JAVA

O Java foi desenvolvido em 1991 na Sun MicroSystem, por uma equipe de engenheiros liderados por Patrick Naughton, Mike Sheridan, James Gosling. Chefe do projeto. Sun Fellow era um título

Com o intuito de criar uma linguagem que fosse capaz de funcionar nos mais diversos dispositivos!

Na época, a tecnologia não estava preparada para esta conectividade entre aparelhos
Porém o projeto não foi a frente, a ideia principal do Java era que todos os aparelhos eletrônicos de consumo se comunicassem entre si. Ex: você poderia deixar sua cafeteira ligando e no ônibus poderia enviar uma mensagem para o microondas para soar o alarme!

Como perceberam que não poderiam ficar presos aos sistemas operacionais, até porque os clientes não estavam interessados no tipo de processador que estavam utilizando, e sim na tecnologia.

Daí! nasceu o projeto GREEN, tendo o surgimento da linguagem OAK

Em 1993, apareceu uma oportunidade para o grupo GREEN a empresa FirstPerson junto com a Time-Warner estavam pedindo propostas de sistemas operacionais de decodificadores e tecnologias de vídeo sob-demanda

Foi na época em que o NCSA apresentou o Mosaic 1.0, o primeiro navegador gráfico popular para WEB. Então a empresa FIRSTPERSON apostou nos testes da TV da TIME-WARNER, mas esta empresa acabou escolhendo a tecnologia oferecida pela Silicon Graphics

Em 1995 a Sun viu uma oportunidade na WEB, nessa época as páginas não existia muita interatividade, apenas conteúdos estáticos eram exibidos. Então nesse ano a SUN anunciou o ambiente JAVA, sendo uma aposta excesso, gerando uma reação nos navegadores populares como o Netscape Navigator e diversos fabricantes como o VRML (Virtual Reality Modeling Language)

A Oracle adquiriu o Java em 2008. Desde então, é uma referência no mundo de desenvolvimento de sistemas

Porém existem outros fatores representados por características extremamente relevantes

- O Java é uma plataforma de desenvolvimento com ferramentas, bibliotecas, ambiente de execuções e etc

Principal

- **Simples:** possui regras muito bem definidas que facilitam o entendimento do código, com uma sintaxe clara e sem a necessidade de manipular ponteiros (alocar memória da máquina para armazenar informações).
- **Robusto:** Java foi desenhado para construir sistemas confiáveis, o que reduz imprevistos em tempo de execução.
- **Seguro:** Java é uma plataforma que proporciona um ambiente seguro para desenvolvimento, no qual a memória é gerenciada automaticamente e fornece canais de comunicação seguros, protegendo a privacidade dos dados.
- **Alto desempenho:** o código Java é compilado para ser convertido em bytecodes, que são interpretados em um ambiente de execução chamado JVM (Java Virtual Machine). Se for necessário mais desempenho, esse ambiente de execução transforma os bytecodes em código de máquina nativo para ganhar desempenho, além de ser multi-thread, permitindo executar várias tarefas simultaneamente dentro do mesmo programa.
- **Orientado a objetos:** o paradigma da programação orientada a objetos (POO) é uma das mais difundidas e utilizadas atualmente. Ela permite trabalhar com um modelo similar ao mundo real, onde os "componentes" dos softwares são objetos que podem ser reutilizados em vários pontos da aplicação, assim como os objetos da vida real.

Ponteiros!

É uma variável que armazena o endereço de outra variável na memória do computador.

Validações

1.

Em Java, não existem **ponteiros** no mesmo sentido que em linguagens como C ou C++. Java não expõe ponteiros diretamente ao programador por motivos de segurança e simplicidade. No entanto, é possível entender a ideia de "ponteiro" de maneira conceitual ao estudar como Java lida com referências.

Ponteiros vs. Referências

- **Ponteiros:** São variáveis que armazenam o endereço de memória de outra variável. Em linguagens como C, você pode manipular diretamente esses endereços, o que pode levar a erros perigosos como acesso a memória inválida.
- **Referências em Java:** Em Java, quando você trabalha com objetos, na verdade está manipulando **referências** a esses objetos, não os objetos diretamente. Uma referência é um identificador que aponta para a localização do objeto na memória, mas você não tem acesso direto ao endereço de memória como em C.

2.

Como funciona em Java:

1. Para tipos primitivos (int, double, char, etc.), Java trabalha diretamente com valores armazenados na memória.
2. Para objetos (incluindo arrays), Java trabalha com **referências**. Quando você cria um objeto, a variável contém um "ponteiro indireto" para a área de memória onde o objeto está armazenado.

```
Java                                   Copiar código
String nome = "Ana";
// A variável "nome" contém uma referência ao objeto "Ana" na memória.
```

Diferença-chave:

- Com um ponteiro, você poderia manipular diretamente o endereço de memória.
- Com uma referência em Java, você só pode interagir com o objeto através de métodos e atributos expostos por ele.

Por que Java não usa ponteiros diretamente?

1. **Segurança:** Impedir manipulação direta de memória ajuda a evitar bugs como estouro de buffer e acessos ilegais.
2. **Facilidade:** Programar sem ponteiros explícitos reduz a complexidade do código.
3. **Gerenciamento de Memória:** O Garbage Collector do Java cuida automaticamente da alocação e liberação de memória.

3.

Como acessar endereço de memória de outra variável?

O endereço de memória é o local específico na memória do computador onde uma variável ou objeto está armazenado. Cada variável ou objeto em um programa ocupa um espaço na memória do sistema, e esse espaço tem um identificador único chamado endereço.

