

Programação de Computadores (versão “Java 101”)

André Kishimoto
2024

Sobre este material...

Para cada tópico sobre programação, há um conjunto de slides que apresenta exemplos em pseudocódigo e não usa uma linguagem de programação específica. Por exemplo, `read()` indica entrada de dados (ex. teclado) e `write()` indica saída de dados (ex. saída em tela).

Na sequência, há um conjunto de slides que descreve o tópico apresentado usando a linguagem Java.

Parte do conteúdo e exemplos de código Java foram adaptados do material do prof. Dr. Ivan Carlos Alcântara de Oliveira.

1

Tipos de Dados

Tipos de Dados

- Número inteiro
- Número real
- Lógico
- Caractere
- Texto (conjunto de caracteres)

Tipos de Dados: número inteiro

- `int`
- Representa um dado numérico
- Números inteiros (\mathbb{Z})
- Exemplos:

0

100

-42

2

Tipos de Dados: número real

- float, double (ponto flutuante)
- Representa um dado numérico
- Números reais (R)
- Exemplos:

0.0
1.2
-36.5
4.0



Ponto ao invés de vírgula!

Em algumas linguagens, a diferença entre um valor literal float e double é a letra f após o número.

Por exemplo:

2.5 é um valor literal do tipo double

2.5f** é um valor literal do tipo float**

Tipos de Dados: lógico

- `boolean` (ou `bool`)
- Representa valores verdadeiro ou falso (`true/false`)
- Ligado/desligado, aberto/fechado, 1/0

Tipos de Dados: caractere

- char
- Representa um caractere
- Letras, números e símbolos
- Geralmente indicado entre aspas simples
- Exemplos:

'A'

'1'

'@'

'_'

← **Caractere '1' (representação textual),
diferente do número 1
(representação numérica)**

Tipos de Dados: texto

- `string`
- Representa um conjunto de caracteres
- Geralmente indicado entre aspas duplas
- Exemplos:

`"Brasil"`

`"01"`

`"Qual seu nome?"`

`"@"` ← **Diferente do '@' do slide anterior!**

Tipos de Dados

- Como armazenar dados para uso posterior?



Tipos de Dados

- Como armazenar dados para uso posterior?
- Memória!



1a

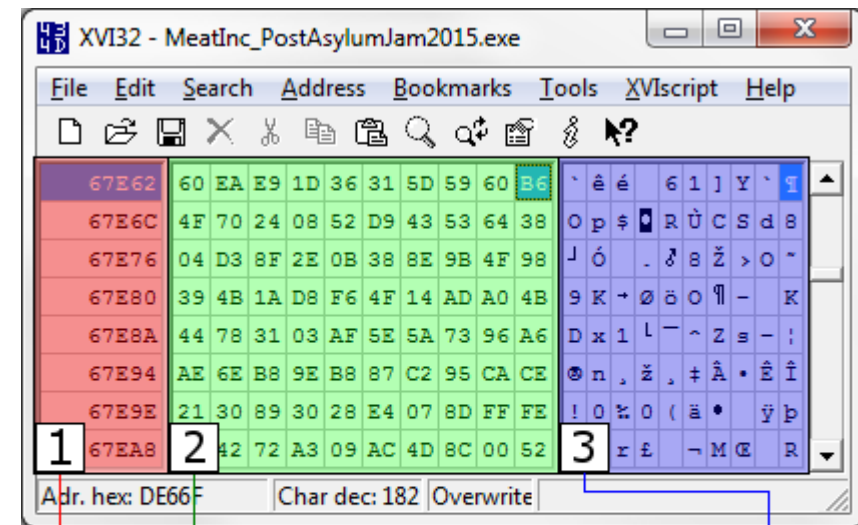
Variáveis

Variáveis

- Usadas para armazenar e acessar dados
- Região (divisão) da memória do computador
 - Possui endereço único
 - Armazena um dado por vez
 - Enquanto programa está sendo executado

Variáveis

- Variáveis == nomes para endereços (posições) da memória
 - Exemplo: escreva 99 no endereço 0AFC:00BB
- variavel = 99;



Endereço de memória do computador.
Divisões da memória do computador.
Dados na memória do computador.

Variáveis

- Declaração de variáveis (assim como outras operações) dependem da linguagem de programação

Fortemente tipada: precisamos definir o tipo de dado

 `int numero = 10;` → **C, C++, C#, Java, ...**

`var numero = 10;` → **Javascript, GML, ...**

`$numero = 10;` → **PHP**

 **Fracamente tipada: não precisamos definir o tipo de dado**

Identificadores: nomenclatura (1/3)

- Todo identificador precisa ter um nome definido pelo programador
 - **Identificador:** classes, variáveis, métodos
- Todo nome deve ser único no escopo do algoritmo
- Todo nome deve começar com uma letra (maiúscula ou minúscula) ou o símbolo _ (underscore)

Identificadores: nomenclatura (2/3)

- Números podem ser usados, mas somente após o primeiro caractere do nome
- Espaços em branco não devem ser usados
- Caracteres ou símbolos especiais também não devem ser usados
 - Algumas linguagens de programação suportam caracteres Unicode

Identificadores: nomenclatura (3/3)

- Geralmente, os nomes ficam restrito ao conjunto [a..z, A..Z, 0..9, _]
- Não é permitido nomear identificadores usando palavras-chave da linguagem de programação
- Usar nomes significativos: `resultadoFinal` é melhor que `rf` (não é uma regra, mas uma boa prática de programação)

Variáveis: nomenclatura

- Nomes válidos para identificadores

- PosicaoX

- _pontos

- Jogador1

- valorFinal

- ValorFinal

Em linguagens “case-sensitive”, letras maiúsculas são diferentes de minúsculas. Nesse caso, esses identificadores são diferentes.

Variáveis: nomenclatura

- Nomes inválidos para identificadores
 - Meu placar ← **Espaço entre “Meu” e “placar”**
 - 2jogadores ← **Começa com número**
 - você ← **‘ê’ é considerado caractere especial**
 - else ← **“else” é uma palavra-chave da linguagem**
 - novo-Menu ← **‘-’ é o operador de subtração**

Variáveis

- Depois que uma variável é criada, podemos alterar seu conteúdo
 - Atribuição de novo valor/conteúdo
- E se quisermos que o conteúdo seja fixo?
 - Exemplo: valor de $\pi = 3.14159265359$

Constantes

- Constantes têm seu valor definido uma única vez
- Seguem nomenclatura de variáveis
- Armazenam um valor por vez
- Somente leitura

1b

Linguagem Java

Java

- Case-sensitive
 - Letras maiúsculas são diferenciadas de letras minúsculas
- Identificadores em Java
 - Sequência de parte dos caracteres Unicode
 - Um nome pode ser composto por letras (minúsculas e/ou maiúsculas), dígitos e os símbolos _ (underscore) e \$ (dólar)
 - Embora o símbolo \$ não seja muito usado...
 - Um nome não pode ser iniciado por um dígito (0..9)
 - Palavra-chave da linguagem Java não pode ser um identificador
 - <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/keywords.html>
 - É recomendável usar a convenção de programação (estilo de escrita) do Java
 - Mas procure sempre adotar o padrão usado pela equipe/projeto!
 - <https://www.oracle.com/java/technologies/javase/codeconventions-contents.html>

Java

- Fortemente tipada
 - Precisamos definir o tipo de dado a ser armazenado
 - Exceção: var
 - Java 10+
 - Somente variáveis locais
 - A variável sempre deve ser declarada e iniciada com um valor
 - O compilador decide o tipo da variável baseado no valor atribuído à variável
 - Uma vez criada, o tipo da variável não pode ser alterado

Java

A linguagem Java não é totalmente Orientada a Objetos, pois possui atributos (variáveis) do tipo primitivo, ou seja, são tipos de dados que não representam classes, mas sim valores básicos.

Category	Types	Size (bits)	Minimum Value	Maximum Value	Precision	Example
Integer	<code>byte</code>	8	-128	127	From +127 to -128	<code>byte b = 65;</code>
	<code>char</code>	16	0	$2^{16}-1$	All Unicode characters ^[1]	<code>char c = 'A';</code> <code>char c = 65;</code>
	<code>short</code>	16	-2^{15}	$2^{15}-1$	From +32,767 to -32,768	<code>short s = 65;</code>
	<code>int</code>	32	-2^{31}	$2^{31}-1$	From +2,147,483,647 to -2,147,483,648	<code>int i = 65;</code>
	<code>long</code>	64	-2^{63}	$2^{63}-1$	From +9,223,372,036,854,775,807 to -9,223,372,036,854,775,808	<code>long l = 65L;</code>
Floating-point	<code>float</code>	32	2^{-149}	$(2 \cdot 2^{-23}) \cdot 2^{127}$	From 3.402,823,5 E+38 to 1.4 E-45	<code>float f = 65f;</code>
	<code>double</code>	64	2^{-1074}	$(2 \cdot 2^{-52}) \cdot 2^{1023}$	From 1.797,693,134,862,315,7 E+308 to 4.9 E-324	<code>double d = 65.55;</code>
Other	<code>boolean</code>	--	--	--	false, true	<code>boolean b = true;</code>
	<code>void</code>	--	--	--	--	--

https://en.wikibooks.org/wiki/Java_Programming/Primitive_Types

Java

Variáveis em Java podem ser do tipo:

- **Primitivas:** podem ser de um dos tipos de dados apresentados no slide anterior.
 - Armazenadas na memória stack.
 - Cópia: uma segunda variável apenas armazena o mesmo valor, mas o endereço de memória é diferente.
 - Alteração da segunda variável: não afeta a variável original.
- **Referências:** usadas para referenciar um objeto.
 - A referência é armazenada na memória stack.
 - O objeto original é armazenado na memória heap.
 - Referência: uma segunda variável (stack) aponta para o mesmo objeto da memória heap.
 - Alteração da segunda variável: altera a variável original.

