

VEM SER

Módulo 01 - Java + OO

Aula 01 - Fundamentos

Acordos

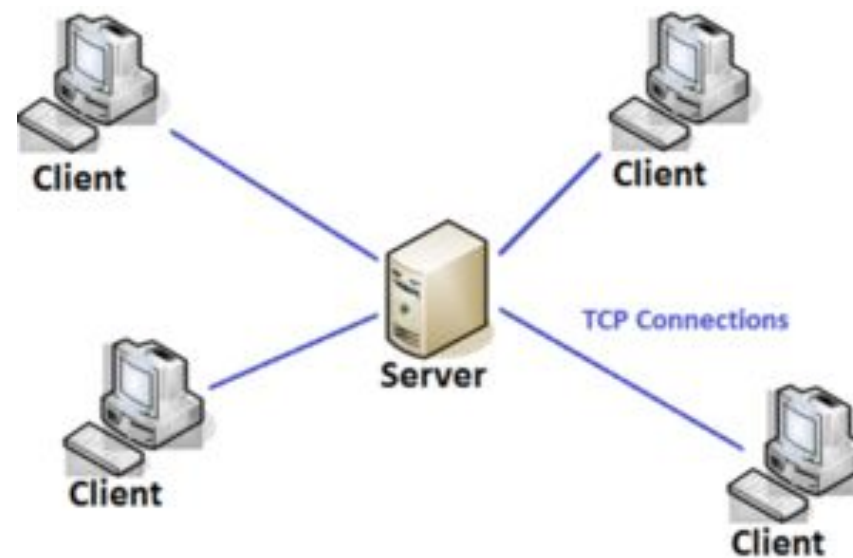
- Câmera sempre aberta;
- Foquem 100% na aula;
- Garrafa de água do lado!;
- Evitem atrasos;
- Sem plágio!

Conteúdo do módulo

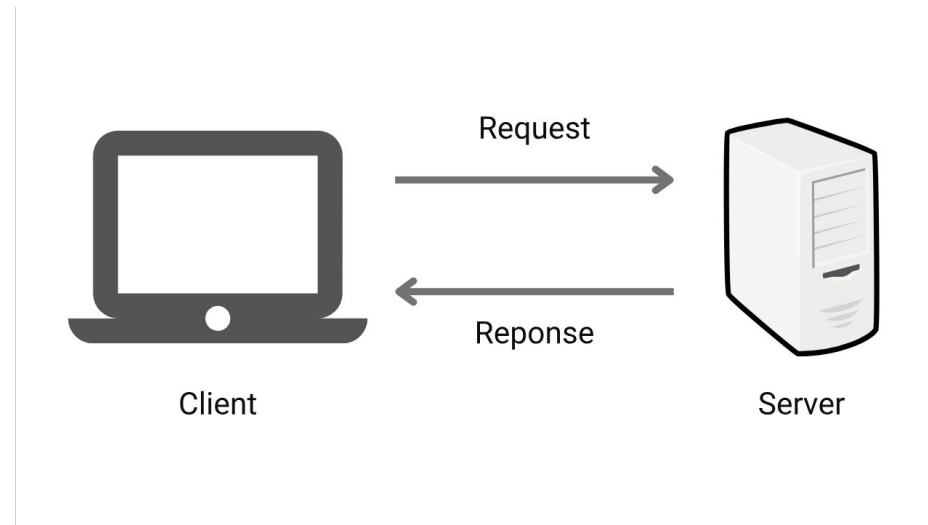
1. Fundamentos da computação
2. Introdução ao backend
3. Tipos de linguagem de programação
4. História do Java
5. Algoritmos
6. Tipagem de dados
7. Armazenamento: variáveis
8. Estruturas condicionais (ou de decisão)
9. Estruturas de repetição (ou laços, ou *loops*)
10. *Arrays*
11. Matrizes
12. Tipos de paradigma de linguagens de programação
13. Orientação a Objetos
14. Collections
15. *Enums*
16. Projeto Final do Módulo