Analogia Simples

Pensar na memória do computador como um prédio cheio de apartamentos:

- Cada apartamento é um espaço de memória.
- O número do apartamento é o endereço de memória.
- O conteúdo do apartamento é o valor armazenado na memória.

Quando falamos sobre "endereço de memória", estamos nos referindo ao "número do apartamento", ainda uma determinada variável ou objeto está localizado.

```
Copiar código
Como funciona:
Variáveis "normais" (valores diretos)
• Quando você declara uma variável como: int a = 42;, o valor 42 é armazenado diretamente em uma posição específica da memória, e você acessa essa posição pelo nome a.
Ponteiros (em linguagens como C)
• Um ponteiro é apenas uma variável especial que armazena o endereço de memória de outra variável.
• Por exemplo, se p está na posição de memória 8000000, você pode usar um ponteiro para guardar essa posição.
Copiar código
int a = 42;  // Valor 42 armazenado na memória
int *p = &a; // Ponteiro "p" guarda o endereço de "a"
Java
• p guarda o valor 42.
• p guarda o endereço de memória de p (digamos, 8000004).
Ideiais o ponteiro p, você pode acessar ou alterar o valor armazenado em p.
```

A plataforma Java é um ambiente de software no qual os programas escritos em Java são resolvidos, essa plataforma é composta por dois elementos

- Biblioteca de classe para desenvolvimento Java.
- Java Machine, que o responsável por "executar" os programas

Nos decodificando usando instruções em portugues, essa interpretação passará por um processo de compilação e depois a máquina virtual Java (JVM) se encarregará de se encarregar de se comunicar com o seu computador/dispositivo

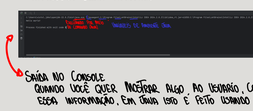
Funcionamento

1. Código fonte (Source Code)
   Quando você escreve um programa em Java, você cria um arquivo com extensão .Java, que contém o código-fonte em linguagem humana (Código de alto nível)
2. Compilação (Compilation)
   A etapa de compilação transforma seu código-fonte em formato que o computador possa entender facilmente.
   - O compilador javac (Java Compiler) pega o arquivo e converte para Byte Code
   - O Byte code é armazenado em um arquivo .class
3. Bytecode
   Ele é uma linguagem intermediária. Ele não é diretamente compreensível para o computador, mas é projetado para ser executado pela máquina virtual Java JVM. O Bytecode é portável, ou seja, pode ser executado em qualquer sistema operacional que tenha uma JVM instalada
4. Execução
   Agora que temos o Bytecode (class) entre a JVM. A máquina virtual Java realiza os seguintes passos

   1. Interpretação: A JVM lê o bytecode linha por linha e converte para a linguagem de máquina (code binário nível) específica do seu sistema operacional e do processador que esta rodando o programa
   2. Just-in-Time Compilation: Para melhorar a performance, a JVM pode converter partes do bytecode diretamente em código de máquina nativo, evitando a interpretação repetitiva

Resumo do Processo

1. Escrita: Você cria o código em arquivo .Java
2. Compilação: O compilador (javac) converte o .Java em Bytecode (.class)
3. Execução: A JVM lê e executa o Bytecode, convertendo em linguagem de máquina



O bytecode não roda diretamente na máquina e sim na JVM, dessa forma, é possível executar o programa independentemente da plataforma, não sendo preciso compilar para cada sistema e arquitetura de hardware. Uma vez compilado, o programa pode ser executado em qualquer JVM, não importando a plataforma que está sendo utilizada.

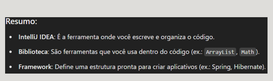
Edições do Java (Conceito interessante, Todas licenças são SE, a o sufixo indicam para licenciamento, como o Spring)

Java SE → Contém as bibliotecas básicas e mais utilizadas (Java Iniciantes)
EE → Focado em desenvolvimento de sistemas corporativos, web, escalaveis, porem tem maior foco e web
FX → Desenvolvido para interfaces ricas, tanto web quanto desktop, é utilizado para criar aplicações rich internet application (RIA), permite criar interfaces de desktop para qualquer plataforma
ME → Focado em dispositivos moveis
CARD → Cartões inteligentes

Edições do Java	Java EE / FX como parte da plataforma Oracle	Java EE virou Jakarta EE (Eclipse Foundation, desde 2017). FX virou parte do JDK no Java 11 e vive como projeto OpenJFX.

Diferenças de Bibliotecas, Frameworks e IDE

1. Bibliotecas (Caixa de ferramentas)
   - É algo que você usa no seu código para resolver problemas específicos
   - Você escolhe as ferramentas que precisa (Ex: Como martelo, chave de fenda e etc)
   - Você é o responsável por decidir onde fazer e como fazer
   - Como uma biblioteca, você controla tudo você chama as funções ou classes que precisa
   - A biblioteca te da liberdade, mas você faz tudo do ZERO!
2. Frameworks
   - É uma estrutura pronta que organiza como o programa deve funcionar, já há um esqueleto de como a "casa" será construída.
   - Você segue as regras e o plano do framework, não pode personalizar algumas partes tem as cores ou os moveis) mas ele controla o fluxo do programa, seguindo oque já preparado
   - Exemplo: Rodar a criar aplicações web e back-end com moveis esforço, ele já define a organização e execução
   - Vantagem: Um framework para trabalhar com banco de dados, ele faz boa parte do trabalho difícil, como salvar dados sem que você precisa escrever SQL
3. IDE
   - Ele é como uma ferramenta de trabalho para programadores, que vários a escrever, organizar e testar o código de forma mais eficiente.
   - O software é usado como editor de código inteligente, depurador (debugger), gerenciador de projetos e integração com ferramentas
   - O intellij idea te ajuda a trabalhar com bibliotecas e frameworks, mas não faz parte do código que você escreve
   - Nele, você adicionou uma biblioteca, como Gson, para trabalhar JSON
   - Se você decidir usar um framework, como Spring Boot, o intellij te ajuda a configurar e seguir as regras do framework



Interface (ferramenta o intellij)

No mercado profissionalmente sempre serão usadas IDE's como as principais ferramentas de trabalho Ex VSCode e Intellij

Estrutura do Projeto!