Java variáveis

```
<tipo> <nomeDaVariavel>;  
<tipo> <nomeDaVariavel> = <valorInicial>;
```

// Exemplo

```
String disciplina = "Estrutura de Dados I";
```



The diagram consists of a curved arrow pointing from the text 'Estrutura de Dados I' in the code line above to the text 'Associa/atribui o valor "Estrutura de Dados I" na variável disciplina' in the text block below. A straight arrow points from the text 'Associa/atribui o valor "Estrutura de Dados I" na variável disciplina' to the variable name 'disciplina' in the code line above.

Associa/atribui o valor "Estrutura de Dados I" na variável disciplina, que é do tipo String.

***** OBSERVAÇÃO: String não é um tipo de dado primitivo em Java! *****

Java variáveis

```
<tipo> <nomeDaVariavel>;  
<tipo> <nomeDaVariavel> = <valorInicial>;
```

// Exemplos

```
String disciplina = "Estrutura de Dados I";  
int numero1;  
int numero2 = 99;  
bool jogando;  
float nota = 7.75f;  
char letra = 'Y';
```

Java constantes

```
static final <tipo> <NOME_DA_CONSTANTE> = <valorFixo>;
```

// Exemplos

```
static final double PI = 3.14159265359;
```

```
static final String CODIGO_DISCIPLINA = "ENEX50328";
```

1c

Exemplo Java

Estrutura básica de um programa Java

```
[import ...]
```

```
[public] class Identificador {
```

```
    public static void main(String[] args) {  
        // Constantes, variáveis e comandos locais.  
    }
```

```
}
```


Estrutura básica de um programa Java

Quando o código faz uso de bibliotecas adicionais, usamos o comando `import <nome da biblioteca>`.

`[import ...]`

Todo programa Java precisa ter uma classe que contém a `main()`.

`[public] class Identificador {`

Indicamos o ponto de partida (inicial) do projeto.

```
public static void main(String[] args) {  
    // Constantes, variáveis e comandos locais.  
}
```


```
}
```

Exemplo 1: Hello, World!

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

Exemplo 1: Hello, World!

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



Imprime a mensagem "Hello, World!" (sem aspas) no terminal do sistema, pulando uma linha após a string "Hello, World!".
System.out.print(<string>); não pula uma linha.

Exemplo 2: Leitura/escrita de variáveis

Exemplo:

Entrada, saída e tipos primitivos (ou não) de dados.

Programa que solicita ao usuário alguns dados: nome, idade, peso e altura, depois imprime uma frase contendo os valores lidos.

Exemplo 2: Leitura/escrita de variáveis

A entrada de dados via console pode ser feita com a classe `java.util.Scanner`

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Scanner.html>

Tipo de dado	Método Scanner	Funcionamento
boolean	nextBoolean()	Varre o próximo token da entrada em um valor booleano e retorna esse valor.
byte	nextByte()	Verifica o próximo token da entrada como um byte.
double	nextDouble()	Verifica o próximo token da entrada como um double.
float	nextFloat()	Verifica o próximo token da entrada como um float.
int	nextInt()	Verifica o próximo token da entrada como um int.
String	nextLine()	realiza a leitura de uma string com espaço em branco e finaliza com “return” ou enter.
long	nextLong()	Verifica o próximo token da entrada como um long.
short	nextShort()	Verifica o próximo token da entrada como um short.
String	next()	Localiza e retorna o próximo token completo deste scanner.

Exemplo 2: Leitura/escrita de variáveis

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Digite seu nome: ");
        String nome = s.nextLine();
        System.out.print("Digite sua idade: ");
        int idade = s.nextInt();
        System.out.print("Digite seu peso: ");
        double peso = s.nextDouble();
        System.out.print("Digite sua altura: ");
        float altura = s.nextFloat();
        System.out.println("\n" + nome + " tem " + idade + " anos, pesa " + peso + " Kg, mede "
+ altura + " m.");
    }
}
```

Exemplo 3: Funções matemáticas

A classe `Math` já está no pacote (package) `java.lang`.

Esse pacote já é automaticamente importado, então, não precisamos importar, como foi feito com a classe `Scanner` do exemplo anterior.

Exemplo 3: Funções matemáticas

```
class Main {  
    public static void main(String[] args) {  
        int n1 = Math.abs(80);  
        System.out.println("Valor absoluto de 80 é " + n1);  
  
        int n2 = Math.abs(-60);  
        System.out.println("Valor absoluto de -60 é " + n2);  
  
        double n3 = Math.sqrt(36.0);  
        System.out.println("Raiz quadrada de 36.0 é " + n3);  
  
        double n4 = Math.cbrt(8.0);  
        System.out.println("Raiz cúbica de 8.0 é " + n4);  
  
        int n5 = Math.max(15, 80);  
        System.out.println("Valor máximo é " + n5);  
  
        int n6 = Math.min(15, 80);  
        System.out.println("Valor mínimo é " + n6);  
  
        // Continua no próximo slide...
```


Exemplo 3: Funções matemáticas

```
// ...continuação do slide anterior.
```

```
double n7 = Math.ceil(6.34);  
System.out.println("Teto de 6.34 é " + n7);
```

```
double n8 = Math.floor(6.34);  
System.out.println("Piso de 6.34 é " + n8);
```

```
double n9 = Math.round(22.445);  
System.out.println("Valor arredondado de 22.445 é " + n9);
```

```
double n10 = Math.round(22.545);  
System.out.println("Valor arredondado de 22.545 é " + n10);
```

```
double n11 = Math.pow(2.0, 3.0);  
System.out.println("Potência de 2 elevado a 3 é " + n11);
```

```
double n12 = Math.random();  
System.out.println("Um valor aleatório a partir da data e hora atual do sistema é " + n12);
```

```
System.out.println("Valor de pi é " + Math.PI);
```

```
}  
}
```

2

Operadores

Operadores

- Atribuição
- Aritméticos
- Relacionais
- Lógicos

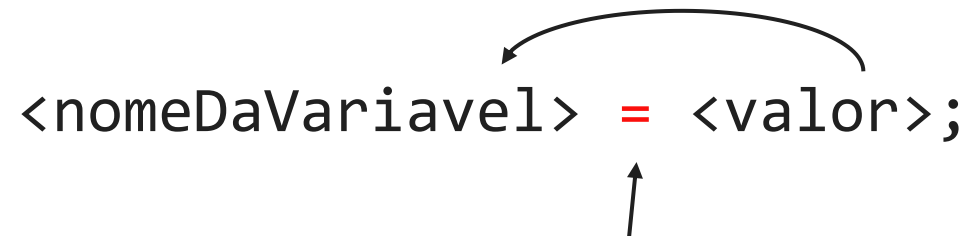
2a

Operador de Atribuição

Operador de atribuição

- Atribuir valor/conteúdo para uma variável
- Símbolo de igualdade (=)
 - Alguns autores usam a seta ←

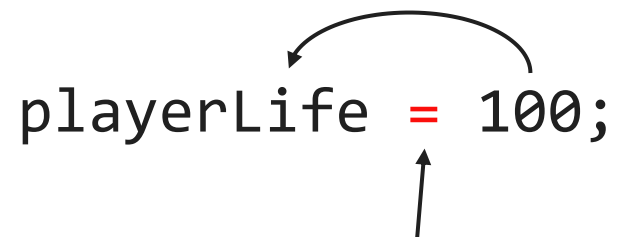
`<nomeDaVariavel> = <valor>;`



**Associa/atribui <valor> na variável
<nomeDaVariavel>**

Operador de atribuição

Exemplo:



```
playerLife = 100;
```

“playerLife recebe o valor 100”

“o valor 100 é associado ou atribuído à playerLife”



Operadores Aritméticos

Operadores aritméticos

- Realizar cálculos matemáticos com dados numéricos (int/float/double)
- Unários ou binários
 - Unário: requer apenas um operando
 - Binário: requer dois operandos

Operadores aritméticos

Operador	Operação	Tipo
+	Adição	Binário
-	Subtração	Binário
*	Multiplicação	Binário
/	Divisão	Binário
%	Módulo (resto da divisão)	Binário
-	Inversão de sinal	Unário
++	Adição unária (incremento)	Unário
--	Subtração unária (decremento)	Unário

Operadores aritméticos

Exemplos:

adicao = 5 + 10;

← **adicao recebe valor 15**

subtracao = 444 - 33;

← **subtracao recebe valor 411**

multiplicacao = adicao * 2;

← **multiplicacao recebe valor 30**

divisaoInt = 31 / 6;

← **divisaoInt recebe valor 5**

divisaoFloat = 31.0 / 6.0;

← **divisaoFloat recebe valor 5.166...**

modulo = 31 % 6;

← **modulo recebe valor 1 (resto da divisão)**

Operadores aritméticos

Exemplos:

adicao = 5 + 10; ← **adicao recebe valor 15**

inversao = -adicao; ← **inversao recebe valor -15**

adicao++; ← **adicao passa a valer 16**

++adicao; ← **adicao passa a valer 17**

adicao--; ← **adicao passa a valer 16**

--adicao; ← **adicao passa a valer 15**

Operadores aritméticos


- Adição unária (++)
- Antes ou depois da variável
adicao++;
++adicao;
- Diferença?

Operadores aritméticos

number possui valor 1.

n1 possui valor 0.