Fundamentos

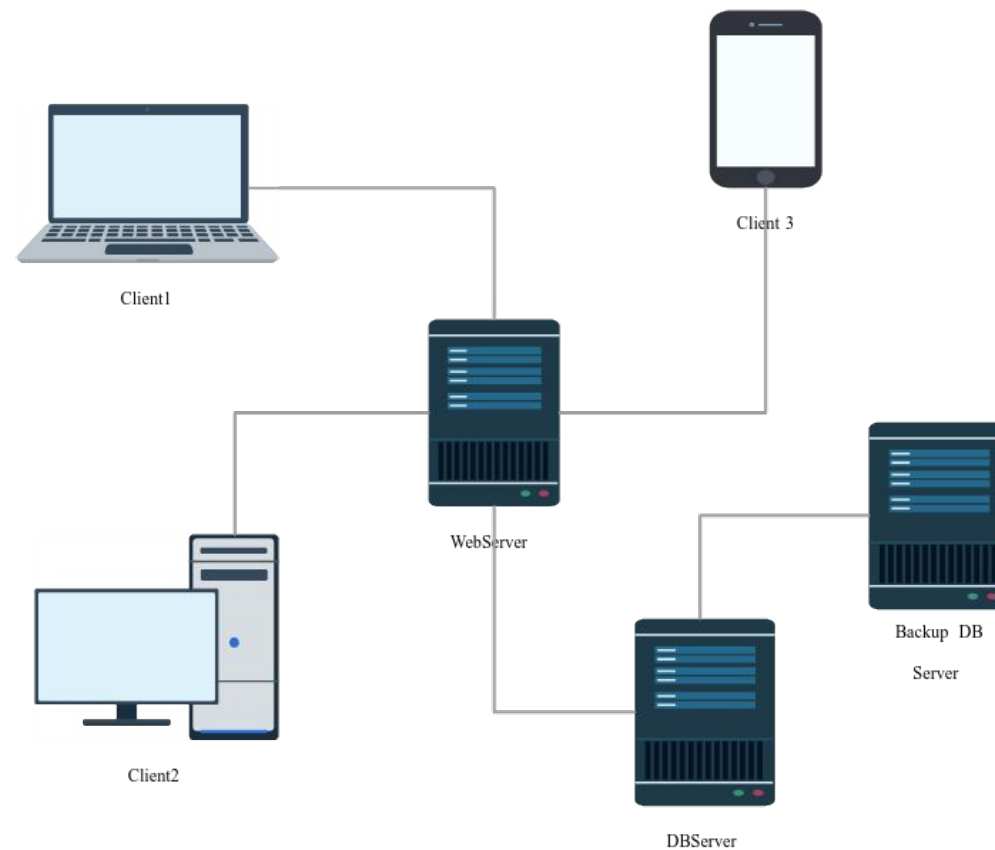
Arquitetura *client-server*



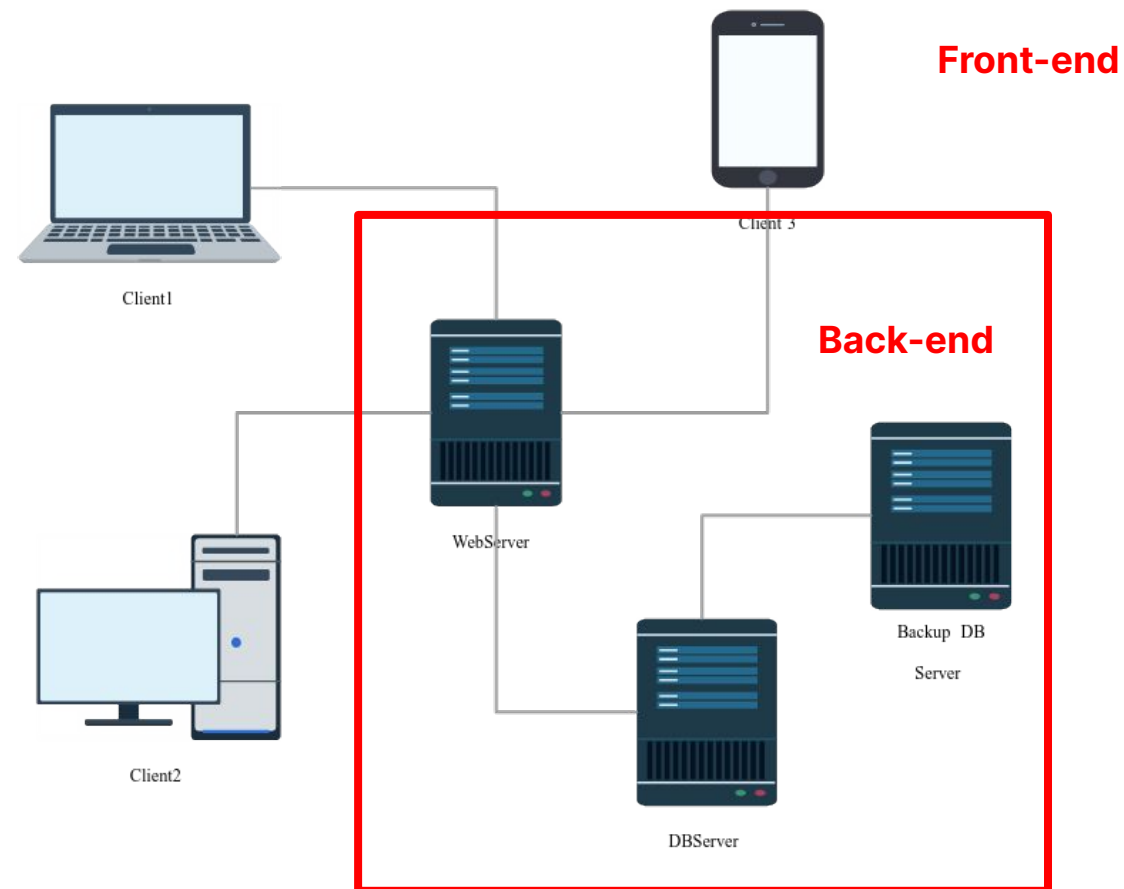
Método *request-response*



Arquitetura *client-server*

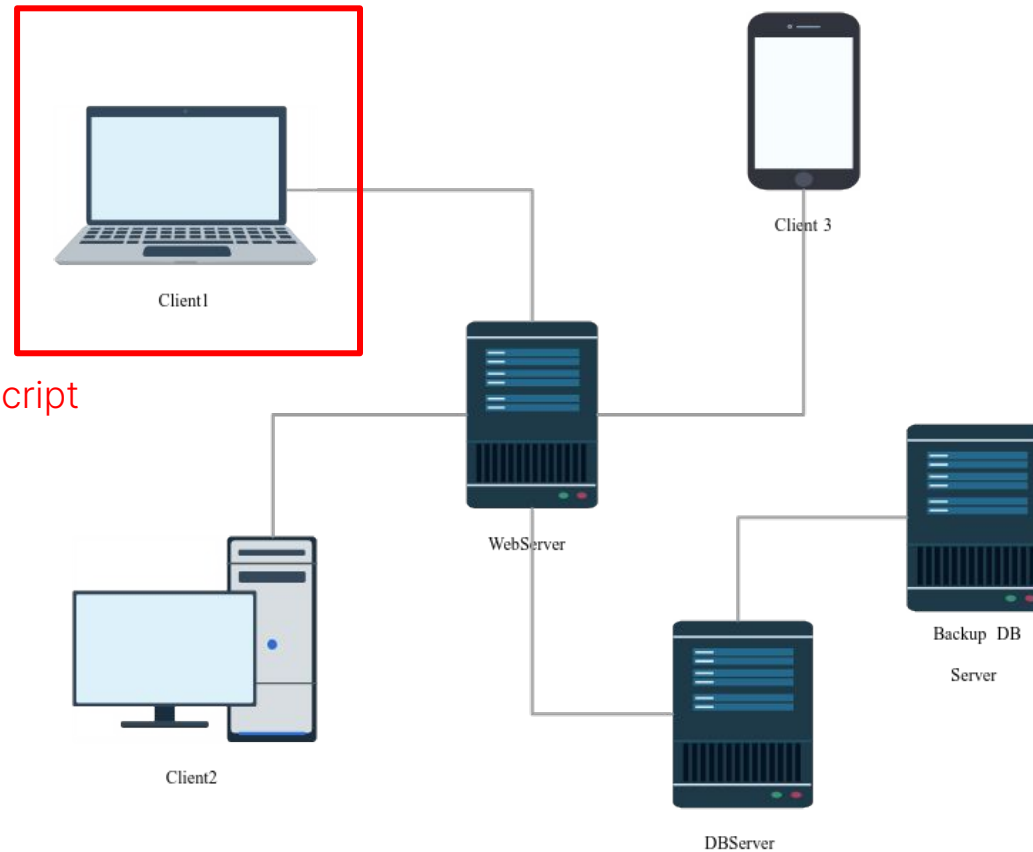


Arquitetura *client-server*

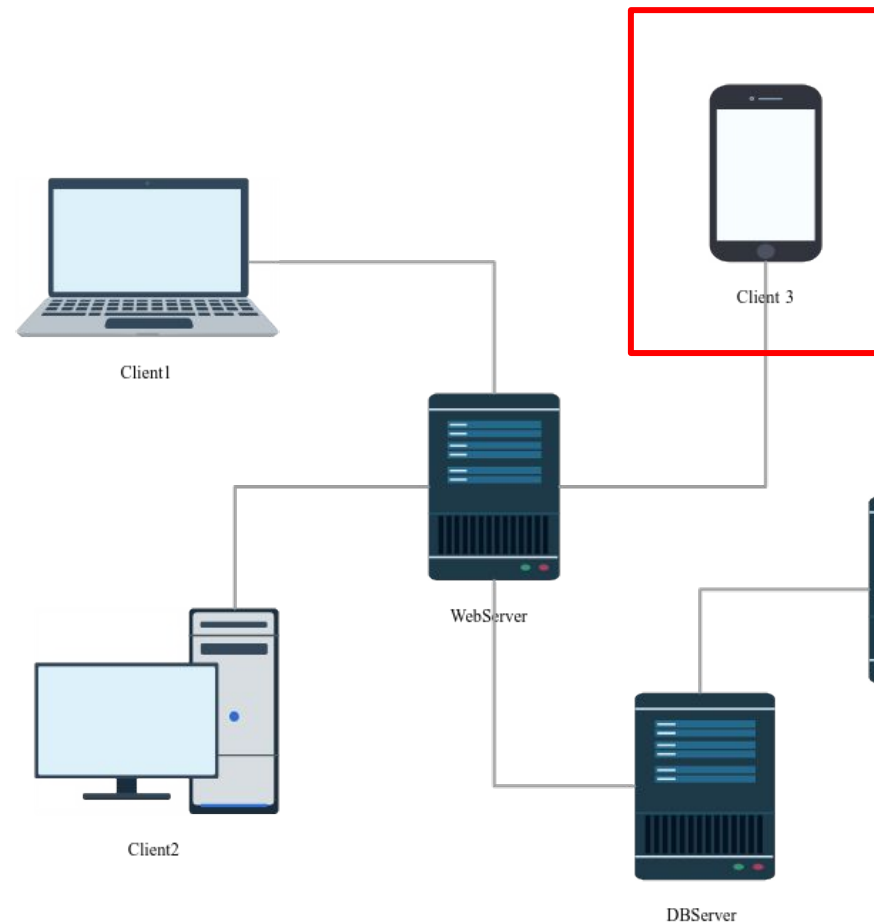


Arquitetura *client-server*

- **Web (Front-end):**
- React
- Angular
- Vue
- Svelte
- Remix
- Gatsby
- HTML, CSS e JavaScript
- jquery