Toda parte de dentro é chamada de Editor

Logo no início do editor, nas primeiras linhas e em específico na linha 1 é perceptível a existência de codigos, eles são um modelo básico (template) automatico para se começar.

Quando se cria um novo projeto, o intellij precisa de uma classe para iniciar. O nome Main é padrão e indica que essa a classe principal do programa

Oque é? vou dividir e 3 partes (1/3)

```
1  ▷    public class Main {
```

Esse é o método principal (main), que serve como ponto de entrada do programa. Quando executamos o código, aqui que o Java começa a rodar, ele é o ponto de partida

O método main é obrigatório em todo programa Java que você executar. Sem ele, o Java não sabe por onde começar a rodar

Pense numa classe como uma caixa onde todo o seu código vai morar. Em Java, existe uma regra rígida: nada pode ficar solto, tudo tem que estar dentro de uma classe. Então essa linha é você abrindo essa caixa

Oque é? (2/3)

```
public static void main(String[] args) {
```

Ainda segue como parte do método main, porem agora parte vamos destrancar e explicar melhor

Se a classe é a caixa, isto aqui é uma ação dentro da caixa, em Java chamamos de método (em outras linguagens você ouve "função"). Um método é um bloco de instruções que o computador executa

Oque significa cada parte

- Public: diz quem pode executar/usar essa caixa. O Public é um "todo mundo pode", ele permite que o método seja acessado por qualquer parte do programa
- Static: significa que o método pertence a classe em si, e não a um objeto criado a partir dela. (se ainda essa não for nada, tudo bem, ainda vamos aprender esse ponto na próxima parte)
- Void: é o tipo de retorno que diz "não, ou seja: esse método não devolve nenhum valor para quem o chamou" ou seja o trabalho é apenas
- String[] Args: os parâmetros, ou seja, informações que o método pode receber de fora. Aqui é uma lista ([]) de textos (String) chamada Args. Serve para passar dados pro programa na hora de executar. No começo você quase nunca vai usar isso, mas ele precisa estar ali, o {} abre o corpo do método: as instruções que ele vai executar

Programa Wireiro (3/3)

Agora o programa wireiro

```
1  ▷    public static void main(String[] args) {
         System.out.println("Hello, World!");
     }
}
```

Oque é?

```
System.out.println("Hello, World!");
```

Esse comando imprime "Hello World!" no console
E só um exemplo para te mostrar que o programa está funcionando, do executar, apareceria no console
Apareceria dessa maneira

```
C:\Users\Ana Clara Da Mata>cd C:\Users\Ana Clara Da Mata\Desktop\JV Pessoal\Java
C:\Users\Ana Clara Da Mata>cd C:\Users\Ana Clara Da Mata\Desktop\JV Pessoal\Java>
C:\Users\Ana Clara Da Mata>cd C:\Users\Ana Clara Da Mata\Desktop\JV Pessoal\Java>javac Main.class
C:\Users\Ana Clara Da Mata>cd C:\Users\Ana Clara Da Mata\Desktop\JV Pessoal\Java>java Main
Hello, World!
```

Saída no Console
Quando você quer imprimir algo no terminal, como mensagem, número ou resultados de calculos, você usa o Console para imprimir
Essa informação, em Java isso é feito usando o comando System.out.println("Mensagem aqui");

Curiosidade!
A explicação da origem do nome, foi que enquanto planejavam numa estruturar de diretórios para a linguagem, desenharam pela janela um carvalho. Mas esse nome já estava registrado, então o nome acabou surgindo na cafeteria local da cidade onde tomavam café Java, pois era o nome da terra de origem do café

· Níveis de Linguagem!

- Alto Nível: É aquela cuja sintaxe é voltada o entendimento humano, é um tipo de linguagem de programação que se baseia na compreensão humana. Por isso, é mais intuitivo e fácil de aprender, especialmente para iniciantes, oque facilita projetos complexos, alguns exemplos de linguagens: Java, C#, Ruby, Python, Swift, Javascript, PHP.
- Baixo Nível: É um tipo de código que se aproxima do código de máquina, sendo constituído por instruções que são mais diretas ao processador, as linguagens de baixo nível são mais complexas e menos intuitivas, mas oferecem mais controle e desempenho
  Características
  - São mais próximas do hardware do computador
  - São compostas por instruções que são facilmente compreendidas pelo computador
  - São utilizadas para programas operacionais, drivers de dispositivo e outras aplicações de nível sistema

  Exigem mais tempo e esforço para escrever código e depurados
  Ex: Assembly, C, C++, Assembly ARM, Assembly AMD/Intel, Assembly 6502

Case Sensitive

"Case" aqui quer dizer caixa da letra: caixa alta (A, maiúscula) e caixa baixa (a, minúscula).
"Sensitive" = sensível. Então sensitive sensitive significa: a linguagem enxerga letra minúscula e maiúscula como coisas completamente diferentes!