```
int number = 0;
int n1 = number++;
write("number: " + number + ", n1: " + n1);
number = 0;
int n2 = ++number;
write("number: " + number + ", n2: " + n2);
```



Operadores aritméticos


```
int number = 0;  
int n1 = number++;  
write("number: " + number + ", n1: " + n1);  
number = 0;  
int n2 = ++number;  
write("number: " + number + ", n2: " + n2);
```

 **number possui valor 1.**

 **n2 possui valor 1.**

Operadores aritméticos

**Atribuição do valor da variável `number` para `n1` ocorre primeiro. Logo, `n1 = 0`.
Em seguida, `number` é incrementada.
Assim, `number = 1`.**




```
int number = 0;
int n1 = number++;
write("number: " + number + ", n1: " + n1);
number = 0;
int n2 = ++number;
write("number: " + number + ", n2: " + n2);
```

Operadores aritméticos

```
int number = 0;
int n1 = number++;
write("number: " + number + ", n1: " + n1);
number = 0;
int n2 = ++number;
write("number: " + number + ", n2: " + n2);
```

Variável number é incrementada primeiro, ou seja, number = 1. Em seguida, ocorre atribuição de number para n2. Portanto, n2 = 1.

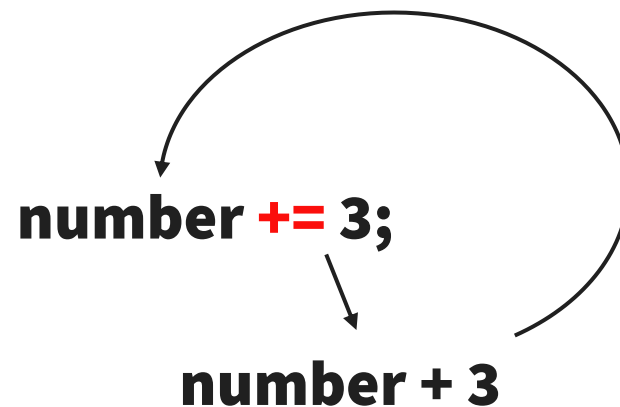


Operadores aritméticos

- Subtração unária (--)
- Antes ou depois da variável
adicao--;
--adicao;
- Mesmo comportamento da adição unária

Operadores aritméticos com atribuição

**// Assumindo que
int number = 2;**



**// Isto é...
number = number + 3;**

**// Logo...
number = 5;**

Operadores aritméticos com atribuição

Exemplos:

number += 3;

← **number = number + 3;**

number -= 1;

← **number = number - 1;**

number *= 5;

← **number = number * 5;**

number /= 4;

← **number = number / 4;**

number %= 2;

← **number = number % 2;**

2c

Operadores Relacionais

Operadores relacionais

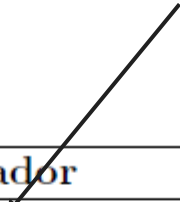
- Comparar valores
- Essenciais para tomada de decisões
 - “Personagem A está à esquerda ou direita do personagem B?”
 - Comparação da posição X dos personagens
- Binários
- Sempre resultam em `true` ou `false`
 - `boolean`

Operadores relacionais

Operador	Operação
==	Igual a
!=	Diferente de
<	Menor que
<=	Menor que ou igual a
>	Maior que
>=	Maior que ou igual a

Operadores relacionais

Repare que a comparação usa dois sinais de igualdade (==), enquanto que a atribuição usa somente um sinal (=).



Operador	Operação
==	Igual a
!=	Diferente de
<	Menor que
<=	Menor que ou igual a
>	Maior que
>=	Maior que ou igual a

Operadores relacionais

Exemplos:

```
// Assumindo as variáveis  
int x = 10;  
int y = 20;  
int z = 10;
```

```
// E também as variáveis  
boolean equalTo, notEqualTo;  
boolean lessThan, lessThanOrEqualTo;  
boolean greaterThan, greaterThanOrEqualTo;
```

```
equalTo = (x == y);  
           └──  
           false
```

← equalTo recebe o valor false,
pois x não é igual a y.

```
notEqualTo = (x != y);  
             └──  
             true
```

← notEqualTo recebe o valor true,
pois x é diferente de y.


Operadores relacionais

Exemplos:

```
// Assumindo as variáveis  
int x = 10;  
int y = 20;  
int z = 10;
```

```
// E também as variáveis  
boolean equalTo, notEqualTo;  
boolean lessThan, lessThanOrEqualTo;  
boolean greaterThan, greaterThanOrEqualTo;
```

```
lessThan = (x < y);
```



true

← lessThan recebe o valor true,
pois x é menor que y.

```
lessThanOrEqualTo = (y <= x);
```



false

← lessThanOrEqualTo recebe o
valor false, pois y não é
menor que ou igual a x.

Operadores relacionais

Exemplos:

```
// Assumindo as variáveis  
int x = 10;  
int y = 20;  
int z = 10;
```


```
// E também as variáveis  
boolean equalTo, notEqualTo;  
boolean lessThan, lessThanOrEqualTo;  
boolean greaterThan, greaterThanOrEqualTo;
```

```
greaterThan = (x > y);
```


false

← greaterThan recebe o valor false,
pois x não é maior que y.

```
greaterThanOrEqualTo = (x >= z);
```


true

← greaterThanOrEqualTo recebe
o valor true, pois x é igual a z.

Operadores relacionais

Operadores “igual a” e “diferente de” também pode ser usadas para boolean

equalTo = (x == y); ← **equalTo** armazena o valor **false**.

notEqualTo = (x != y); ← **notEqualTo** armazena o valor **true**.

```
boolean test1 = (equalTo == notEqualTo);
```



```
boolean test2 = (equalTo != notEqualTo);
```


true

Operadores relacionais


Em algumas linguagens de programação, as comparações de igualdade e diferença também se aplicam a strings e caracteres

