haja mais confusão.

- **Crie tratamento de erros:** caso haja alguma falha, é preciso garantir formas de tratá-la.

- **Formatação:** cuide da formatação e da indentação do código para melhorar sua legibilidade.

- **Pacote ou package:** categorize os códigos em pacotes para que fiquem mais organizados e elegantes.

## Como Criar Nomes Significativos?

Use nomes que revelem seu propósito. O nome de uma variável, função ou classe deve responder a todas as grandes questões. Ele deve dizer porque existe, o que faz e como é usado.

Evite informações erradas. Os programadores devem evitar passar dicas falsas que confundam o sentido do código. Devemos evitar palavras cujos significados podem se desviar daquele que desejamos.

Faça distinções significativas. Faça a distinção dos nomes de forma a que o leitor compreenda as diferenças.

Use nomes pronunciáveis. Um exemplo de um nome não pronunciável é, por exemplo, “private Date genymdhms”.

Use nomes passíveis de busca. Nomes longos se sobressaem aos pequenos e qualquer nome passível de busca se sobressai a uma constante no código. Se usar nomes pequenos, que eles sejam claros.

Atente nos prefixos de variáveis. Quanto mais lemos o código, menos prefixos enxergamos. No final, estes se tornam partes invisíveis é um indicativo de código velho.

Evite o mapeamento mental. Uma diferença entre um programador esperto e um programador profissional é que este entende que clareza é fundamental. Os profissionais se preocupam em escrever código que outros possam entender, por outras palavras, código limpo.

Escolha bem os nomes de classes. Classes e objetos devem ter nomes com substantivo(s).

Cuide dos seus nomes de métodos. Os nomes de métodos devem ter verbos.

Opte pela clareza no lugar do divertimento. Os programadores saberão o que deve fazer a função HolyHandsGrenade? Claro, é engraçado, mas talvez Delete Items seja mais fácil de entender.

Selecione uma palavra por conceito. Um léxico consistente é uma grande vantagem para os programadores que precisam usar seu código.

Não faça trocadilhos. Evite usar a mesma palavra para dois propósitos. Usar o mesmo termo para duas ideias diferentes é basicamente um trocadilho.

Adicione um contexto significativo. Há poucos nomes que são significativos por si só – a maioria não é. Por conta disso, você precisa usar nomes que façam parte do contexto para o leitor. Para isso você os coloca em classes, funções e namespaces bem nomeados.

# Desenvolvimento de software

Desde que os softwares se tornaram itens valiosos para as empresas, começou-se a pensar em um processo estruturado para desenvolvê-los. Isso poderia ser uma tarefa da própria equipe interna ou poderia ser tercerizado.
Quando é terceirizado, geralmente as empresas entram em contato com companhias especializadas, como fábricas de software, e conseguem uma qualidade incrível.
O desenvolvimento de software só ganhou camadas com o tempo. Se tornou uma atividade multidisciplinar e complexa, ao mesmo tempo em que se tornou mais ágil por conta da automação, do reuso e das informações que se encontram na internet, diante disso, as empresas precisam saber o que fazer quando tem uma demanda de desenvolvimento de software.

## Desenvolvimento de Software

O desenvolvimento de software é um conjunto de processos que visam coletar requisitos para transformá-los em um produto final, um software utilizável e eficiente. Esse produto visa resolver um ou mais problemas, especificados pelos requisitos, com funcionalidades principais.
A noção de software mudou muito com o tempo. Antigamente, poderíamos dizer que se tratava de uma tela simples, com comandos, botões e informações visuais. Hoje, temos aplicações robustas e complexas, interativas e dinâmicas, cada vez mais inovadoras em termos de usabilidade e experiência. Aliás, é bom guardar essa palavra: experiência.  Atualmente, a preocupação não é só atender a requisitos, mas pensar no usuário. Pensar em como criar algo que vai fazer o usuário se sentir bem, satisfeito, enquanto usa. Esse é um dos indicadores de sucesso de um sistema, Não vale apenas criar um

sistema com os botões certos e as funcionalidades adequadas. É preciso pensar no tempo que o usuário leva para conseguir fazer algo, na agilidade, na limpeza e clareza das informações na interface e no quão efetivamente o sistema se encaixa no dia a dia do seu usuário.