- etc...



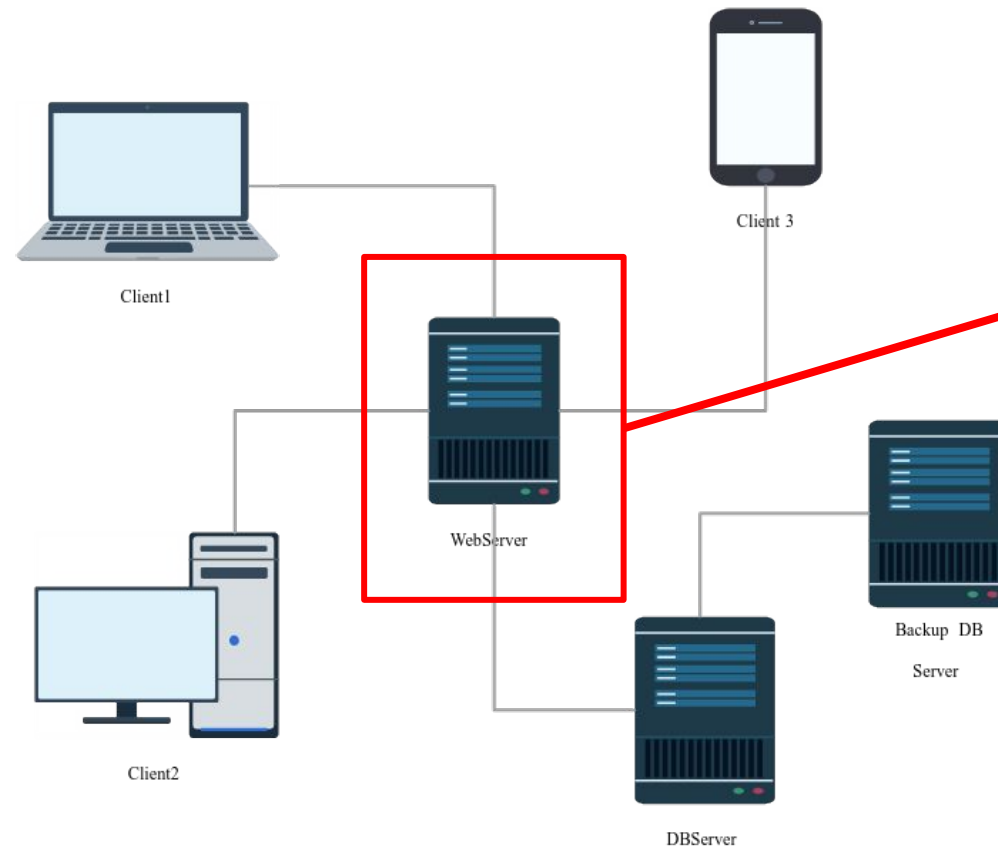
Arquitetura *client-server*



Mobile:

- **Híbrido** - React Native, Flutter, Xamarin, etc...
- **Nativo** - Kotlin, Java (Android) Swift, Objective-C (iOS)

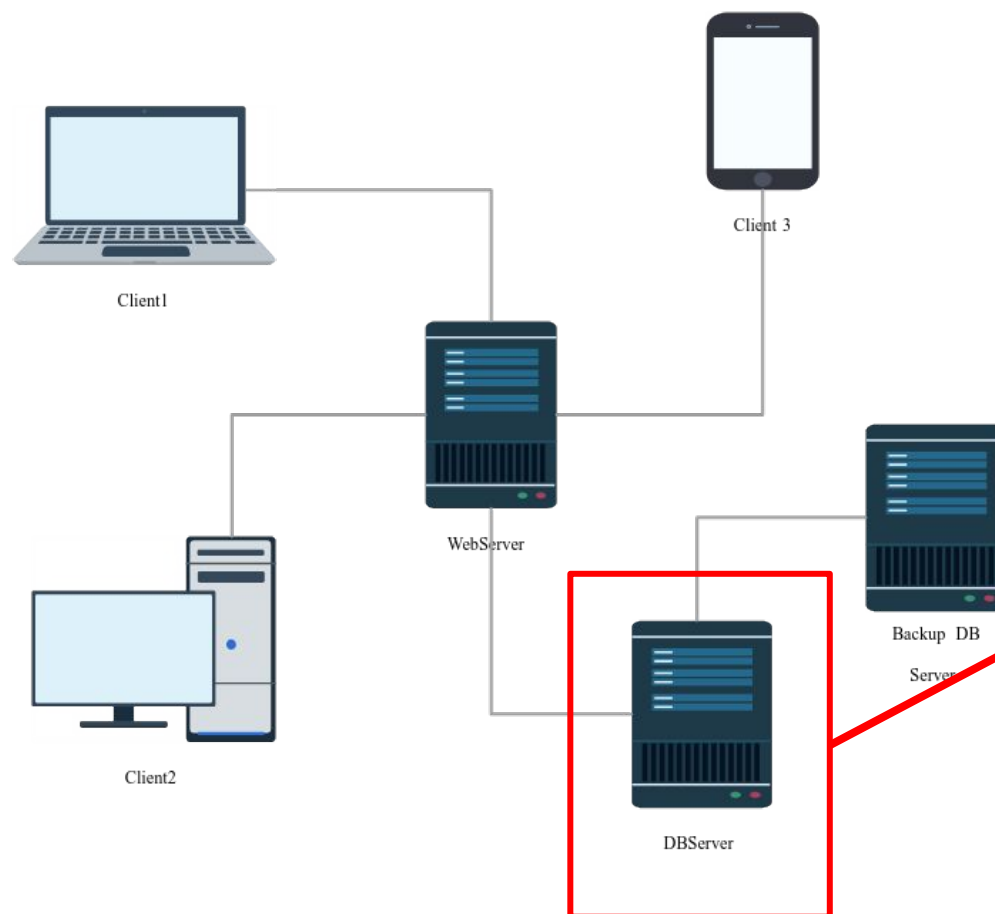
Arquitetura *client-server*



Web (Back-end):

- Java (Spring)
- JavaScript (Node, Nest)
- C# (.Net)
- Go
- Python (Django, Flask)
- Ruby (on Rails)
- Elixir
- PHP (Laravel)
- etc...