Para um ser humano, casa, Casa e CASA são a mesma coisa são a mesma palavra escrita de jeitos diferentes. Para o Java, são três nomes distintos, sem nenhuma relação entre si, tão diferentes quanto casa e nuvem
Isso vale para tudo: nomes de classes, de métodos, de variáveis e também as palavras da propria linguagem.

Na pratica é assim:

Você escreve	O Java entende
public	palavra da linguagem, função
Public	um nome qualquer, desconhecido = erro
String	o tipo texto do Java, função
string	não existe = erro
main	o ponto de entrada que o JVM procura
Main	um método comum qualquer = o JVM não acha o ponto de entrada
System.out.println	imprime na tela
system.out.println	erro em dois lugares

Perceba no caso do main, é exatamente o que conversamos acima. A classe se chama Main (M maiúsculo) e o método se chama main (m minúsculo), isso não é redundância, nem descuido, são dois nomes diferentes, propositalmente. Trocar a caixa de um deles quebra o programa

Regra pratica: em Java, digitar é literal

E existem convenções (não são regras obrigatórias, mas todo programador Java segue):

- Classes começam com maiúscula: Main, Pessoa, ContaBancaria
- Métodos e variáveis começam com minúscula: main, nome, calcularSalario
- Quando o nome tem varias palavras, grude e bote a inicial de cada palavra seguinte, isso se chama CamelCase: calcularSalarioLiquido

Palavras Reservadas

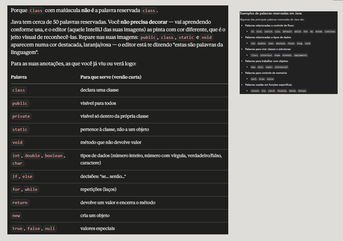
Palavras Reservadas ou também chamadas (Keywords ou Palavras-Chave) são as palavras que pertencem à linguagem. O Java já as usa para dar ordens específicas ao computador, por isso elas são "reservadas", ou seja, você não pode usálas como nome dos seus próprios classes, métodos ou variáveis.

A analogia mais direta: numa língua, se encontrar referente são palavras com função gramatical. Você não batizaria seu cachorro de "enquanto" imagine a frase "enquanto está com fome". Ficaria ambíguo. As palavras reservadas do Java são isso: o compilador precisa que elas signifiquem sempre a mesma coisa para não ficar em duvida sobre oque você quer dizer.

```
int class = 10;      // erro: 'class' é reservada
String public = "oi"; // erro: 'public' é reservada
```

Porque class com maiúscula não é a palavra reservada class.
Java tem cerca de 50 palavras reservadas. Você não precisa decorar — vai aprendendo conforme usa, e o editor (aquele Intelli das suas imagens) as pinta com cor diferente, que é o jeito visual de reconhecê-las. Repare nas suas imagens: public, class, static e void aparecem numa cor destacada, laranja/roxa — o editor está te dizendo "estas são palavras da linguagem".
Para as suas anotações, as que você já viu ou verá logo

Palavra	Para que serve (versão curta)
class	declara uma classe
public	visível para todos
private	visível só dentro da própria classe
static	pertence à classe, não a um objeto
void	método que não devolve valor
int, double, boolean, char	tipos de dados (número inteiro, número com vírgula, verdadeiro/falso, caractere)
if, else	decidem "se... senão..."
for, while	repetições (laços)
return	devolve um valor e encerra o método
new	cria um objeto
true, false, null	valores especiais



não é permitido declarar variáveis cujo nome seja igual a uma das 56 palavras reservadas em java

abstract	continue	for	new	switch	assert	default	goto	package	synchronized
boolean	do	if	private	this	break	double	implements	protected	throw
byte	else	import	public	throws	case	enum	instanceof	return	transient
catch	int	extends							

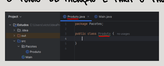
MÃO NA MASSA

CODAGEM

Antes de dar início ao processo de codificação, é importante ressaltar que existe um **PADRÃO** de **NOMENCLATURA**.

No Java não é preferível ter duas **Classes** com o mesmo nome, no mesmo **PACOTE**. Porém, em pacotes diferentes, desta forma, se o projeto utiliza uma biblioteca de classes fornecidas por outras empresas, não teremos conflito de nomes. Já que cada classe terá o seu pacote correspondente.

No Java, assim como em outras linguagens, existem os padrões de nomenclatura. Que são convenções recomendadas para tornar o código mais legível e organizado. Diferente das palavras reservadas, elas não são obrigatórias, não são boas práticas.
O ponto de atenção é para o padrão de nomenclatura das classes: para os pacotes, nunca que todos os caracteres são **minusculos**. Para as classes o padrão é conhecido como **UPPERCAMELCASE**, ou seja, a primeira letra da classe é **MAIUSCULA** e as demais **MINUSCULAS**. Só o nome possui mais de uma palavra, as outras palavras também devem começar com **Maiuscula**. Ex: Pessoa, PessoaFisica e ContaCorrente



Depois da declaração do pacote temos a declaração das classes, a palavra reservada **CLASS** define a classe, seguida do nome da classe. Uma regra importante é que o nome do arquivo deve ser o mesmo da classe Java.
Ex: "Produto.Java" e o nome da classe "Produto", caso você altere alguns desses nomes, o programa Java não vai mais compilar, inclusive as letras **MINUSCULAS** e **MAIUSCULAS** são diferenciadas no Java, no caso o **Case Sensitive**. Como mostrado anteriormente

As chaves [ ] **Delimitam** os **Blocos** de um programa Java, neste caso, as chaves dais o nome da class estão delimitando o **Inicio** e o **Fim**

Os blocos possuem hierarquia e as regras de nomenclatura funcionam por cada bloco, sendo assim, é possível criar vários **variaveis** com nomes iá usados porém em blocos **diferentes**

Só variaveis **Frase**

Para String é recomendado usa o: \n para quebra de linha

O que é uma variável?