Esse usuário pode ser uma pessoa que compra o software para uso pessoal ou profissional. Como também pode ser um membro de uma equipe corporativa adotando o sistema para lidar com demandas diárias, então, o desenvolvimento de software de hoje pensa também no usuário, entendê-lo. Isso faz parte da interdisciplinaridade da área, algo que a torna pronta para o mundo dinâmico que temos.

## Tipos de software

Para avançar nesse assunto, vamos especificar os tipos de software. Trabalharemos com as principais terminologias a fim de esclarecer as diferenças.

## Aplicativo

Geralmente quando se fala em aplicativo, o objetivo é definir um sistema criado especificamente para celulares ou tablets — telas menores, em geral. Claro, a terminologia pode ser aplicada em outros casos, mas, no popular, app é sinônimo de mobile.

Basicamente, são aplicações que requerem o mesmo nível de complexidade, só que com uma interface mais simples e responsiva, pensada para telas menores. Além disso, há também o cuidado com a integração com o dispositivo, de modo a assegurar o máximo de eficácia na usabilidade.

Apps tendem a pensar ainda mais na experiência do usuário. Hoje, a área de UX foca bastante em criar coisas responsivas, prontas para rodar bem em qualquer aparelho.

Como as pessoas passam muito tempo com os celulares, é comum investir um pouco mais para conquistar as pessoas. Exemplos são:

apps de bancos digitais, apps para controle de produtividade, apps para personalizar a tela do aparelho, etc.

## Programa

Um programa é uma aplicação geral, desenvolvida tanto para desktop quanto para mobile. Também inclui aplicações web. “Programa” é uma definição mais genérica e até antiga para designar sistemas criados com requisitos, a partir de uma metodologia de desenvolvimento.

## Sistema

Já o termo sistema é algo mais comum na bibliografia e em contextos acadêmicos. Trata-se de uma aplicação criada para qualquer tipo de dispositivo ou propósito. É simplesmente o produto do desenvolvimento que cumpre alguma função e processa dados.
Inclusive, temos o termo “sistema de informação” para determinar uma aplicação que realiza uma função específica dentro de um ambiente sistêmico, como uma empresa ou uma universidade.
Exemplos: sistema de gestão de uma empresa ou sistema de controle de alunos em uma faculdade.

## Gerenciar projetos de software

O que é necessário para desenvolver um software, primeiro, vamos listar e depois explicar.

Em suma:

- Requisitos;
- Metodologia;
- Custos e tempo;
- Gestão de projeto;
- Uma equipe interdisciplinar.

Primeiro, tudo começa com os requisitos. São as funcionalidades solicitadas pelo cliente ou contratante, que especificam o que é preciso garantir no software. Ou seja, em um sistema de gestão de alunos em uma escola, um requisito é o cadastro desses alunos. Outro pode ser a exclusão de alunos que saíram.

Depois, temos a metodologia. Trata-se de uma forma de desenvolver o software e levantar o produto.

Pode ser cascata, com etapas rígidas que se sucedem, com muita documentação e pouco contato com o cliente; ou ágil, com etapas menores e mais flexíveis, menos documentação, versões utilizáveis do produto sendo lançadas constantemente e colaboração constante com os clientes, evidentemente, é preciso empreender custos e tempo para um projeto de software. Não é algo simples.

Por isso, também é necessária uma boa gestão de projetos, com gestão dos gastos, do prazo, do orçamento, dos riscos, das necessidades, etc. Tudo deve ser analisado com cuidado e com o apoio de ferramentas tecnológicas. Por fim, é fundamental ter uma equipe que ajude a criar o sistema.

O projeto pode envolver pessoas especialistas em banco de dados, programadores, designers, gestores de projetos e coordenadores.

## Funcionamento do Desenvolvimento de Software

Neste tópico, vamos dissecar como funciona o desenvolvimento de software, com a análise de cada uma das etapas.