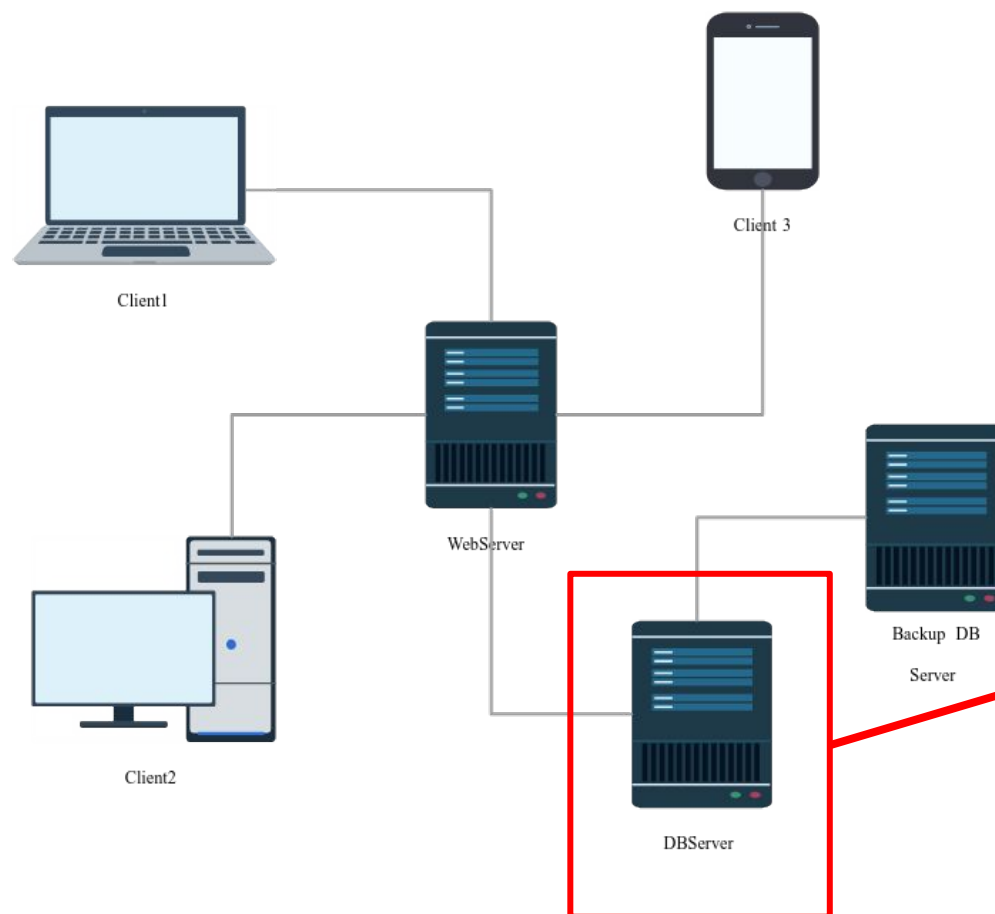
Arquitetura *client-server*



Banco de dados Relacionais:

- Oracle
- MySQL
- MariaDB (*open-source do MySQL*)
- PostgreSQL (*open-source*)
- SQLite
- IBM
- etc...

Arquitetura *client-server*



Banco de dados Não Relacionais (NoSQL):

- MongoDB
- Redis
- Cassandra
- DynamoDB
- etc...

O básico para um dev back-end

O que um dev back-end precisa saber

- Conhecer profundamente uma linguagem de programação
- Conhecer a arquitetura dos servidores
- Ter conhecimentos acima da média em banco de dados
- Conhecer o funcionamento das APIs
- Escalabilidade
- Segurança
- Containers (Docker)
- Git e Github

Tipos de linguagem de programação

Tipos de linguagem de programação

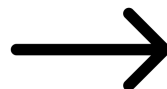
Linguagem compilada

O código-fonte da linguagem é traduzido integralmente para código de máquina (linguagem de baixo nível) antes da execução através de um programa chamado **compilador**.

O compilador analisa o código e gera um arquivo executável independente.



Código-fonte

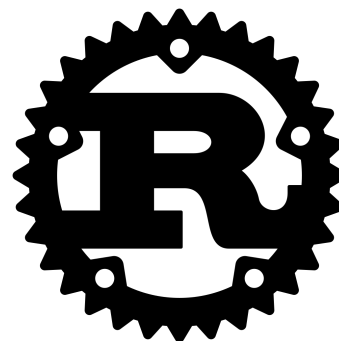
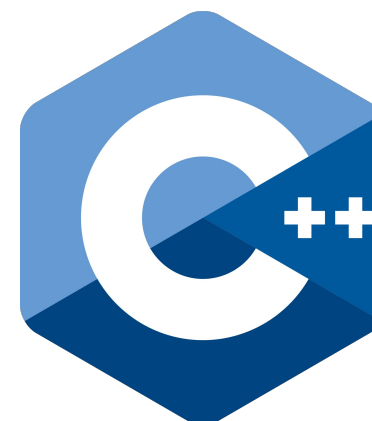


Linguagem de máquina
(Arquivo executável)

Tipos de linguagem de programação

Linguagem compilada

Exemplos: C, C++, Rust, Go.



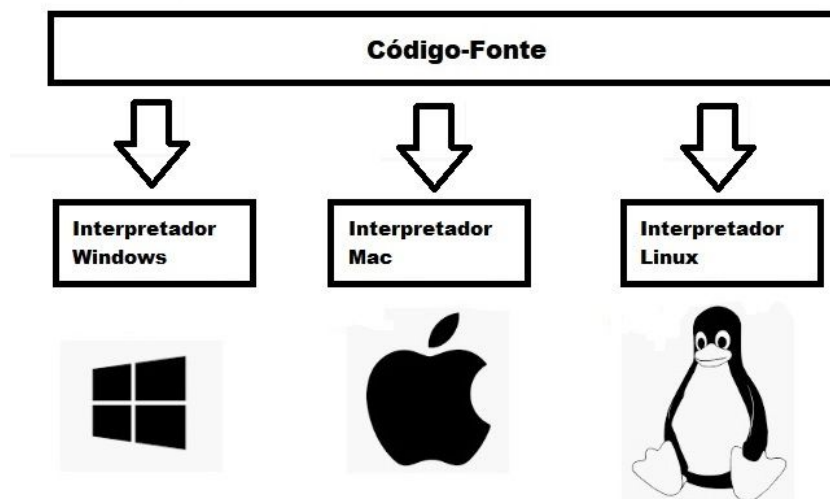
Tipos de linguagem de programação

Linguagem interpretada

O código é interpretado, linha por linha, por um programa chamado **interpretador**.