Uma variavel é um espaço nomeado na memoria do computador onde voce guarda um valor para usar depois.

A imagem mental mais util no começo é a de uma gaveta

O armario é a memoria, cada gaveta é uma variavel
Cada gaveta tem três coisas
- Tipo: que especie de conteudo ela aceita. Gaveta de meia não recebe cadaço. Definição e imutavel
- Etiqueta(nome): Como voce chama aquela gaveta especifica para encontrala. Ex: meias, cueca, camisetas
- Valor: Onde esta guardado ali dentro. Pode ser trocado quantas vezes quiser, desde que respeite o tipo.

```
int cuecas = 5;   //  tipo   etiqueta   valor
```

Uma gaveta guarda um conteúdo por vez. Meias e cuecas exigem duas gavetas, ainda que ambas sejam do mesmo tipo
Consultar uma gaveta não a esvazia. O programa pede, olha quantas cuecas tem, anota o numero e fecha. As 5 cuecas continuam lá. Ele trabalha com a cópia da informação. Calculos criam ou preenchem outras gavetas, sem mexer nas originais

Tipos de dados (minha anotação antiga)

O Java contem DOIS tipos de dados.

Não cria o primitivo x referencia (ou objetos).

Comecemos pelos **primitivos**:

- No Java, os tipos primitivos são os blocos básicos para armazenar valores simples (números, caracteres, etc.). Eles são os mais leves e não são considerados objetos.

Tipos numericos inteiros (sem casa decimal)

1. **Byte**
   - Tamanho: 8 bits (1 byte)
   - Variavel: -128 a 127
   - Uso: Utilizado quando é necessário economizar memória, como para armazenar números pequenos em dados bem, arrays grandes.
2. **Short**
   - Tamanho: 16 bits (2 byte)
   - Variavel: -32.768 a 32.767
   - Uso: Utilizado para economizar memória em comparação ao **INT**, mas raramente é necessário em programas modernos.
3. **INT** (mais comum)
   - Tamanho: 32 bits (4 bytes)
   - Variavel: -2.147.483.648 a 2.147.483.647
   - Uso: Ideal para armazenar números inteiros na maioria das vezes.
4. **Long**
   - Tamanho: 64 bits (8 bytes)
   - Variavel: -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
   - Uso: Usado quando é necessário trabalhar com números maiores que os suportados pelo **INT**.

	Uso principal
1	Números pequenos (economia de memória).
2	Números inteiros menores.
3	Números inteiros comuns.
4	Números inteiros grandes.
5	Números decimais leves.
6	Números decimais precisos.
7	Representar letras e símbolos.
8	Lógica e controle de fluxo.

Numeros decimais (ponto flutuante)

Os tipos **float** e **double** tem tamanhos menores porque precisam armazenar números com casas decimais, inclusivé o **Expoente** x **Mantissa** (que definem a precisão do número).

5. **Float**
   - Descrição: Tipo numerico de ponto flutuante de precisão simples
   - Tamanho: 32 bits (4 bytes)
   - Variavel/Precisão: ~ 6-7 casas decimais. $\pm 1.4 \times 10^{-45}$ a $\pm 3.4 \times 10^{38}$
   - Uso: Para números decimais onde precisão não é crítica.
6. **Double**
   - Descrição: Tipo numerico de ponto flutuante de precisão dupla
   - Tamanho: 64 bits (8 bytes)
   - Variavel/Precisão: 15 casas decimais.
   - Uso: Para números decimais de maior precisão
7. **Char**
   - Descrição: Representa uma única caractere unicode.
   - Tamanho: 16 bits (2 bytes).
   - Variavel: 0 a 65.535 (valores unicode)
   - Uso: Para armazenar letras, números ou símbolos individuais
8. **Boolean**
   - Descrição: Representa valores lógicos (verdadeiro ou falso)
   - Tamanho: 1 byte porém não é especificado, depende da implementação da Jvm.
   - Variavel: TRUE ou FALSE
   - Uso: Para controle de decisões

Byte e Bits

Bits ou Dígitos (Binary Digits), são os unidades mais básicas de informação em computação.
É a definição de menor informação em um computador. Um bit pode assumir apenas dois valores: 0 ou 1 (ligado ou desligado) verdadeiro ou falso

Byte: É uma unidade de medida de informação na computação e armazenagem de dados
Não mais pra uma unidade básica para medir tamanho de arquivos, memoria e capacidade de armazenamento em dispositivos
Usado para representar valores maiores na execução (como letras, numeros) no sistema.

Referencia (ou objetos)

1. Além dos tipos primitivos, temos os tipos por referência, que armazenam endereços de memória, onde os dados reais estão localizados, isso significa que, em vez de armazenar diretamente um valor (como tipos primitivos) porque, uma referência aponta para um objeto na memória HEAP. Todos os objetos em Java são manipulados por referência.