## Rascunho do projeto

Tudo começa com o protótipo do produto, ou um rascunho. Nesse primeiro momento, o foco é estabelecer o objetivo do sistema e os requisitos principais que deverão ser transformados em funcionalidades. É uma fase de compreensão e entendimento da proposta.

## Definição da linguagem

Então, é hora de escolher as tecnologias que serão utilizadas como base para o desenvolvimento de software. Essas linguagens variam bastante a depender do que se procura, do ambiente no qual o sistema vai rodar.

Por exemplo, em uma aplicação web, as tecnologias devem incluir HTML, CSS, uma linguagem de programação de front-end e uma de back-end. Além disso, também é crucial escolher a linguagem de banco de dados e outras tecnologias que vão ajudar no processo.

## Análise de requisitos

Depois, temos uma fase mais profunda de análise dos requisitos especificados. É preciso olhar para o que foi solicitado e pensar em como transformar isso em uma versão prática, utilizável. Também é o momento de filtrar alguns requisitos que não poderão ser implementados.

## Projeto

O projeto é a fase de levantar uma especificação mais detalhada e próxima do que é o produto de fato. Geralmente, é o momento de documentar o sistema e criar os relatórios que vão ajudar a compreender o produto. Esses documentos ajudam até em questão de manutenção posteriormente. Inclui também a modelagem do sistema, com modelos que auxiliam na compreensão das funcionalidades do sistema como um todo e permitem visualizar isso com representação gráfica.

## Desenvolvimento propriamente dito

Esse é o momento de criar o produto de fato com as linguagens escolhidas. É a parte da codificação do sistema, de acordo com os requisitos, com o projeto e com as indicações do cliente.

## Teste

Logo depois, temos a fase de teste, que visa garantir que o sistema está bom, sólido e confiável.

## Implementação

Depois, temos o momento de implementar o produto, isto é, lançar o produto para o ambiente em que será utilizado pelos clientes.

## Desenvolver Software Para Uma Empresas

Caso você tenha uma demanda de produção de software em sua empresa, há dois caminhos.

Um deles é a criação da própria equipe. Isso facilita a gestão e permite assegurar um bom nível de personalização.

Porém é importante ter pessoas no time que possuam as competências necessárias para que o software atenda os requisitos técnicos, de segurança, de usabilidade, entre outros.

Quando não há, é necessário contar com um parceiro que tenha o foco em desenvolvimento de software e equipe que possa complementar esses perfis.

Portanto, o outro caminho é melhor: contratar uma equipe especializada. A possibilidade de personalização é ainda mais avançada, com o uso dos melhores recursos e das melhores estratégias.

Você garante um desenvolvimento seguro e eficaz que vai atender ao que você procura e precisa. Na maioria das vezes essa demanda é temporária e ao final é necessário desmobilizar a equipe. Dessa forma

não haverá gastos e nem esforço de contratação e treinamento de um time interno para isso além de todo trabalho com a desmobilização do mesmo.

# Próximos Passos

Após concluir este módulo, você pode:

- Estudar as Bases de Programação Orientada a Objetos
- Explorar frameworks Java como Spring ou Hibernate.
- Construir interfaces gráficas usando JavaFX.
- Estudar estruturas de dados e algoritmos avançados.

# Projetos

Aqui estão 15 ideias de projetos em Java para iniciantes, cada uma com uma breve explicação sobre o que o projeto envolve e as habilidades que ele ajuda a desenvolver:

## 1. Calculadora Básica

Descrição: Uma calculadora que realiza operações matemáticas simples (adição, subtração, multiplicação e divisão).
Aprendizado: Variáveis, operadores aritméticos e controle de fluxo.
Desafios: Implementar verificações de erros (como divisão por zero) e usar estruturas de repetição para fazer vários cálculos.

## 2. Lista de Tarefas

Descrição: Um aplicativo que permite adicionar, remover e visualizar tarefas.
Aprendizado: Manipulação de listas (`ArrayList`), manipulação de strings e estrutura de dados.
Desafios: Implementar persistência de dados usando arquivos para salvar a lista de tarefas.