```
String text = "aula";
```

```
boolean test3 = (text == "aula");
```



```
boolean test4 = (text != "aula");
```



2d

Operadores Lógicos

Operadores lógicos

- Operandos são valores ou expressões lógicas
 - boolean (true/false)
- Três operações
 - AND (e), binária
 - OR (ou), binária
 - NOT (não), unária
- Sempre resultam em true ou false

Operadores lógicos

- Tabela-verdade
 - Combinação de possíveis valores de cada expressão lógica
 - Resultado gerado por cada operação e combinação de valores

Operadores lógicos: AND

- Os dois operandos precisam ser true para que resultado seja true
- Caso contrário, resultado é false

<code>A AND B</code>	<code>A == true</code>	<code>A == false</code>
<code>B == true</code>	true	false
<code>B == false</code>	false	false

Operadores lógicos: AND

- A = “Personagem está correndo”
- B = “Personagem possui habilidade de voar”
- Regra: personagem só pode voar caso esteja correndo E possua a habilidade de voar

A AND B	A == true	A == false
B == true	true	false
B == false	false	false

**Situação em que as condições são
aceitas pela regra.**

Operadores lógicos: AND

- Substituindo `false` por 0 e `true` por 1, AND é similar à multiplicação

A * B	A == 1	A == 0
B == 1	1 * 1 = 1	0 * 1 = 0
B == 0	1 * 0 = 0	0 * 0 = 0

Operadores lógicos: OR

- Pelo menos um dos operandos precisa ser `true` para que resultado seja `true`
- Resultado é `false` somente quando os dois operandos são `false`

<code>A OR B</code>	<code>A == true</code>	<code>A == false</code>
<code>B == true</code>	<code>true</code>	<code>true</code>
<code>B == false</code>	<code>true</code>	<code>false</code>

Operadores lógicos: OR

- A = “Personagem está correndo”
- B = “Personagem possui habilidade de voar”
- Regra: personagem só pode voar caso esteja correndo E possua a habilidade de voar

A OR B	A == true	A == false
B == true	true	true
B == false	true	false

**Situações em que as condições são aceitas pela regra.
Usar OR nesse caso está errado.**

Operadores lógicos: OR

- A = “Clique do botão esquerdo do mouse”
- B = “Barra de espaço apertada”
- Funcionalidade: ação principal do personagem é acionada pelo clique do botão esquerdo do mouse ou barra de espaço

A OR B	A == true	A == false
B == true	true	true
B == false	true	false

 Situações em que a funcionalidade é ativada.

Operadores lógicos: OR

- Substituindo `false` por 0 e `true` por 1, OR é (quase) similar à adição

<code>A + B</code>	<code>A == 1</code>	<code>A == 0</code>
<code>B == 1</code>	<code>1 + 1 = 1</code>	<code>0 + 1 = 1</code>
<code>B == 0</code>	<code>1 + 0 = 1</code>	<code>0 + 0 = 0</code>

Operadores lógicos: NOT

- Negar (inverter) uma condição/expressão.
- Caso operando seja `true`, terá seu valor invertido para `false` e vice-versa

A	NOT A
<code>A == true</code>	<code>false</code>
<code>A == false</code>	<code>true</code>

Operadores lógicos: NOT

- A = “Personagem está correndo”