O interpretador lê e executa cada instrução em tempo de execução (*runtime*), sem necessidade de compilação.

Linguagem Interpretada:



Tipos de linguagem de programação

Linguagem interpretada

Exemplos: JavaScript, Python, Ruby, PHP.



Tipos de linguagem de programação

Linguagem híbrida

Algumas linguagens mesclam interpretação e compilação.

Elas compilam um código intermediário, que é interpretado por uma máquina virtual ou ambiente de execução específico.

Tipos de linguagem de programação

Linguagem híbrida

O Java é uma das principais linguagens que adotam esta forma.

Outro exemplo é o C#.



Tipos de linguagem de programação

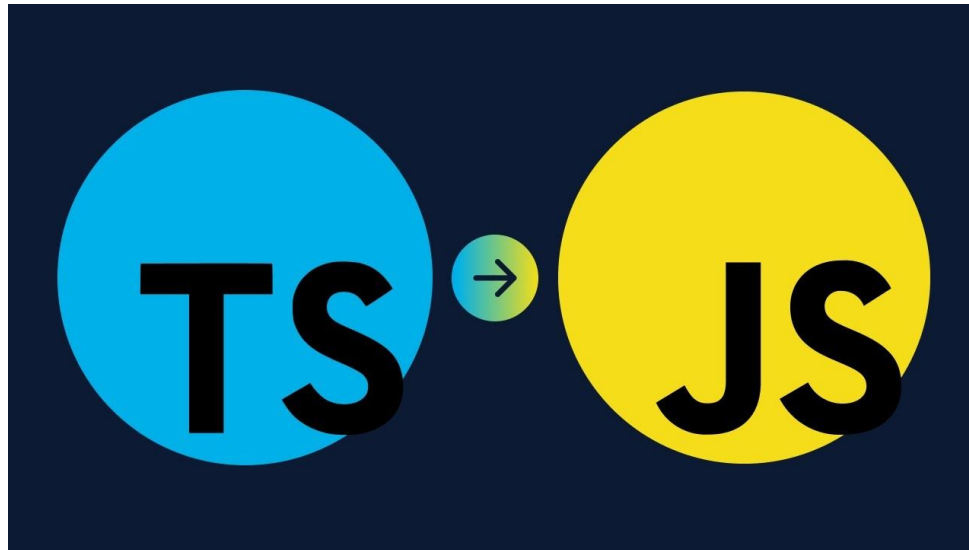
Linguagem transpilada

São linguagens que são “traduzidas” para outra linguagem para que, só então, o código desta última seja executado.

Tipos de linguagem de programação

Linguagem transpilada

Por exemplo: o código TypeScript, para ser rodado, precisa ser convertido para JavaScript e esse código JavaScript que é executado.



Tipos de linguagem de programação

Linguagem transpilada

Exemplos: TypeScript, CoffeeScript.



Tipos de linguagem de programação

Linguagem de *script*

São linguagens voltadas para **automação e/ou execução de tarefas** e geralmente são interpretadas. Elas conseguem, geralmente, fornecer recursos para manipulação de arquivos e interação com sistemas operacionais através de linha de comando.

Tipos de linguagem de programação

Linguagem de *script*

Exemplos: Bash (Linux/Unix), PowerShell (Windows), Perl e Lua.



Java

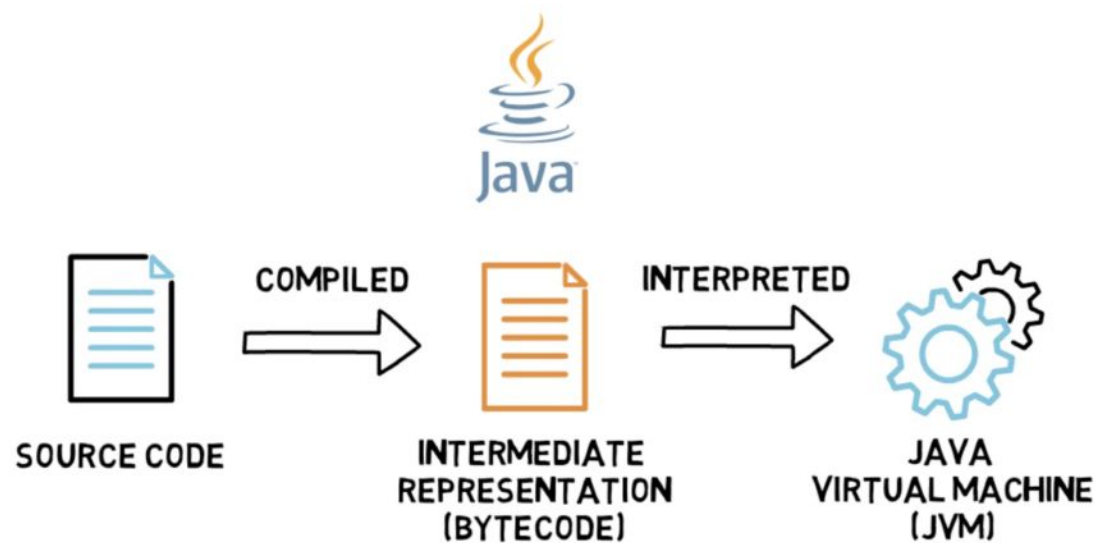
História do Java

- O Java foi criado em 1991 pela empresa Sun Microsystems.
- A ideia do Java era possibilitar que os dispositivos fossem escritos em uma linguagem que proporcionasse que eles fossem interligados.
- Por exemplo, uma cozinha inteligente, em que fogão, geladeira e microondas se comunicam.
- E isso acabou se realizando, nos anos 2000 surgiu o Android, utilizando o kernel Linux de base para o sistema operacional e rodando aplicativos desenvolvidos em Java - hoje temos televisões, geladeiras, até mesmo microondas que rodam o sistema Android.