## 3. Conversor de Moedas

Descrição: Um conversor que converte valores de uma moeda para outra com base em taxas de câmbio.
Aprendizado: Manipulação de variáveis, entrada e saída de dados, e matemática básica.
Desafios: Configurar taxas de câmbio dinâmicas e atualizar as taxas periodicamente.

## 4. Jogo da Adivinhação

Descrição: Um jogo simples onde o programa gera um número aleatório, e o usuário deve adivinhar qual é.
Aprendizado: Uso da classe `Random`, controle de fluxo (`if`, `while`) e manipulação de entrada/saída.
Desafios: Oferecer dicas para o jogador e implementar um contador de tentativas.

## 5. Sistema de Gerenciamento de Contatos

Descrição: Um programa que permite armazenar e visualizar informações de contatos.
Aprendizado: Manipulação de listas e objetos, criação de classes, encapsulamento e uso de métodos.
Desafios: Implementar uma interface de busca e salvar os dados em arquivos para persistência.

## 6. Jogo da Forca

Descrição: O usuário tenta adivinhar uma palavra escolhendo letras.
Aprendizado: Controle de fluxo, manipulação de strings e listas.
Desafios: Gerar uma palavra aleatória, mostrar o progresso ao usuário e controlar o número de tentativas.

## 7. Calculadora de IMC

Descrição: Calcula o Índice de Massa Corporal (IMC) com base no peso e altura do usuário.
Aprendizado: Entrada de dados, cálculos simples e instruções condicionais.
Desafios: Exibir o resultado com base na classificação do IMC.

## 8. Sistema de Gerenciamento de Biblioteca

Descrição: Gerencia o empréstimo e devolução de livros.

Aprendizado: Manipulação de listas e arrays, estrutura de dados e criação de classes.

Desafios: Implementar uma função para verificar se o livro já foi emprestado e um sistema de penalidades para devoluções atrasadas.

## 9. Sistema de Vendas Simples

Descrição: Gerencia a venda de produtos com um sistema de inventário básico.

Aprendizado: Manipulação de listas, encapsulamento e métodos.

Desafios: Implementar um sistema de estoque que atualize o número de produtos disponíveis após cada venda.

## 10. Conversor de Temperatura

Descrição: Converte valores entre Celsius, Fahrenheit e Kelvin.

Aprendizado: Estrutura de controle, cálculos básicos e manipulação de entrada/saída.

Desafios: Permitir múltiplas conversões consecutivas e incluir todas as fórmulas de conversão.

## 11. Simulador de Caixa Eletrônico (ATM)

Descrição: Um sistema que simula operações bancárias como saldo, saque e depósito.

Aprendizado: Controle de fluxo, manipulação de variáveis e operações matemáticas.

Desafios: Implementar verificações de saldo, fornecer recibos de transação e exibir uma interface amigável.

## 12. Gerador de Senhas Aleatórias

Descrição: Gera senhas aleatórias de diferentes tamanhos e complexidade (letras, números, caracteres especiais).
Aprendizado: Manipulação de strings, controle de fluxo e aleatoriedade.
Desafios: Permitir ao usuário definir o comprimento e o nível de segurança da senha.

## 13. Agenda Telefônica Simples

Descrição: Armazena e exibe contatos telefônicos.
Aprendizado: Criação de classes, arrays/listas, encapsulamento.
Desafios: Implementar busca por nome e número, permitir edição e exclusão de contatos.

## 14. Conversor de Unidades de Comprimento

* Descrição: Converte medidas (em metros, centímetros e polegadas) entre si.
Aprendizado: Operações matemáticas, controle de fluxo e manipulação de dados.
Desafios: Adicionar outras unidades de medida e garantir precisão nas conversões.

## 15. Quiz de Perguntas e Respostas

Descrição: Um jogo de perguntas e respostas onde o usuário tenta acertar o máximo possível.
Aprendizado: Controle de fluxo, manipulação de strings, criação de métodos.
Desafios: Contabilizar a pontuação do usuário e fornecer um feedback no final do jogo.