- NOT A = “Personagem não está correndo”
- B = “Barra de espaço apertada”
- NOT B = “Barra de espaço não está apertada”

A	NOT A
<code>A == true</code>	<code>false</code>
<code>A == false</code>	<code>true</code>

2e

Linguagem Java

Java

- Operadores
 - Atribuição
 - Aritméticos
 - Aritméticos com atribuição
 - Relacionais
- Mesmos símbolos que foram apresentados anteriormente

Java

- Operadores lógicos
 - AND → && (dois “e comercial” juntos)
 - OR → | | (duas barras verticais juntas)
 - NOT → ! (ponto de exclamação)

Java

Operadores	Comentários
()	agrupamento
++, --, +, -	Incremento, decremento e operadores unários
*, /, %	Multiplicação, divisão e resto
+, -	Soma e subtração
<, <=, >, >=	Operadores relacionais: menor, menor ou igual, maior, maior ou igual
==, !=	Operadores relacionais: Igualdade, desigualdade
&&	Operador lógico E
!!	Operador lógico OU
!	Operador Lógico Negação
(tipo) variavel	Conversão da valor da variável para o tipo especificado em ()
? :	Operador condicional ternário
=, +=, -=, *=, /=, %=	Operadores de atribuição

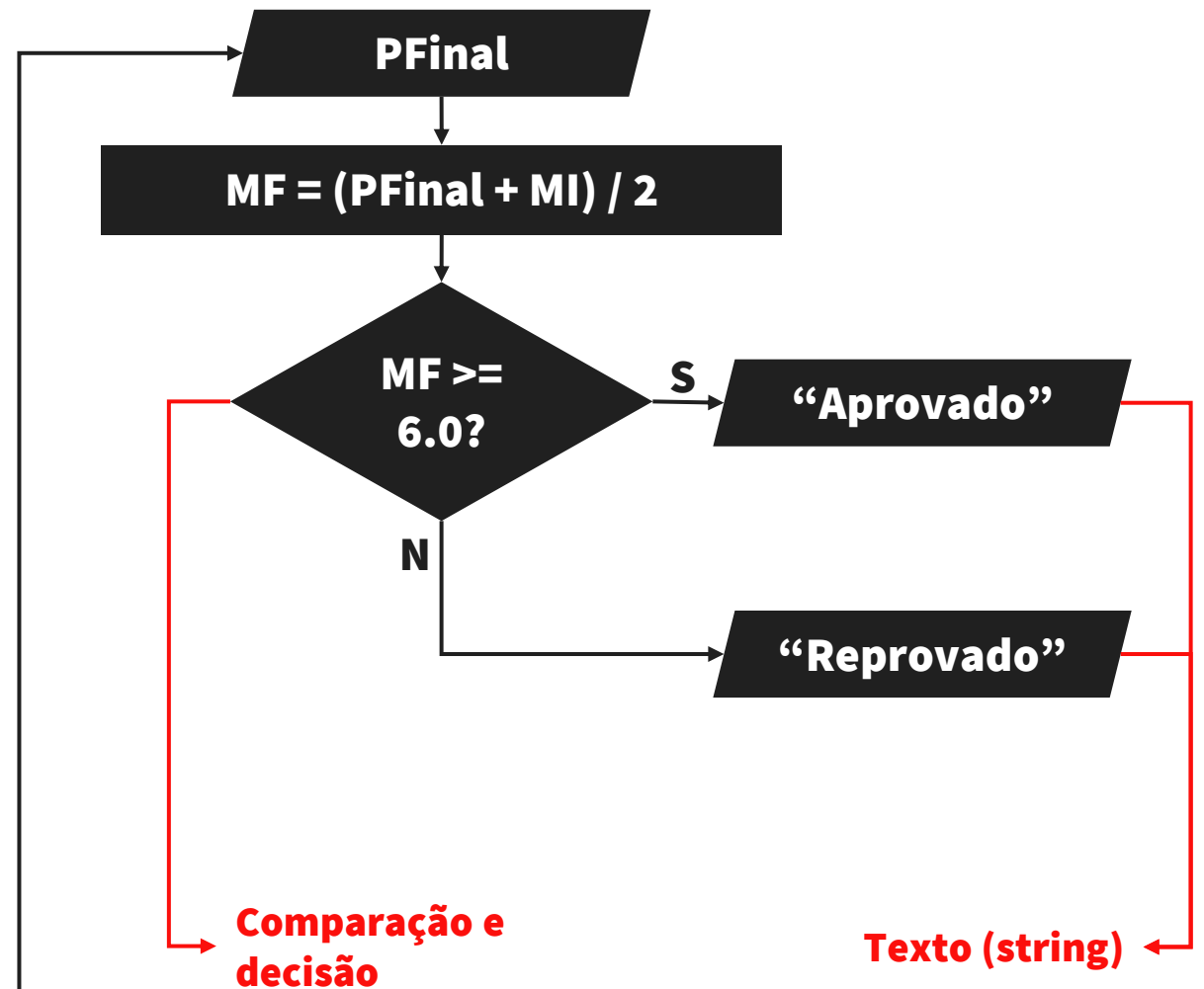
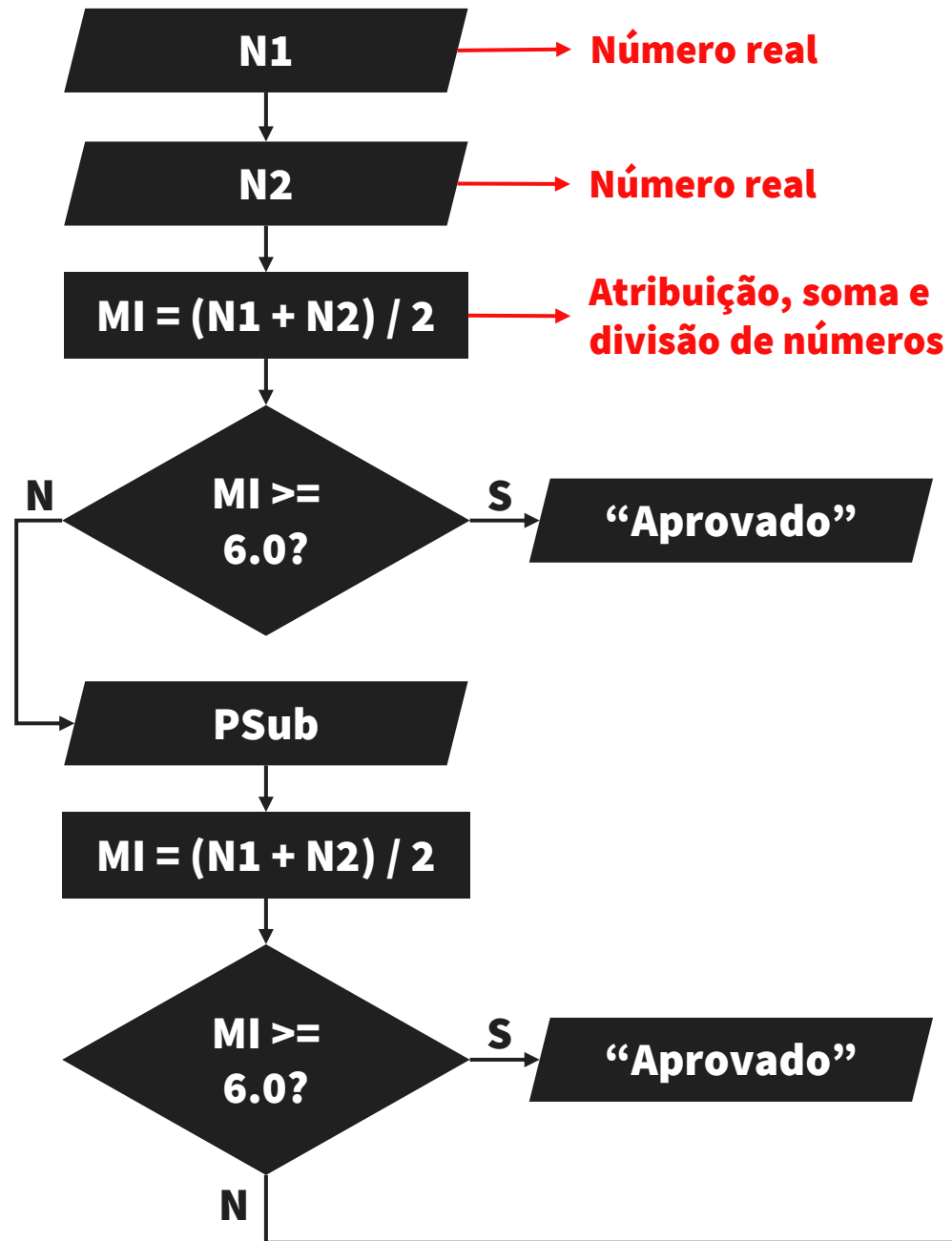
2f

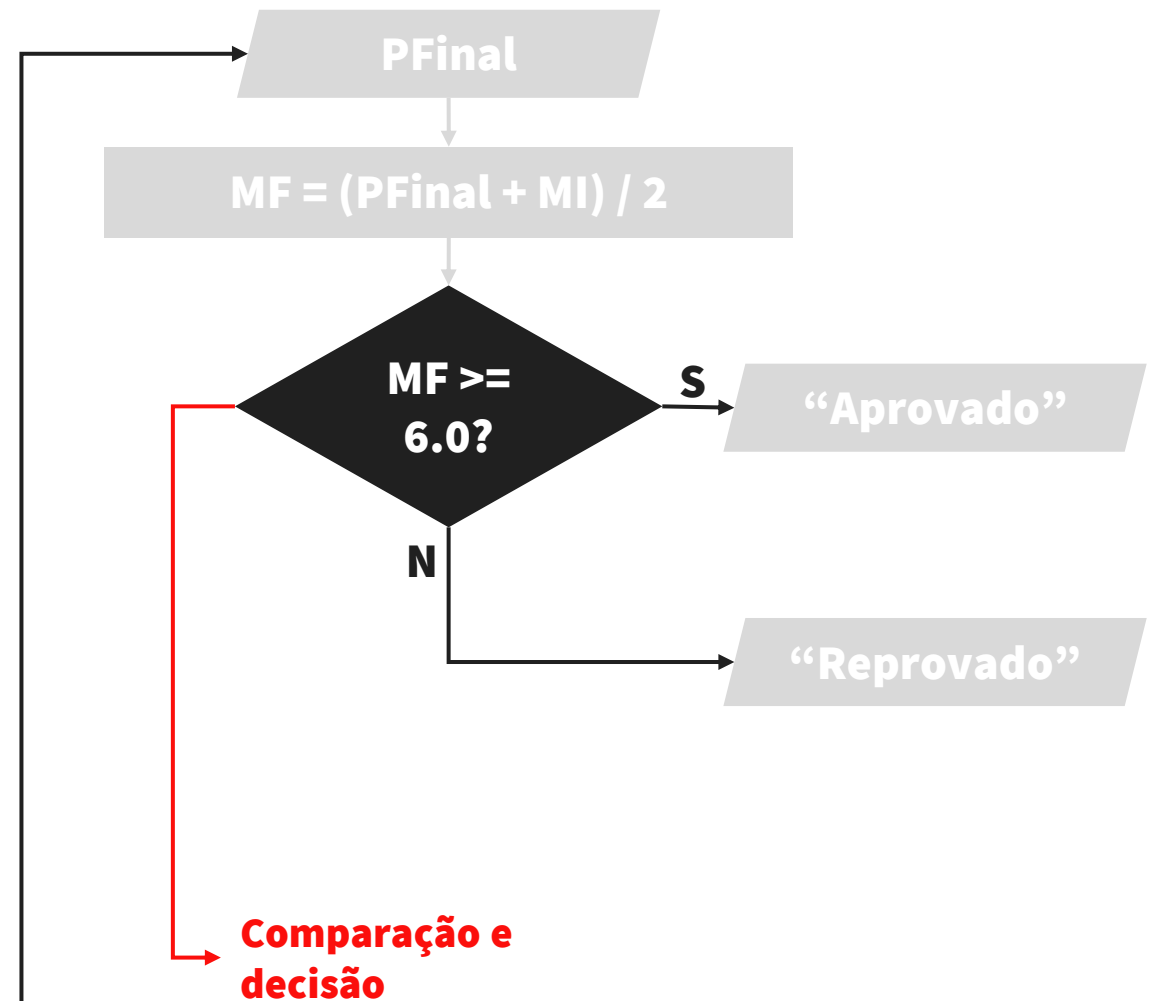
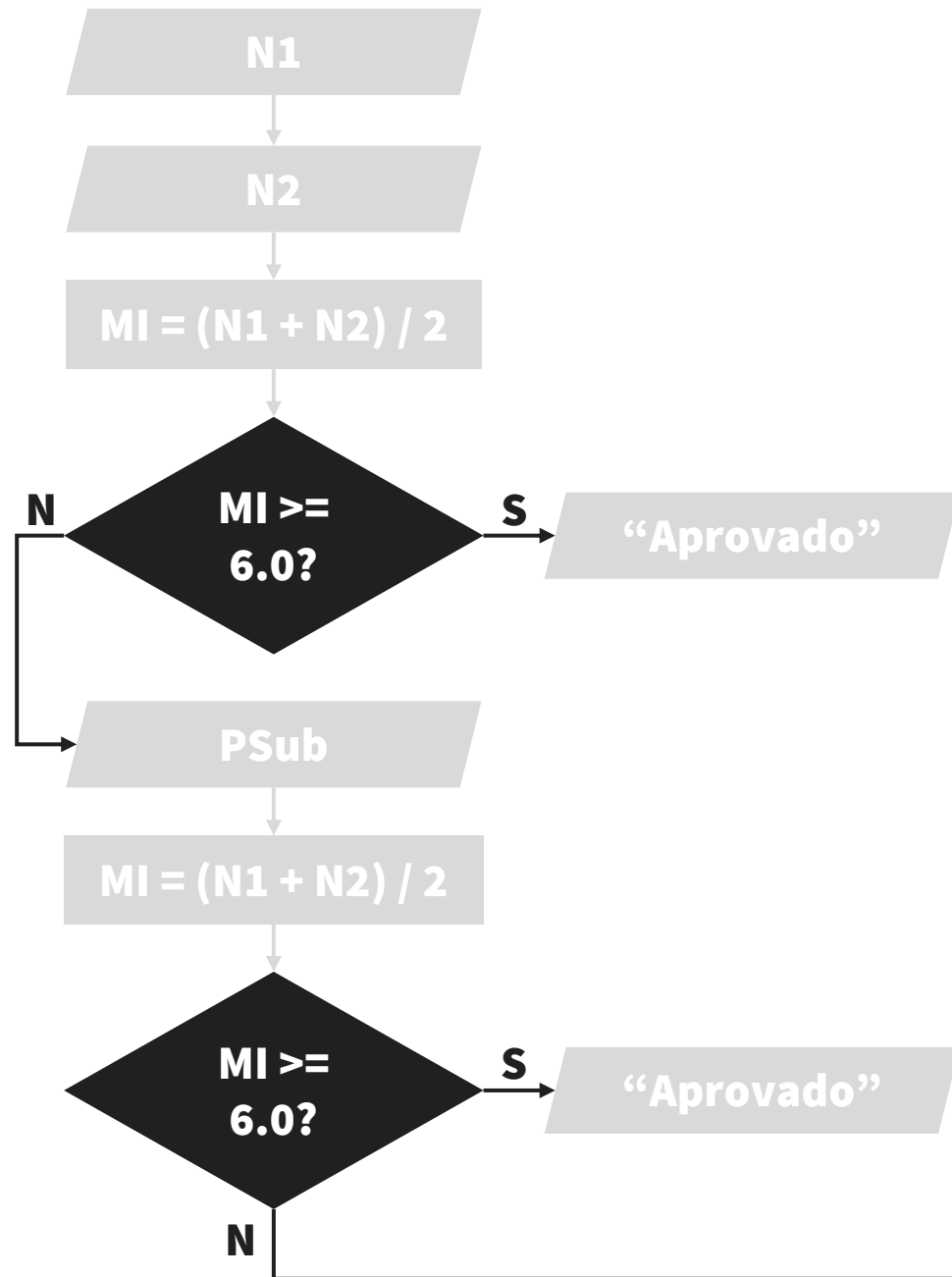
Exemplo Java

Exemplo Java

Reproduzir todos os exemplos dos slides anteriores.

Estruturas de Decisão





Estruturas de decisão

- Decidindo o que fazer
- Decisão simples
- Decisão composta
- Decisão encadeada
- Decisão e operadores lógicos
- Decisão e condições
- Switch-case

3a

Decidindo o que fazer

Decidindo o que fazer

- Depois de uma aula cansativa e transporte público lotado, você finalmente volta pra casa...
- ...acende a luz da sala e...
- ...puf! A lâmpada queima.

Decidindo o que fazer

- Quando isso acontece, você sabe que precisa trocar a lâmpada para iluminar a sala.
- Quais decisões são feitas neste exemplo?

Decidindo o que fazer

Vamos supor duas situações:

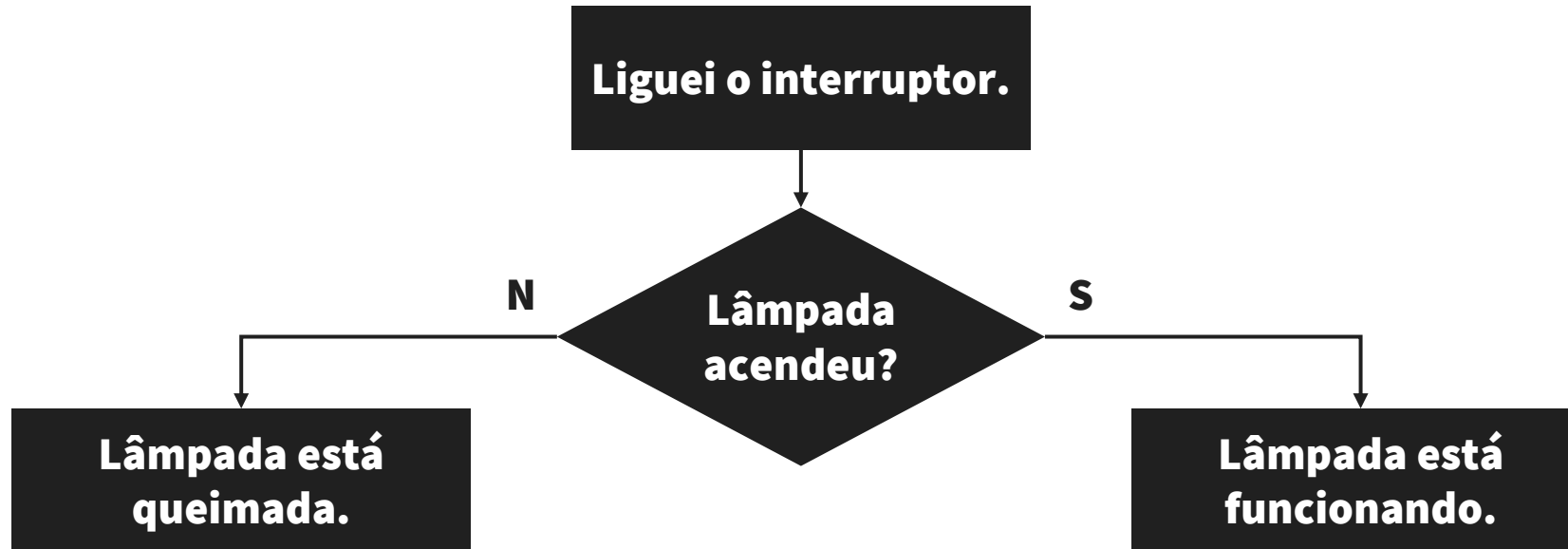


Usar o interruptor

**Trocar a lâmpada
se estiver queimada**

Decidindo o que fazer

Usar o interruptor → **Define se a lâmpada está ou não funcionando.**



Decidindo o que fazer

**Trocar a lâmpada
se estiver queimada** →

**Define se precisamos ou não
trocar a lâmpada.**



Decidindo o que fazer

**Estruturas de
decisão**



**Definem se um conjunto de
instruções deve ou não ser
executado pelo computador.**

A red square logo with the white text "3b" inside.

Decisão Simples

Decisão simples

se <condição> **então**
 <instruções quando resultado da condição é verdadeira>
fim_se

if (<condição>)
{
 <instruções quando resultado da condição é verdadeira>
}

Decisão simples

se <condição> então
 <instruções quando resultado da condição é verdadeira>
fim_se

if (<condição>)
{
 <instruções quando resultado da condição é verdadeira>
}

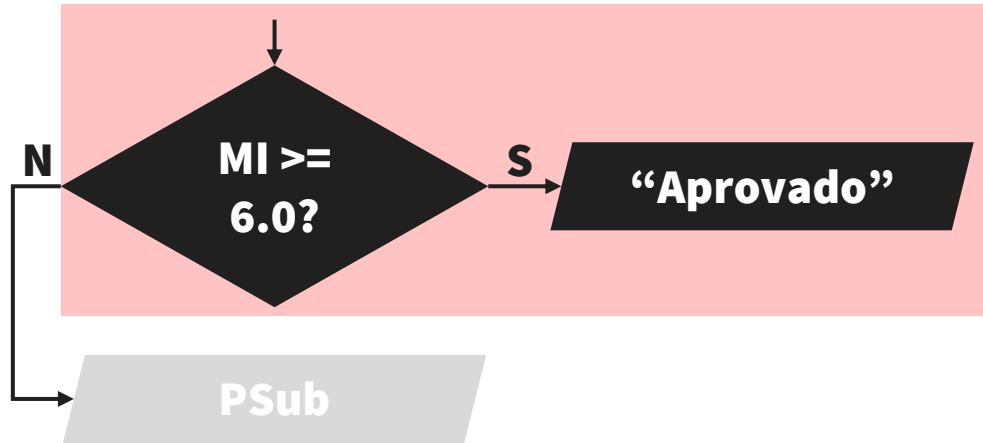
Decisão simples

Expressão que retorna true ou false