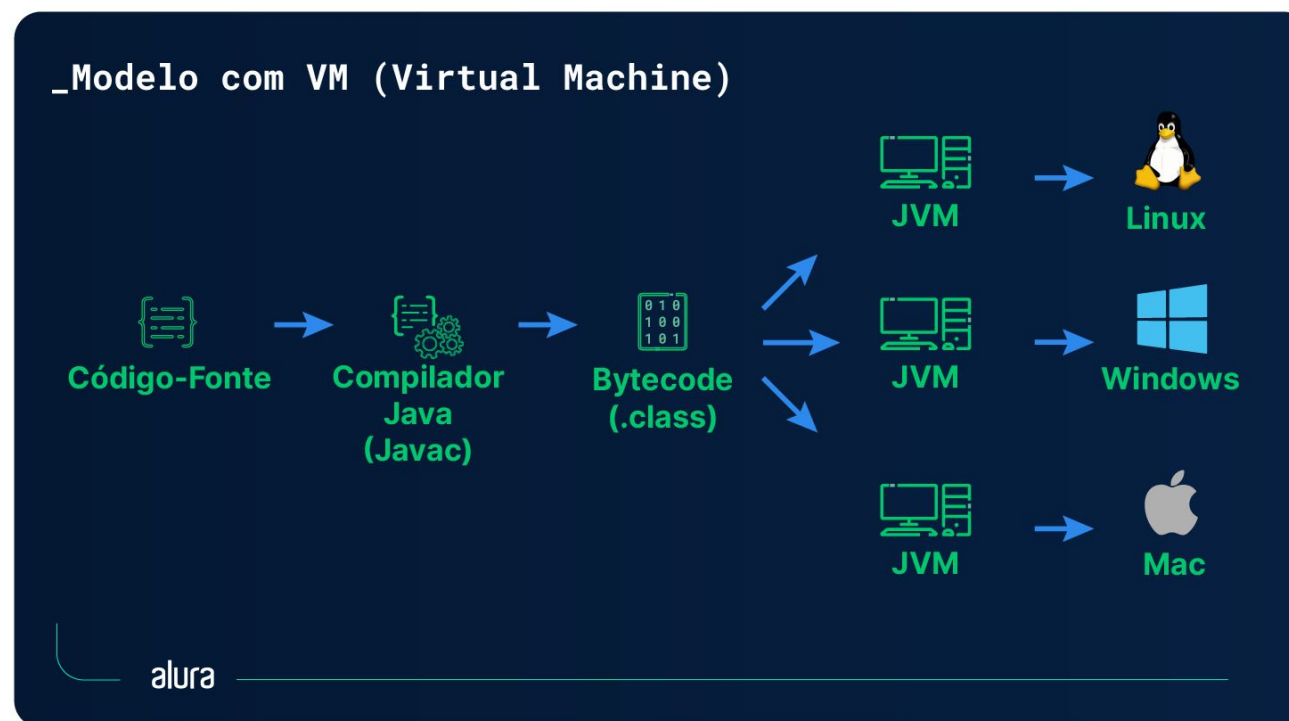
Qual problema o Java resolve?

- O Java se tornou uma solução extremamente usada comercialmente justamente por se tratar de uma linguagem híbrida.
- O maior problema das linguagens compiladas é que são gerados diferentes arquivos para diferentes sistemas operacionais, sendo necessário um time de desenvolvimento para cada um.
- O que a Sun fez foi tirar essa responsabilidade das equipes de desenvolvimento das empresas, através do JVM (Java *Virtual Machine*), toda a parte de integração entre os sistemas operacionais passa a ser obrigação da Sun (e agora da Oracle).

E como ele resolve?



E como ele resolve?



Dicas de leitura

<https://schoolcode.com.br/linguagens-compiladas-interpretadas-e-hibridas>

<https://www.alura.com.br/artigos/java>

<https://www.alura.com.br/artigos/jvm-conhecendo-processo-execucao-de-codigo>

Adendo!

O acrônimo **WORA** (*Write Once Run Anywhere*, em tradução literal: "Escreva um código, rode em qualquer lugar") se refere justamente a essa característica do Java de ser um código "feito para rodar em qualquer lugar".

Algoritmos

Algoritmos

“Um algoritmo representa um conjunto de regras para a solução de um problema. (...) Dessa forma, uma receita de bolo é um exemplo de um algoritmo, pois descreve as regras necessárias para a conclusão de seu objetivo: a preparação de um bolo.”

[SOUZA et al \(2019, p. 4\)](#)

Algoritmos

Em termos gerais, um algoritmo nada mais é do que um conjunto de instruções com um determinado objetivo.

Algoritmos

Quando utilizamos uma linguagem de programação para criar um algoritmo, estamos, na verdade, criando um conjunto de instruções para serem executadas por um computador, visando atingir o objetivo inicial.

Sintaxe de uma linguagem de programação

“A **sintaxe** de um algoritmo resume-se nas regras para escrevê-lo corretamente.” SOUZA et al (2019, p. 12)


Cada linguagem de programação tem sua sintaxe.

Semântica de um código

“(...) a semântica de um algoritmo estabelece regras para sua interpretação.” (SOUZA et al, p. 15)
Nada mais é do que o significado do código.

Semântica de um código

Vamos analisar o código Python abaixo:



```
print("Vem Ser DBC")
```

Qual a semântica dele?
"Imprimir" algo na tela.

Tipagem de dados

Tipagem de dados

Este é um assunto **muito importante** quando estamos falando de linguagens de programação. Existem quatro “tipos de tipagem” de dados em linguagens de programação:

- Estática
- Dinâmica
- Forte
- Fraca

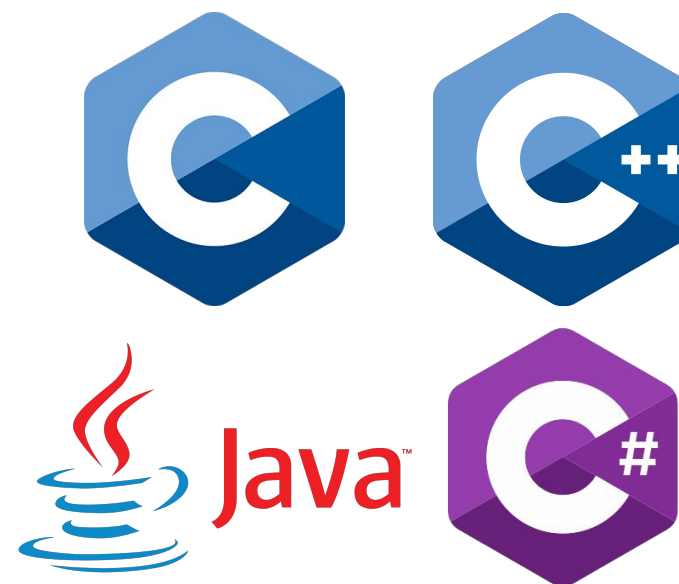


Tipagem estática

Nesse tipo de tipagem, os tipos de dados são verificados em tempo de compilação. Isso quer dizer que as variáveis têm um tipo definido e **não podem ser alteradas posteriormente**.

Tipagem estática

Exemplos de linguagens com tipagem estática são C, C++, Java e C#.



Tipagem dinâmica

Nesse tipo de tipagem, os tipos de dados são verificados em tempo de execução. As variáveis podem ter seu tipo alterado durante a execução do programa.

Tipagem dinâmica

Exemplos de linguagens com tipagem dinâmica são Python, Ruby, JavaScript e PHP.