Exemplos de Expansão para Projetos Avançados

Para cada um desses projetos, você pode expandir suas funcionalidades à medida que adquire mais conhecimento:

1. Adicionar interface gráfica (usando JavaFX ou Swing) para melhorar a experiência do usuário.
2. Implementar persistência de dados com banco de dados (MySQL, SQLite) para armazenar dados de forma duradoura.
3. Adicionar um sistema de login e autenticação em sistemas de gerenciamento para aumentar a segurança.
4. Conectar a APIs externas, como APIs de câmbio em um conversor de moedas, para tornar o projeto mais dinâmico e útil.

Esses projetos ajudam a desenvolver habilidades fundamentais em Java,
como manipulação de dados, controle de fluxo, encapsulamento, e uso de bibliotecas padrão da linguagem.

# Depuração de código

Depuração corresponde ao processo de localizar e corrigir erros ou bugs no código-fonte de qualquer software. Quando o software não funciona conforme o esperado, os programadores de computadores estudam o código para determinar as causas dos erros. Eles usam ferramentas de depuração para executar o software em um ambiente controlado, verificar o código detalhado, analisar e corrigir o problema.

## De onde surgiu o termo “debugging”?

O termo “debugging” (depuração) remonta à almirante Grace Hopper, que trabalhou na Harvard University na década de 1940. Quando um de seus colegas encontrou uma mariposa impedindo o funcionamento de um dos computadores da universidade, ela disse que eles estavam “debugging” o sistema. Os primeiros registros de programadores de computadores usando os termos “bugs” e “debugging” data da década de 1950. No início da década de 1960, esses termos já eram comumente aceitos na comunidade de programação.

## A importância da depuração

Bugs e erros acontecem na programação de computadores porque é uma atividade abstrata e conceitual. Os computadores manipulam dados na forma de sinais eletrônicos. As linguagens de programação abstraem essas informações para que os humanos possam interagir com os computadores de forma mais eficiente. Quaisquer tipos de softwares possuem diversas camadas de abstração com diferentes componentes que se comunicam para que uma aplicação funcione adequadamente. Quando ocorrem erros, encontrar e resolver o problema pode ser um desafio. As ferramentas e estratégias de

depuração ajudam a corrigir problemas mais rapidamente e melhoram a produtividade do desenvolvedor. Como resultado, a qualidade do software e a experiência do usuário final são melhoradas.

# O processo de depuração

O processo de depuração normalmente requer as etapas a seguir.

## Identificação do erro

Desenvolvedores, verificadores e usuários finais relatam bugs que descobriram ao testar ou usar o software. Os desenvolvedores devem localizar a linha exata de códigos ou o módulo de código que está causando o bug. Esse pode ser um processo cansativo e demorado.

## Análise do erro

Os codificadores analisam o erro registrando todas as alterações de estado do programa e os valores de dados. Eles também priorizam as correções de bugs com base em seu impacto na funcionalidade do software. A equipe de software também identifica um cronograma para correções de bugs, com base nas metas e nos requisitos de desenvolvimento.

## Correção e validação

Os desenvolvedores corrigem o bug e executam testes para garantir que o software continue funcionando conforme o esperado. Eles também podem programar novos testes para verificar se o bug se repete no futuro.

## Depuração versus teste

A depuração e o teste são processos complementares que garantem que os programas de software estão funcionando como deveriam. Após

escrever uma seção completa ou a parte de um código, os programadores realizam testes para identificar bugs e erros. Uma vez que os bugs são encontrados, os codificadores podem iniciar o processo de depuração e trabalhar para solucionar quaisquer erros do software.

## Quais são os erros de codificação que requerem depuração?

Os defeitos de software surgem devido à complexidade inerente ao desenvolvimento de softwares. Erros de produção menores também são observados depois que o software está ativo porque os clientes o usam de maneiras inesperadas.