```
if (<condição>)  
{  
    <instruções quando resultado da condição é verdadeira>  
}
```

Decisão simples



```
if (MI >= 6.0)  
{  
    write("Aprovado");  
}
```

3c

Decisão Composta

Decisão composta

Expressão que retorna true ou false



if (<condição>)

{

<instruções quando resultado da condição é verdadeira>

}

else

Basicamente:

!condição (negação da condição do if)

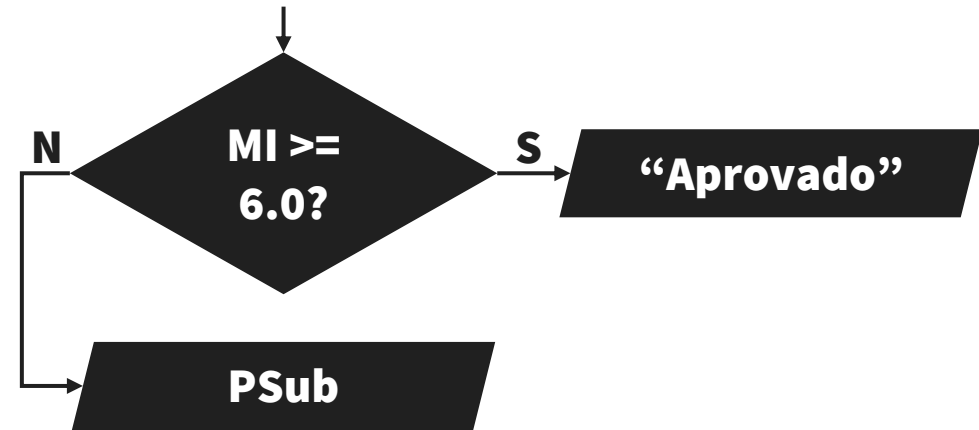


{

<instruções quando resultado da condição é falsa>

}

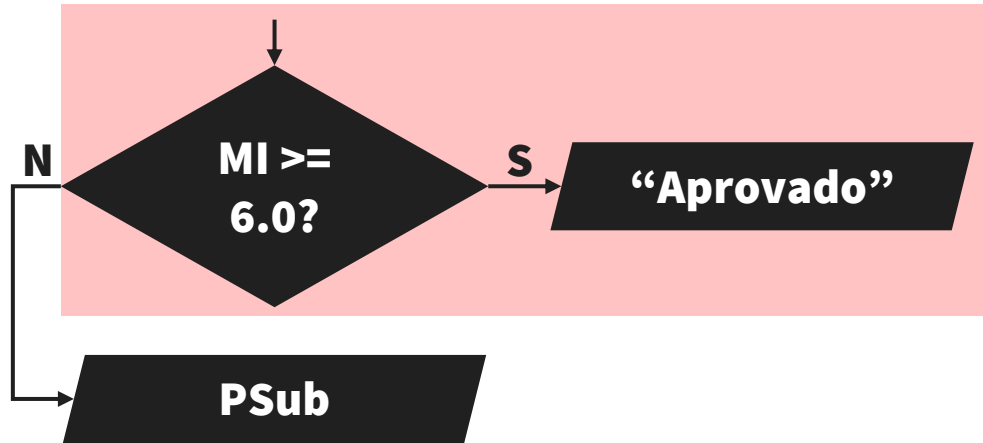
Decisão composta



```
if (MI >= 6.0)
{
    write(“Aprovado”);
}
else ← Isto é, (MI < 6.0)
        Ou ainda, !(MI >= 6.0)
{
    read(PSub);
}
```

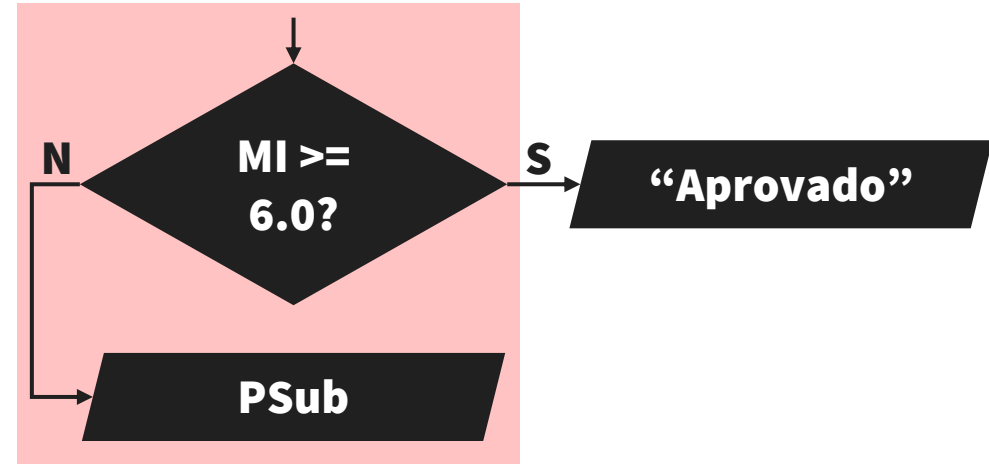
Decisão composta

```
if (MI >= 6.0)
{
    write(“Aprovado”);
}
else
{
    read(PSub);
}
```



Decisão composta

```
if (MI >= 6.0)
{
    write(“Aprovado”);
}
else
{
    read(PSub);
}
```



3d

Decisão Encadeada

Decisão encadeada

- Verificar uma condição baseada no resultado de outra condição já verificada.
- Uma estrutura de decisão dentro de outra.

Decisão encadeada