Tipagem forte

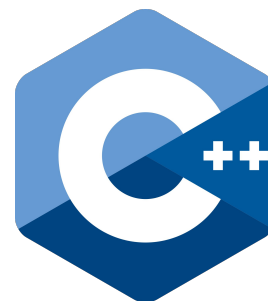
Nesse tipo de tipagem, as restrições dos tipos de dados são rigorosamente aplicadas, e não são permitidas conversões automáticas entre tipos incompatíveis.

Isso significa que é necessário realizar conversões explícitas de tipo quando necessário.

Quando uma linguagem possui tipagem forte, dizemos que ela é **fortemente tipada**.

Tipagem forte

Muitas linguagens com tipagem estática também possuem tipagem forte, como C++ e Java.



Tipagem fraca

Nesse tipo de tipagem, as conversões automáticas entre tipos são permitidas e são realizadas implicitamente.

Isso pode levar a comportamentos inesperados, pois as operações podem ser aplicadas a tipos de dados não esperados.

Quando uma linguagem possui tipagem fraca, dizemos que ela é **fracamente tipada**.

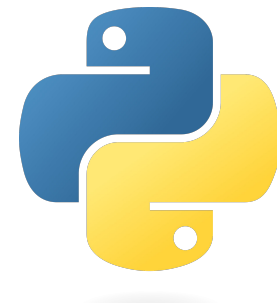
Tipagem fraca

Linguagens com tipagem dinâmica, como JavaScript e PHP, geralmente possuem tipagem fraca.



Adendo!

Isso nem sempre pode ser verdade, Python é um exemplo de linguagem com tipagem dinâmica e que é fortemente tipada, ou seja, precisa de **conversão explícita** para ter o tipo do dado alterado.





JAVA!

Java é uma linguagem **fortemente tipada** e de tipagem **estática**!

Instalações



Primeiro precisamos do Oracle JDK

Baixem o **JDK 17!!!**

<https://www.oracle.com/java/technologies/downloads/#java17>



Sugestão!

Quem utilizar Linux, uma sugestão para depois: **sdkman**

<https://sdkman.io/install>



Agora precisamos de uma IDE (ambiente de desenvolvimento integrado)

Vamos utilizar o IntelliJ, da JetBrains

Baixem a Community Edition

<https://www.jetbrains.com/pt-br/idea/download>

Variáveis

Armazenamento: variáveis

O que é uma variável?

Uma variável, no contexto da programação, é um nome que representa um **valor armazenado** na memória do computador.

Tipos de valores possíveis no Java

No Java, temos os tipos primitivos e os tipos de referência. Neste momento, vamos focar nos **tipos primitivos**.

Eles são tipos de dados básicos que representam valores simples.

São eles:


- Números inteiros;
- Números flutuantes;
- Caractere;
- Booleanos (verdadeiro/falso);
- ...

Tipos primitivos

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

Declaração de variáveis

Operador de **atribuição**



```
tipo nomeDaVariavel = valor;
```



Declaração de variáveis

```
int idade = 10;
```

Dicas de leitura

<https://glysns.gitbook.io/java-basico/sintaxe/variaveis>

<https://www.devmedia.com.br/tipos-de-dados-por-valor-e-por-referencia-em-java/25293>

Padrões de nomenclatura

Padrões (ou convenções) de nomenclatura

Também conhecido como *case styles*. São as boas práticas que devemos seguir ao **nomear variáveis** e demais elementos (classes, objetos, métodos, etc) no nosso código.



Case Styles (ou convenções de nomenclatura)

As principais são:

- **camelCase:** nomeDaVariavel -> primeira palavra começa com letra minúscula e demais palavras começam com a primeira maiúscula. Usada principalmente no **JavaScript e Java**;
- **PascalCase:** NomeDoComponente -> primeira letra de cada palavra maiúscula. Usada principalmente para **classes** (POO) e **nomes de componentes no React**;
- **kebab-case:** nome-do-elemento -> espaço entre palavras substituído pelo hífen (-). Usado principalmente no **CSS**, também para **nome de arquivos** e para **links e URLs**;
- **snake_case:** nome_da_variavel -> espaço entre palavras substituído pelo *underline*/sublinhado (_). Usado principalmente no **Python e C**;

Case Styles (ou convenções de nomenclatura)

Algo comum entre praticamente todas as linguagens é nomear constantes absolutas (**inalteráveis!**) no padrão **SCREAMING_SNAKE_CASE**:

Deve ser nomeado assim: NOME_DA_CONSTANTE.

Todas as palavras em maiúsculo e com espaço substituído por *underline*/sublinhado (_).

Exemplo:

```
double VALOR_DE_PI = 3.141592;
```

Operadores

Operador de atribuição

Nome	Operador
Atribuição	=

Operadores aritméticos

Nome	Operador
Soma	+
Subtração	-
Divisão	/
Multiplicação	*
Módulo (resto da divisão)	%

*Em algumas linguagens, a potenciação é expressa por **, por exemplo $2^3 = 2^{**}3 = 8$

Operadores relacionais

Nome	Operador
Igualdade	==
Diferença*	!=
Menor que	<
Maior que	>
Menor ou igual	<=
Maior ou igual	>=

*Em algumas linguagens de programação a diferença é <>

Operadores lógicos

Nome	Operador
Negação	!
E (AND)	&&
Ou (OR)	

*Leitura recomendada: <https://www.todamateria.com.br/tabela-verdade/>

**AND também chamado de conjunção

**OR também chamado de disjunção

Operadores de incremento

Nome	Operador
Incremento	++
Decremento	--
Atribuição composta	+=
Subtração composta	-=
Multiplicação composta	*=
Divisão composta	/=

Controle de fluxo

O que é o controle de fluxo?

São estruturas através das quais podemos determinar **quais partes do código serão executadas, quantas vezes** e em quais **condições**.

O controle de fluxo é usado para direcionar o fluxo de execução com base em decisões lógicas, repetições e outras condições.

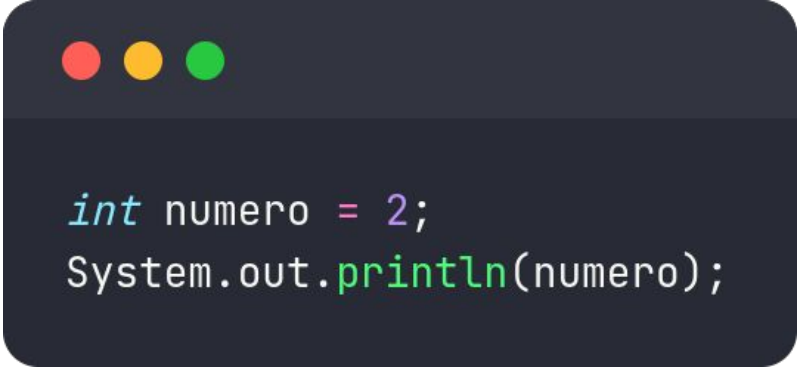
Tipos de estruturas

- Estruturas sequenciais
- Estruturas condicionais (ou de decisão)
- Estruturas de repetição (ou laços, ou *loops*)

Estruturas sequenciais

São estruturas em que o código é executado **imperativamente**, sem interrupção. É o fluxo **padrão** da aplicação.