Ex

```
String nome = "João";  // nome é uma referência ao objeto "João" na heap
```

Aqui, **NOME** não armazena diretamente "**JOÃO**", mas sim o endereço da memória onde o objeto **JOÃO** está armazenado.

Uma referência é um valor que indica a localização de um dado na memória da computador, enquanto um objeto é uma unidade do código que representa um conceito ou uma coisa do mundo real.

Tipos de referencias

2.1 Objetos (new)

Em Java, toda classe pode ser usada para criar objetos. 1 vez objeto não manipulados diferente da referencia

Quando voce cria um objeto usando (NEW)

```
String nome = new String("OLA");
```

nome não contem o valor "OLA" diretamente, ele contem a **REFERENCIA** para o local na memoria onde "OLA" está armazenado.

2.2 Referencias nulas (null)

Uma referencia pode apontar para nada, usando null.

```
String vazio = null;
```

Aqui o vazio não aponta para nenhum objeto

2.3 Arrays

Arrays em Java também são objetos. quando voce cria um array, a variavel armazena uma referencia.

```
int[] numeros = new int[5];
```

2.4 Classes internas (aninhadas)

Pode ser criar classes dentro de outras classes 1 referenciadas

```
Externa.Interna obj = new Externa().new Interna();
```

2.5 Estáticas static

Variaveis e métodos com o modificador **static** pertencem a classe, não ao objeto

```
MinhaClasse.atributoEstatico = 10;
```

Ou referência. NomeClasse é usado para acessar atributo estatico

Objetos em programação representam coisas no mundo real, bem como carros criados com suas características e comportamentos específicos. Um objeto tem suas características, ações que combinam variáveis, funções e estruturas de dados. Usando o nome do objeto e a sintaxe de ponto com a linguagem, você pode visualizar os valores do objeto e chamar as funções que ele produzirem.

Os elementos de um objeto se dividem em duas categorias principais: **propriedades (atributos)** e **métodos**.

As propriedades, também conhecidas como atributos, incluem informações sobre o objeto. Por exemplo, se considerarmos um objeto Carro, algumas de suas propriedades são: **cor, marca, modelo** ou **ano de fab.**

Os métodos definem as operações que podem ser realizadas no objeto. Por exemplo, para o objeto Carro, os métodos podem ser ocultar, abrir ou ligar.

Vamos criar alguns exemplos simples para ilustrar diferentes tipos de objetos:

Conta bancária
- Propriedade: número da conta, titular, saldo.
- Métodos: depositar, sacar, consultar o saldo.

CODAGEM

Sintaxe das variaveis

Para trabalhar com informações dentro de um programa, precisamos de variaveis. Para declararmos no Java, definimos no Java, definimos o **Tipo** de informação que será armazenada e o **nome**.

Sintaxe

<**Tipo da Variavel**> <**nome da variavel**>

Ex: int idade = 25;
double precos = 98.5;

É possivel declarar mais de uma variavel do mesmo tipo na mesma linha

O compilador entende que cada . (ponto) é uma casa decimal

Operadores aritmeticos

Para realizar operações e comparações logicas utilizando símbolos especiais chamados. São cinco no total que realizam operações matematicas são eles:

Operação	Operador	Exemplo
Soma	+	6 + 4
Subtração	-	6 - 4
Multiplicação	*	6 * 4
Divisão	/	6 / 4
Modulo	%	6 % 4

1. Ordem de prioridade dos operadores aritméticos:

Prioridade	Operador	Descrição	Exemplo
1 (mais alta)	()	Parênteses	(2 + 3) * 5 = 25
2	*, /, %	Multiplicação, Divisão, Modulo	8 + 2 * 5 = 18 (multiplica antes de somar)
3 (mais baixa)	+, -	Adição, Subtração	5 - 3 + 2 = 4

- O modulo retorna o resto da divisão de dois numeros inteiros, a divisão entre dois numeros inteiros resulta em um numero inteiro, sem a parte fracionaria, pois é descartada e não faz nenhum arredondamento. Por exemplo, o resultado de 5/2 é 2 e não 2.5 O operador modulo fornece o resto da divisão, desta forma, 5 % 2 produz o resultado 1

5/2
-4 2 → descartado!
-10
-10
0

5/2
-4 2
1

Assim como na matematica no Java é possivel realizar varias operações aritmetica de uma vez. As regras de precedencia de operações aplicadas na matematica são as mesmas no Java
As operações de multiplicação (*), divisão (/) e modulo (%) possuem prioridade em relação as operações de soma (+) e subtração (-). Por exemplo: o resultado da expressão 5*2+1 é 11, pois primeiro é realizada a multiplicação (5*2) e depois a soma (10+1)
Utilizando os parenteses para as operações matematicas, da mesma forma que são usados em expressão algebricas. Assim, a expressão 5*(2+1) produz o resultado 15. primeiro é realizada a soma de 2+1 e depois a multiplicação entre os resultados da soma, 3 e 5, que resulta e 15

Operação simples

```
public class Main {
    public static void main(String[] args) {

        System.out.println(1/2);

    }
```

Para utilizar calculos com numeros quebrados (fracionados) declaramos o **FLOAT** 100f/2f

```
public static void main(String[] args) {

    System.out.println(1+1);
    System.out.println(1*2);
    System.out.println("Resultado" + 100/2);
```

usando em variaveis

```
byte quantidadeOvosVerdes = 50;

byte quantidadeOvosRoxos = 120;

int totalOvos = quantidadeOvosVerdes + quantidadeOvosRoxos;

System.out.println("Total de ovos: " + totalOvos);
```