## Erros de sintaxe

Um erro de sintaxe é um bug que ocorre quando um programa de computador tem uma instrução digitada incorretamente. É o equivalente a um erro de digitação ou ortografia no processamento de textos. O programa não será compilado ou executado se houver erros de sintaxe. O software de edição de código normalmente destaca esse erro.

## Erros semânticos

Erros semânticos ocorrem devido ao uso indevido de instruções de programação. Por exemplo, ao traduzir a expressão $\frac{x}{2\pi}$ para Java, você pode escrever:

O que resulta em um erro de cálculo e pode levar a bugs.

```
y = x / (2 * Math.PI);
```

Código completo em Java:
Aqui está como seria um código Java ilustrando o problema e sua solução:

```
public class SemanticErrorExample {
    public static void main(String[] args) {
        double x = 10; // Exemplo de valor para x

        // Expressão com erro
        double yIncorrect = x / 2 * Math.PI;
        System.out.println("Resultado incorreto: " +
yIncorrect);

        // Expressão corrigida
        double yCorrect = x / (2 * Math.PI);
          System.out.println("Resultado correto: " +
yCorrect);
    }
}
```

Ao executar o código, você verá que o resultado da expressão corrigida será o valor correto da fórmula.

## Erros lógicos

Erros lógicos ocorrem quando os programadores interpretam o processo detalhado ou o algoritmo de um programa de computador de forma distorcida. Por exemplo, o código pode sair de uma repetição precocemente ou pode ter um resultado de verificação incorreto. Você

pode identificar erros de lógica executando o código para vários cenários de entrada e saída diferentes.

### Erros de tempo de execução

Erros de tempo de execução ocorrem devido ao ambiente de computação no qual o código do software é executado. Os exemplos incluem espaço de memória insuficiente ou excedente de pilhas. Você pode resolver erros de tempo de execução colocando instruções em blocos “try-catch” ou registrando a exceção com uma mensagem apropriada.

## Algumas estratégias comuns de depuração

Existem diversas estratégias usadas pelos programadores para minimizar erros e reduzir o tempo necessário para a depuração.

### Desenvolvimento de programas suplementares

O desenvolvimento suplementar corresponde a desenvolver programas em seções gerenciáveis para que pequenas partes do código sejam testadas com frequência. Ao fazer isso, os programadores podem localizar quaisquer bugs que encontrarem. Isso também permite que eles trabalhem em um bug de cada vez, ao invés de vários erros, depois de programarem grandes seções de código.

## Rastreamento inverso

O rastreamento inverso é um método popular de depuração, principalmente para programas menores. Os desenvolvedores trabalham de trás para frente com base em locais em que um erro fatal aconteceu para identificar o ponto exato de sua ocorrência no código. Infelizmente, o processo se torna mais desafiador à medida que o número de linhas de código aumenta.

## Depuração remota

A depuração remota corresponde a depuração de uma aplicação que funciona em um ambiente separado de sua máquina local. Por exemplo, você pode usar ferramentas de depuração instaladas remotamente para resolver o bug.

## Registro em log

A maioria dos programas de computador registra dados e outras informações importantes, como tempo de execução e estados do sistema operacional, em arquivos de log internos. Os desenvolvedores estudam arquivos de log para localizar e resolver bugs. Eles também usam ferramentas como analisadores de log para automatizar o processamento de arquivos de log.

## Depuração em nuvem

A depuração de aplicações complexas em nuvem é um desafio porque os desenvolvedores precisam simular arquiteturas de nuvem em máquinas locais. Ao longo do tempo, diferenças de configuração podem surgir entre o ambiente em nuvem e o ambiente simulado. Isso resulta em mais bugs na produção e ciclos de desenvolvimento mais longos. Ferramentas especiais são necessárias para uma depuração em nuvem mais eficiente.

# Feedback do Autor

Gostaria de ouvir sua opinião!

- O que você achou deste livro?
- Há algo que você gostaria de ver em edições futuras?

Envie seu feedback para [contato@edvaldovunge.com].