```
if (<condição1>)  
{  
    <instruções quando resultado da condição1 é verdadeira>  
    if (<condição2>)  
    {  
        <instruções quando condição2 é verdadeira e condição1 é verdadeira>  
    }  
    else ← Opcional  
    {  
        <instruções quando condição2 é falsa e condição1 é verdadeira>  
    }  
}  
else ← Opcional  
{  
    <instruções quando resultado da condição1 é falsa>  
}
```

Decisão encadeada

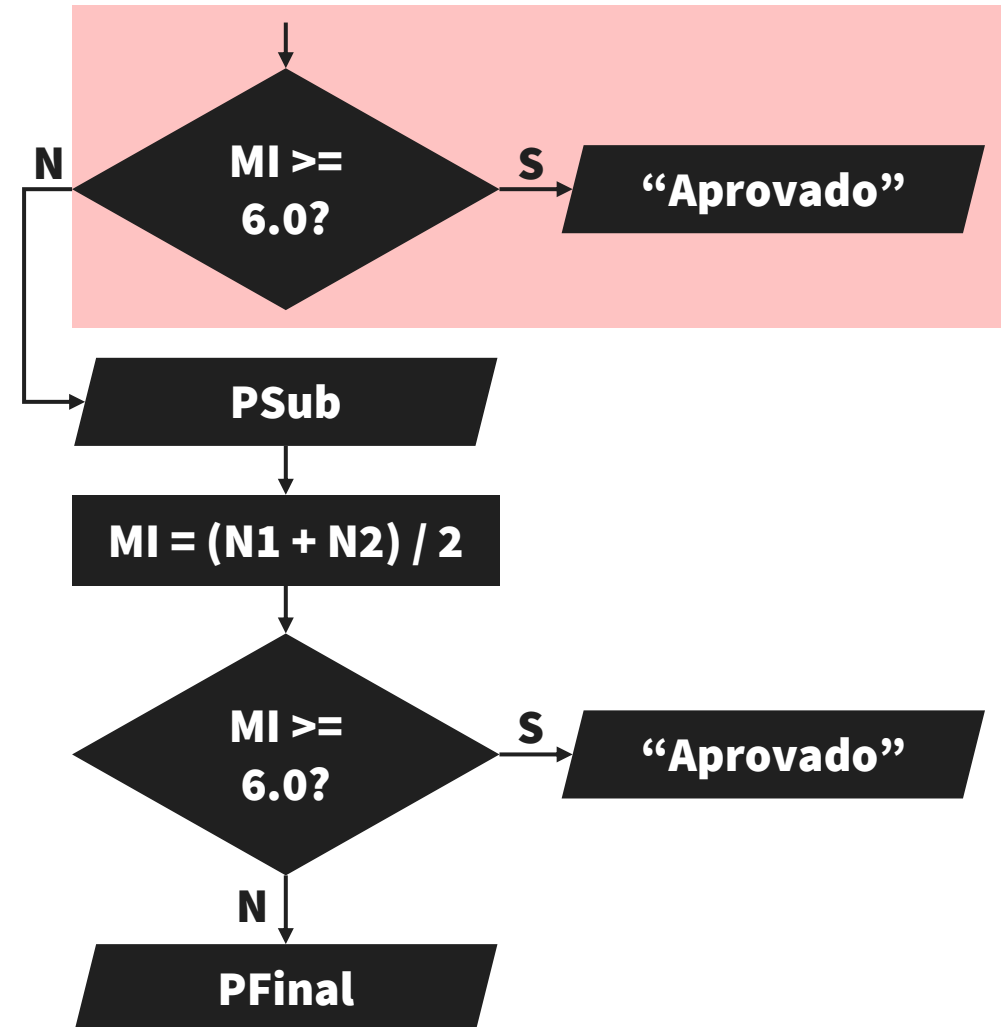
```
if (<condição1>)  
{  
    <instruções quando resultado da condição1 é verdadeira>  
}  
else  
{  
    <instruções quando resultado da condição1 é falsa>  
    if (<condição2>)  
    {  
        <instruções quando condição2 é verdadeira e condição1 é falsa>  
    }  
    else ← Opcional  
    {  
        <instruções quando condição2 é falsa e condição1 é falsa>  
    }  
}
```

Decisão encadeada

```
if (<condição1>)  
{  
    <instruções quando resultado da condição1 é verdadeira>  
}  
else if (<condição2>)  
{  
    <instruções quando condição2 é verdadeira e condição 1 é falsa>  
}  
else if (<condiçãoN>)  
{  
    <instruções quando condiçãoN é verdadeira e condições anteriores são falsas>  
}  
else ← Opcional  
{  
    <instruções quando todas as condições anteriores são falsas>  
}
```

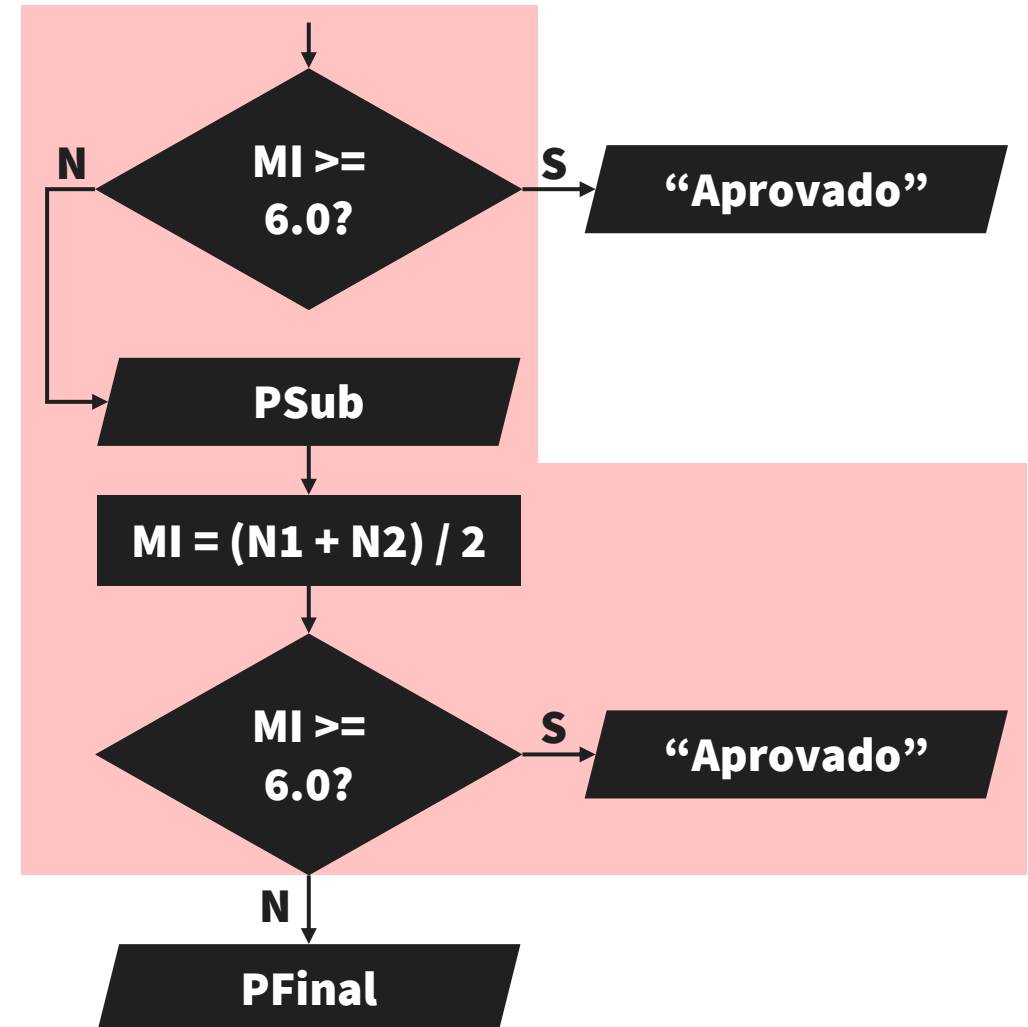

Decisão encadeada

```
if (MI >= 6.0)
{
    write(“Aprovado”);
}
else
{
    read(PSub);
    MI = (N1 + N2) / 2;
    if (MI >= 6.0)
    {
        write(“Aprovado”);
    }
    else
    {
        read(PFinal);
        // ...
    }
}
```



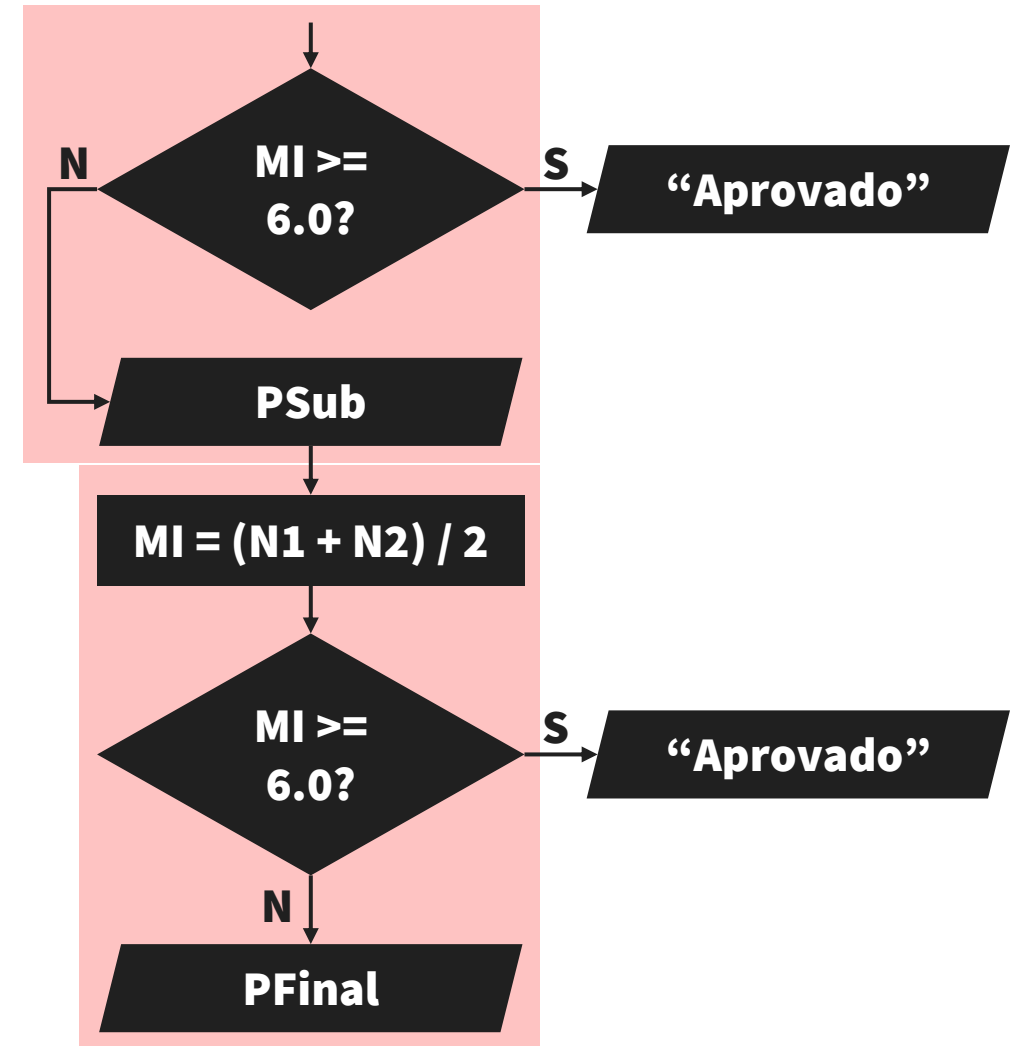
Decisão encadeada