170

Repare que a variavel **TotalDeOvos** tem seu valor pela soma de duas variaveis, tendo agora o valor de **170**
Neste caso também, é possivel realizar um processo chamado **CAST**
CAST: é quando você força um valor de um tipo a ser convertido para outro tipo

Ex: passar de INT para BYTE

Cuidado com o estouro

```
byte totalOvos = (byte) (quantidadeOvosVerdes + quantidadeOvosRoxos);
```

Porque precisa do **CAST**: o Java faz isso por segurança. Como o tipo BYTE só vai de -128 à 127 o Java quer ter certeza de que você sabe onde está fazendo ao forçar. O valor cabe dentro desse limite
Sem o **CAST** o Java daria erro, se o valor passa de 127, o numero "estoura" e fica errado (se chama overflow)

Operadores de atribuição

Para atribuir um valor a uma variavel, existe um operador de atribuição, como já vimos. A atribuição ocorre sempre após o processamento das expressões do lado direito da atribuição

Ex
INT x = 2; ou VAR x = 2;
x = x + 10; x += 10; o sinal da operação pois vai antes do =
x = 12; x = 12; ++ é apenas uma forma mais curta de escrever

Operador	Exemplo	Equivalente
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Operador de incremento e decremento

São operadores que aumentam ou diminuem o valor de uma variavel em **UMA** unidade

```
int a = 10;

a = a / 5;

a++; // incremento + 1 a

a--; // decremento - 1 a

System.out.println("a: " + a);
```

Inferencia automatica de tipos

A inferencia automatica de tipos é quando o Java adivinha automaticamente o tipo de variavel com base no valor que você atribuiu a ela, sem precisa especificar o tipo ao escrever.

1. Diferença entre Tipos Inferidos e Tipos Pré-determinados	
Tipos Inferidos (var)	**Tipos Pré-determinados (int, double, etc.)**
O tipo é decidido e fixado no valor.	Você especifica o tipo do conteúdo.
Ótimo e código mais limpo e legível.	Ótimo e código mais explícito e claro.
Não pode mudar o tipo depois que é definido.	O tipo já é fixo desde o início.
Introduzido na Java 10.	Sempre existiu no Java.

MÃO NA MASSA

OPERADORES DE COMPARAÇÃO

E PREGUEL COMPARANDO VARIÁVEIS, VALORES OU OUTROS TIPOS DE INFORMAÇÕES UTILIZANDO OPERADORES RELACIONAIS, RETORNANDO VALORES BOOLEAN, TRUE OU FALSE

COMENTÁRIOS IMPORTANTES

OPERADORES DE COMPARAÇÃO		
Operador	Significado	Exemplo
==	Igual	a == b
!=	Diferente	a != b
>	Maior	a > b
<	Menor	a < b
>=	Maior ou igual	a >= b
<=	Menor ou igual	a <= b

```
int idade = 18;
int altura = 180;

System.out.println("Idade: " + idade);
System.out.println("Altura: " + altura);

// Comparações
boolean maior = idade > 17;
boolean igual = altura == 180;
boolean diferente = idade != 20;

System.out.println("Maior: " + maior);
System.out.println("Igual: " + igual);
System.out.println("Diferente: " + diferente);
```

SOBRE O OPERADOR ! ELE INVERTE O RESULTADO, A LOGICA FUNCIONA COMO NA MATEMÁTICA ! (SE 5 FOR MENOR QUE 15 SERÁ TRUE)
!(5 < 15)
!TRUE + ! = FALSE

OPERADORES LÓGICOS

A	B	A && B
True	True	True
True	False	False
False	True	False
False	False	False

+

A	B	A || B
True	True	True
True	False	True
False	True	True
False	False	False

RESUMO

- QUANDO DESEJAMOS QUE AS DUAS EXPRESSÕES SEJAM VERDADEIRAS
- QUANDO PRECISAMOS QUE PELO MENOS UMA DAS EXPRESSÕES SEJA VERDADEIRA

OS OPERADORES LÓGICOS SÃO UTILIZADOS PARA FORMAR EXPRESSÕES DE COMPARAÇÃO COM MAIS TERMO, RESULTANDO EM UM VALOR BOOLEAN

FLUXO DE CONTROLE (SELEÇÃO E LOOPS)

CONTROLE DE FLUXO É O JEITO QUE O JAVA DECIDE ONDE E QUANDO EXECUTAR. ELE É DIVIDIDO EM DUAS CATEGORIAS PRINCIPAIS.

1. SELEÇÕES (DECISÕES) - IF, ELSE, SWITCH
2. LAÇOS DE REPETIÇÃO (LOOPS) - WHILE, FOR, DO-WHILE

1. OS COMANDOS DE SELEÇÃO (IF E ELSE) PERMITEM TOMAR DECISÕES E ALTERAM O SEU FLUXO DE EXECUÇÃO. EX: "SE ESTÁ CHOVENDO, PEGUE UM GUARDA-CHUVA. SENÃO, VAI SEM ELE."

```
int idade = 20;

if (idade >= 18) {
    System.out.println("Você pode votar!");
} else {
    System.out.println("Você não pode votar ainda.");
}
```

IF → EXECUTA SE A CONDIÇÃO FOR VERDADEIRA
ELSE → SE IF RETORNAR FALSE, O SEGUNDO BLOCO ELSE SERÁ EXECUTADO

INSTRUÇÃO CONDICIONAL
VIMOS RELEMBRAR, LINHAS QUE NO CHAVES { } DELIMITAM INÍCIO E FIM DAS CLASSES E DOS MÉTODOS. ASSIM, É POSSÍVEL IMAGINÁ-LOS, OS COMANDOS QUE PERTENCEM À CLASSE É DO MÉTODO.