# Referências bibliográficas

SCHILDT, Herbert - Java The Complete Reference.
Estados Unidos da America: McGraw-Hill Education (Publisher), 2019
Eleventh Edition. ISBN9781260440232

DEITEL, Paul e DEITEL, Harvey - Java: Como Programar.
Estados Unidos da America: Pearson Universidades, 10ª edição (24 junho 2016)
ISBN-13:978-8543004792

Lopes, Anita e GARCIA, Guto - Introdução à programação.
Rio de Janeiro: Elsevier Editora Ltda., 2002.
ISBN 85-352-1019-9

MANZANO, José Augusto N. G., Estudo Dirigido: ALGORITMOS - Editora Érica, 2000.

V. Anton Spraul - Think Like a Programmer - An Introduction to Creative Problem Solving Editor: NO STARCH PRESS,US Data de Lançamento: agosto de 2012
ISBN:9781593274245

# Referências Online

Convenção para a nomenclatura - https://google.github.io/styleguide/javaguide.html#s5-naming

https://www.synopsys.com/glossary/what-is-moores-law.html

https://www.ibm.com/docs/pt/rsas/7.5.0

# Glossário de termos técnicos

## A

- **Algoritmos:** Sequências lógicas de instruções para resolver problemas.
- **Ambiente de Desenvolvimento Integrado:** Ferramentas que facilitam o desenvolvimento de software.
- **Análise de dados:** Processamento e interpretação de grandes volumes de informação.
- **Aplicações Informáticas:** Programas utilizados em computadores para tarefas específicas.
- **Armazenamento Virtual:** Tecnologia para guardar dados remotamente, geralmente na nuvem.
- **ASCII:** Código padrão de representação de caracteres em formato digital.
- **Assembly:** Linguagem de programação de baixo nível próxima ao código de máquina.

## B

- **BASIC:** Linguagem de programação fácil de aprender, criada nos anos 1960.
- **Boas práticas de programação:** Conjunto de normas para escrever código claro e eficiente.
- **Buffer do teclado:** Área de memória temporária para armazenar entradas do teclado.
- **Bugs:** Erros ou falhas em programas de software.
- **Bytecode:** Código intermediário executado por máquinas virtuais, como JVM.

# C

- **C:** Linguagem de programação estruturada e amplamente utilizada.
- **C++:** Extensão da linguagem C, com suporte a orientação a objetos.
- **Capacidade de processamento:** Quantidade de operações que um computador realiza em um período.
- **Caracteres Unicode:** Sistema de codificação para representar texto em diversos idiomas.
- **Classe Math:** Biblioteca de funções matemáticas em linguagens como Java.
- **Compilador:** Ferramenta que converte código-fonte em linguagem de máquina.
- **Computação:** Estudo e aplicação de sistemas computacionais.
- **Computação Cliente/Servidor:** Arquitetura onde clientes acessam serviços em servidores.
- **Computação em nuvem:** Tecnologia para acessar recursos computacionais remotamente.
- **Computação paralela e distribuída:** Processamento simultâneo em múltiplos núcleos ou máquinas.
- **Computador:** Máquina eletrônica para processar, armazenar e exibir dados.
- **Computador científico:** Projetado para cálculos intensivos em ciência e engenharia.
- **Computador eletrônico:** Computadores que usam circuitos eletrônicos para operar.
- **Computador pessoal (PC):** Computador destinado a uso individual.
- **Constants:** Valores imutáveis em programação.
- **Construção de Software:** Processo de criação de programas de computador.
- **Construtor:** Método especial de inicialização de objetos em POO.
- **Containers de dados:** Estruturas que armazenam coleções de informações, como listas.
- **Conversão de tipo:** Mudança entre diferentes tipos de dados em programação.

- **CPU (Unidade Central de Processamento):** O cérebro do computador realiza cálculos e executa instruções.

## D

- **Dados:** Informações brutas processadas por computadores.
- **Desenvolvimento de software:** Processo de criação, teste e manutenção de programas.
- **Dispositivos embarcados:** Sistemas computacionais integrados a dispositivos maiores.
- **Download:** Transferência de arquivos de um servidor para um dispositivo local.