Por exemplo:



```
int numero = 2;  
System.out.println(numero);
```

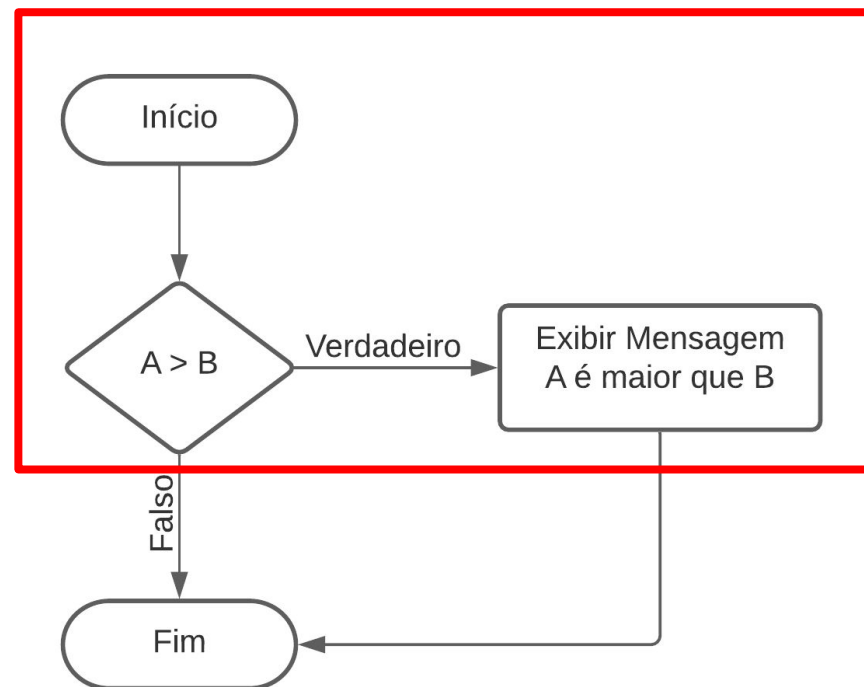
Estruturas condicionais (ou de decisão)

“São estruturas que permitem a tomada de uma decisão sobre qual o caminho a ser escolhido, de acordo com o resultado de uma expressão lógica.” SOUZA et al (2019, p. 151)

Tipos:

- SE-ENTÃO (*if*)
- SE-ENTÃO-SENÃO (*if-else*)
- CASO (*switch case*)

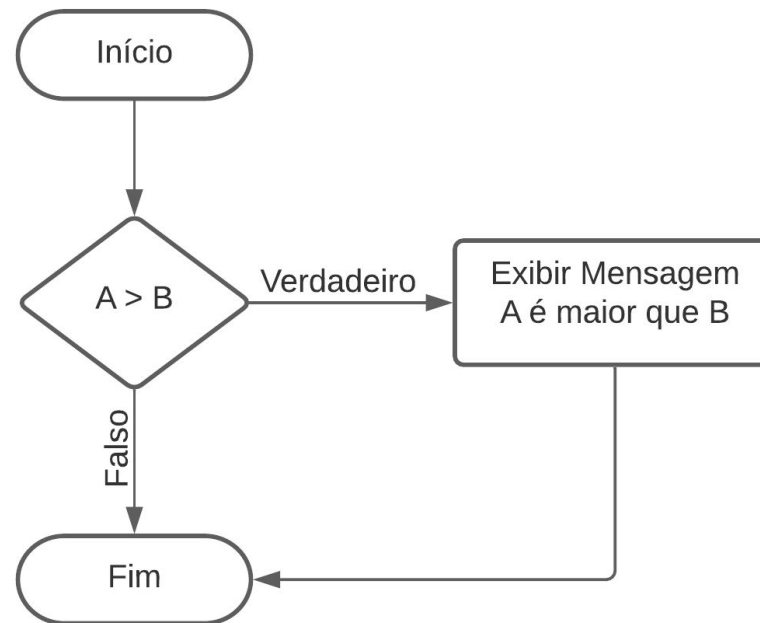
SE-ENTÃO (*IF*)



SE-ENTÃO (*IF*)

```
int numero = 3;  
if(numero == 2) {  
    System.out.println(numero);  
}
```

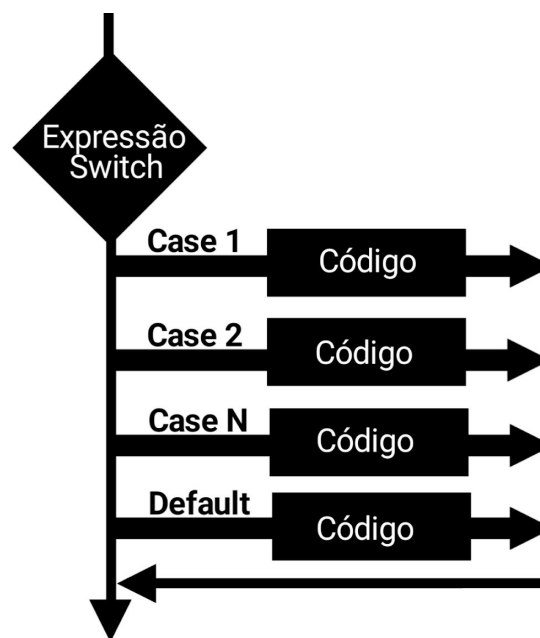
SE-ENTÃO-SENÃO (*IF-ELSE*)



SE-ENTÃO-SENÃO (*IF-ELSE*)

```
int numero = 3;
if(numero == 2) {
    System.out.println(numero);
} else {
    System.out.println("Número não é 2");
}
```

CASO (*SWITCH CASE*)



CASO (*SWITCH CASE*)

```
int numero = 3;

switch (numero) {
    case 1:
        System.out.println(numero);
        break;
    case 2:
        System.out.println(numero);
        break;
    default:
        System.out.println("Número não é 2");
        break;
}
```

CASO (*SWITCH CASE*)

```
int numero = 3;

switch (numero) {
    case 1:
        System.out.println(numero);
        break;
    case 2:
        System.out.println(numero);
        break;
    default:
        System.out.println("Número não é 2");
        break;
}
```

Não esquecer do break!

Referências

SOUZA, Marco A. Furlan de; GOMES, Marcelo M.; SOARES, Marcio V.; CONCILIO, Ricardo. **Algoritmos e lógica de programação:** um texto introdutório para a engenharia. Cengage Learning Brasil, 2019. ISBN 9788522128150. Acesso em: 13 jun. 2023.

SIERRA, Kathy; BATES, Bert. **Use a Cabeça! JAVA.** Alta Books, 2007. ISBN 9788576081739. Acesso em: 12 jun. 2023.

<https://www.youtube.com/playlist?list=PLGxZ4Rq3BOBq0KXHsp5J3PxyFaBIXVs3r>

https://www.youtube.com/playlist?list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR

https://www.youtube.com/playlist?list=PLHz_AreHm4dkqe2aR0tOK74m8SFe-aGsY



Let's *Tech Up Together*