IF, ELSE IF, ELSE

```
int idade = 20;

if (idade >= 18) {
    System.out.println("Você pode votar!");
} else if (idade >= 16) {
    System.out.println("Você pode votar com autorização");
} else {
    System.out.println("Você não pode votar ainda.");
}
```

BONS COMENTÁRIOS

IF - PRIMEIRO TESTE
ELSE IF - OUTRAS OPÇÕES
ELSE - CASO NENHUMA DAS ANTERIOR FOR VERDADEIRO

AS CONDIÇÕES DEVE SER COLOCADAS ENTRE PARÊNTESES E PODE CONTER VÁRIAS COMPARAÇÕES COMBINADAS COM OPERADORES LÓGICOS, QUE NO FINAL, RESULTAM EM UM VALOR BOOLEANO.

```
if (condição){
    instrução 1;
    instrução 2;
}
```

INSTRUÇÕES - É O BLOCO DE CÓDIGO QUE EXECUTADA SE A CONDIÇÃO FOR VERDADEIRA

AS CHAVES SERVEM PARA DELIMITAR AS INSTRUÇÕES QUE FAZEM PARTE DA ESTRUTURA DE SELEÇÃO.

2. OS LOOPS SERVEM PARA REPETIR UM CÓDIGO VÁRIAS VEZES SEM PRECISAR ESCREVER TUDO MANUALMENTE

Tipos de Loops em Java

Loop	Quando usar
while	Quando você não sabe quantas vezes que repetir.
do-while	Quando não sabe quantas vezes ao repetir (executa ao menos uma execução)
for (clássico)	Quando sabe quantas vezes quer repetir (pelo menos uma vez)

OPERADOR TERNÁRIO

É UMA FORMA CONCISA DE ESCREVER UMA EXPRESSÃO IF-ELSE. ELE USA O SÍMBOLO ? PARA SEPARAR A CONDIÇÃO DA EXPRESSÃO A SER EXECUTADA SE A CONDIÇÃO - FOR VERDADEIRA E : PARA SEPARAR A EXPRESSÃO A SER EXECUTADA SE A CONDIÇÃO FOR FALSA

```
int idade = 20;

String mensagem = (idade >= 18) ? "Pode votar" : "Menor de idade";
System.out.println(mensagem);
```

É UMA FORMA MAIS CURTA DE ESCREVER IF-ELSE NA MESMA LINHA (UNICA LINHA)
ELE DEIXA O CÓDIGO

```
int nota = 8;
String resultado = (nota >= 10) ? "Aprovado" : (nota >= 7) ? "Recuperação" : "Reprovado";

System.out.println("Status Final: " + resultado);
```

ENTRADA E SAÍDA DE DADOS

O COMANDO PARA LER OS DADOS QUE SÃO INSERIDOS PELO USUÁRIO. PARA ISSO, PRECISAMOS DA AJUDA DA CLASSE SCANNER.

A CLASSE SCANNER FAZ PARTE DO PACOTE JAVA.UTIL. ELA PERMITE CAPTURAR DIFERENTES TIPOS DE DADOS COMO TEXTO (STRING), NÚMEROS (INT E DOUBLE) E ATÉ MESMO LINHAS INTEIRAS NOS DECLARANDO UMA VARIÁVEL DO TIPO SCANNER. DEPOIS, ATRIBUÍMOS UM OBJETO DO TIPO SCANNER À VARIÁVEL, ISSO É A ORIENTAÇÃO A OBJETOS.

EX SIMPLES:

```
Scanner teclado = new Scanner(System.in);
int idade = teclado.nextInt();
float nota = teclado.nextFloat();
```

O METODO NEXTINT() LÊ UM NÚMERO INTEIRO. E O NEXTFLOAT LÊ DO TIPO FLOAT

Outros Métodos Comuns do Scanner

Método	Tipo de Dado	Exemplo
nextLine()	Texto completo	Nome completo
next()	Palavra única	Apenas "João"
nextInt()	Número inteiro	Idade
nextDouble()	Número decimal	Preço (12.99)
nextBoolean()	Valor lógico	true ou false

Para primitivos	Métodos do Scanner
byte	nextByte()
short	nextShort()
int	nextInt()
long	nextLong()
float	nextFloat()
double	nextDouble()
boolean	nextBoolean()

PARA UTILIZAR
DEVE-SE IMPORTAR O SCANNER

```
import java.util.Scanner;
```

IMPORTANTE! ↓

```
Scanner teclado = new Scanner(System.in);
System.out.print("Digite seu nome: ");
String nome = teclado.nextLine();
System.out.print("Digite sua idade: ");
int idade = teclado.nextInt();
System.out.print("Digite sua altura: ");
double altura = teclado.nextDouble();
System.out.print("Digite seu sobrenome: ");
String sobrenome = teclado.next();
System.out.print("Digite seu salário: ");
double salario = teclado.nextDouble();
System.out.print("Digite seu nome completo: ");
String nomeCompleto = teclado.nextLine();
System.out.print("Digite seu CPF: ");
long cpf = teclado.nextLong();
System.out.print("Digite seu CEP: ");
int cep = teclado.nextInt();
System.out.print("Digite seu ID: ");
int id = teclado.nextInt();
System.out.print("Digite seu nome: ");
String nome = teclado.nextLine
```