## E

- **Editores de Código Java:** Ferramentas para escrever e depurar programas em Java.
- **Erlang:** Linguagem funcional usada em sistemas distribuídos e de alta concorrência.
- **Evolução dos computadores:** Progresso tecnológico desde os primeiros modelos mecânicos até os sistemas modernos.

## F

- **Float (Ponto Flutuante):** Representação numérica que suporta valores decimais.

## G

- **Gerenciamento de memória:** Alocação e liberação de memória em programas.
- **Garbage Collection:** Processo automático de liberar memória não usada.

## H

- **Hardware:** Parte física dos computadores, como CPU e memória.
- **Haskell:** Linguagem funcional com foco em segurança e modularidade.

## I

- **Informática:** Estudo de tecnologias para processar informações.
- **Instruções do algoritmo:** Passos que formam um algoritmo.
- **Inteligência Artificial (IA):** Área que cria sistemas capazes de executar tarefas inteligentes.
- **Interpretador:** Programa que executa código fonte diretamente.

## J

- **JavaScript:** Linguagem de programação usada para desenvolvimento web.
- **JVM (Java Virtual Machine):** Máquina virtual que executa bytecode Java.

## L

- **Lua:** Linguagem de programação leve, projetada para scripts.
- **Linguagem de baixo nível:** Próxima ao hardware, como Assembly.
- **Linguagem Estruturada:** Baseada em blocos claros de instruções, como C.
- **Linguagem Hexadecimal:** Sistema de numeração base 16, usado em programação.
- **Linguagem Oak:** Versão inicial da linguagem Java.

## M

- **Memória:** Armazenamento temporário ou permanente em computadores.
- **Método:** Função associada a uma classe ou objeto.
- **Método da classe:** Função pertencente a uma classe em programação orientada a objetos.

## N

- **Null:** Representa a ausência de valor ou referência em programação.

# O

- **Objeto da classe:** Instância de uma classe em POO.
- **Open-source:** Software com código-fonte disponível publicamente.
- **Orientada a objetos:** Paradigma baseado em objetos e classes.

# P

- **Paradigma declarativo:** Específica o que deve ser feito, não como.
- **Paradigma funcional:** Baseado em funções matemáticas.
- **Paradigma imperativo:** Programação com instruções sequenciais.
- **Paradigma lógico:** Baseado em lógica formal, como Prolog.
- **Paradigma orientado a objetos:** Organização em torno de objetos que interagem.
- **Pensamento computacional:** Habilidade de resolver problemas com soluções computáveis.
- **PHP:** Linguagem de script usada principalmente em desenvolvimento web.
- **Portugol:** Linguagem de programação educacional baseada no português.
- **Programadores:** Profissionais que escrevem e mantêm código.
- **Pseudocódigo:** Representação textual de algoritmos.
- **Python:** Linguagem de programação versátil e popular.

# R

- **Recursos computacionais:** Elementos como CPU, memória e dispositivos de entrada/saída.
- **Redes neurais:** Sistemas computacionais que imitam a estrutura do cérebro.

# S

- **Semântica:** Estudo dos significados das instruções de programação.
- **Sintaxe:** Regras que definem a estrutura do código.

- **Smartphones conectados:** Dispositivos móveis com acesso à internet.
- **Software:** Conjunto de instruções executadas por hardware.
- **Software de aplicativo:** Programas para usuários finais.
- **Software de sistema:** Programas que suportam hardware, como sistemas operacionais.

## T

- **Tecnologia de informação e comunicação:** Convergência de tecnologias digitais para comunicação e processamento.
- **Tecnologias de desenvolvimento:** Ferramentas e metodologias para criar software.
- **Touch screen:** Tela sensível ao toque.
- **Transistores:** Componentes que substituíram válvulas em computadores eletrônicos.

## U

- **Upload:** Envio de arquivos de um dispositivo local para a internet.

## V

- **Variáveis:** Elementos que armazenam valores em programação.
- **Variáveis estáticas:** Associadas a uma classe, não a uma instância.

## W

- **Workstation:** Computador potente para uso técnico ou profissional.

BIBLIOGRAFIA DO AUTOR.