```
if (MI >= 6.0)
{
    write("Aprovado");
}
else
{
    read(PSub);
    MI = (N1 + N2) / 2;
    if (MI >= 6.0)
    {
        write("Aprovado");
    }
    else
    {
        read(PFinal);
        // ...
    }
}
```



Decisão encadeada

```
if (MI >= 6.0)
{
    write("Aprovado");
}
else
{
    read(PSub);
    MI = (N1 + N2) / 2;
    if (MI >= 6.0)
    {
        write("Aprovado");
    }
    else
    {
        read(PFinal);
        // ...
    }
}
```



3e

Decisão e Operadores Lógicos

Decisão e operadores lógicos

- Podemos usar operadores lógicos para compor expressões lógicas mais complexas nas estruturas de decisão
 - && (AND)
 - || (OR)
 - ! (NOT)

Decisão e operadores lógicos

Expressão lógica usando
operadores lógicos.



```
if (<condição>)  
{  
    <instruções quando resultado da condição é verdadeira>  
}
```

Decisão e operadores lógicos

Exemplo:

- Ler pontuação de três jogadores
- Qual dos jogadores obteve a maior pontuação?
 - Desconsiderando empate para simplificar o exemplo

Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;  
read(scoreP1);  
read(scoreP2);  
read(scoreP3);  
  
if ((scoreP1 > scoreP2) && (scoreP1 > scoreP3))  
{  
    write("Jogador 1 obteve a maior pontuação.");  
}  
else if ((scoreP2 > scoreP1) && (scoreP2 > scoreP3))  
{  
    write("Jogador 2 obteve a maior pontuação.");  
}  
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))  
{  
    write("Jogador 3 obteve a maior pontuação.");  
}
```


Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;  
read(scoreP1); // Assumindo scoreP1 = 210.  
read(scoreP2); // Assumindo scoreP2 = 160.  
read(scoreP3); // Assumindo scoreP3 = 390.  
  
if ((scoreP1 > scoreP2) && (scoreP1 > scoreP3))  
{  
    write("Jogador 1 obteve a maior pontuação.");  
}  
else if ((scoreP2 > scoreP1) && (scoreP2 > scoreP3))  
{  
    write("Jogador 2 obteve a maior pontuação.");  
}  
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))  
{  
    write("Jogador 3 obteve a maior pontuação.");  
}
```

Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;  
read(scoreP1); // Assumindo scoreP1 = 210.  
read(scoreP2); // Assumindo scoreP2 = 160.  
read(scoreP3); // Assumindo scoreP3 = 390.  
  
if ((scoreP1 > scoreP2) && (scoreP1 > scoreP3))  
{  
    write("Jogador 1 obteve a maior pontuação.");  
}  
else if ((scoreP2 > scoreP1) && (scoreP2 > scoreP3))  
{  
    write("Jogador 2 obteve a maior pontuação.");  
}  
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))  
{  
    write("Jogador 3 obteve a maior pontuação.");  
}
```

Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;  
read(scoreP1); // Assumindo scoreP1 = 210.  
read(scoreP2); // Assumindo scoreP2 = 160.  
read(scoreP3); // Assumindo scoreP3 = 390.  
  
if ((210 > 160) && (210 > 390))  
{  
    write("Jogador 1 obteve a maior pontuação.");  
}  
else if ((scoreP2 > scoreP1) && (scoreP2 > scoreP3))  
{  
    write("Jogador 2 obteve a maior pontuação.");  
}  
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))  
{  
    write("Jogador 3 obteve a maior pontuação.");  
}
```

Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;  
read(scoreP1); // Assumindo scoreP1 = 210.  
read(scoreP2); // Assumindo scoreP2 = 160.  
read(scoreP3); // Assumindo scoreP3 = 390.
```

```
if ((true) && (false))  
{  
    write("Jogador 1 obteve a maior pontuação.");  
}  
else if ((scoreP2 > scoreP1) && (scoreP2 > scoreP3))  
{  
    write("Jogador 2 obteve a maior pontuação.");  
}  
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))  
{  
    write("Jogador 3 obteve a maior pontuação.");  
}
```

Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;  
read(scoreP1); // Assumindo scoreP1 = 210.  
read(scoreP2); // Assumindo scoreP2 = 160.  
read(scoreP3); // Assumindo scoreP3 = 390.  
  
if (false)  
{  
    write("Jogador 1 obteve a maior pontuação.");  
}  
else if ((scoreP2 > scoreP1) && (scoreP2 > scoreP3))  
{  
    write("Jogador 2 obteve a maior pontuação.");  
}  
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))  
{  
    write("Jogador 3 obteve a maior pontuação.");  
}
```

Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;  
read(scoreP1); // Assumindo scoreP1 = 210.  
read(scoreP2); // Assumindo scoreP2 = 160.  
read(scoreP3); // Assumindo scoreP3 = 390.  
  
if (false)  
{  
    write("Jogador 1 obteve a maior pontuação.");  
}  
else if ((scoreP2 > scoreP1) && (scoreP2 > scoreP3))  
{  
    write("Jogador 2 obteve a maior pontuação.");  
}  
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))  
{  
    write("Jogador 3 obteve a maior pontuação.");  
}
```

Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;
read(scoreP1); // Assumindo scoreP1 = 210.
read(scoreP2); // Assumindo scoreP2 = 160.
read(scoreP3); // Assumindo scoreP3 = 390.

if (false)
{
    write("Jogador 1 obteve a maior pontuação.");
}
else if ((160 > 210) && (160 > 390))
{
    write("Jogador 2 obteve a maior pontuação.");
}
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))
{
    write("Jogador 3 obteve a maior pontuação.");
}
```

Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;  
read(scoreP1); // Assumindo scoreP1 = 210.  
read(scoreP2); // Assumindo scoreP2 = 160.  
read(scoreP3); // Assumindo scoreP3 = 390.  
  
if (false)  
{  
    write("Jogador 1 obteve a maior pontuação.");  
}  
else if ((false) && (false))  
{  
    write("Jogador 2 obteve a maior pontuação.");  
}  
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))  
{  
    write("Jogador 3 obteve a maior pontuação.");  
}
```


Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;  
read(scoreP1); // Assumindo scoreP1 = 210.  
read(scoreP2); // Assumindo scoreP2 = 160.  
read(scoreP3); // Assumindo scoreP3 = 390.  
  
if (false)  
{  
    write("Jogador 1 obteve a maior pontuação.");  
}  
else if (false)  
{  
    write("Jogador 2 obteve a maior pontuação.");  
}  
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))  
{  
    write("Jogador 3 obteve a maior pontuação.");  
}
```


Decisão e operadores lógicos

```
int scoreP1, scoreP2, scoreP3;  
read(scoreP1); // Assumindo scoreP1 = 210.  
read(scoreP2); // Assumindo scoreP2 = 160.  
read(scoreP3); // Assumindo scoreP3 = 390.
```

```
if (false)  
{  
    write("Jogador 1 obteve a maior pontuação.");  
}  
else if (false)  
{  
    write("Jogador 2 obteve a maior pontuação.");  
}  
else // if ((scoreP3 > scoreP1) && (scoreP3 > scoreP2))  
{  
    write("Jogador 3 obteve a maior pontuação.");  
}
```

Decisão e operadores lógicos

Correto!



```
if ((scoreP1 > scoreP2) && (scoreP1 > scoreP3))  
{  
    // ...  
}
```

```
if (scoreP1 > scoreP2 > scoreP3)  
{  
    // ...  
}
```



Errado!

Decisão e operadores lógicos

```
// Assumindo que leftMouseButtonPressed(), spaceBarPressed(),  
// playerIsRunning() e playerCanFly() geram valores true ou false.
```

```
// Se o botão esquerdo do mouse foi pressionado  
// OU a barra de espaço foi pressionada...
```

```
if (leftMouseButtonPressed() || spacebarPressed())  
{  
    // Se o jogador está correndo E o jogador pode voar...  
    if (playerIsRunning() && playerCanFly())  
    {  
        // Voe!  
        fly();  
    }  
}
```

Decisão e condições

Decisão e condições

Expressão que retorna true ou false



```
if (<condição>)  
{  
    <instruções quando resultado da condição é verdadeira>  
}
```

Decisão e condições

Exemplo 1 que pode gerar confusão...

```
boolean booleanValue;  
booleanValue = false;
```

```
if (booleanValue == false)  
{  
    write("Esse trecho de código SEMPRE é executado!");  
}  
else  
{  
    write("Esse trecho de código NUNCA é executado!");  
}
```

Decisão e condições

Exemplo 1 que pode gerar confusão...

```
boolean booleanValue;  
booleanValue = false;
```

**booleanValue == false resulta em true, pois
false == false é true.**




```
if (booleanValue == false)  
{  
    write("Esse trecho de código SEMPRE é executado!");  
}  
else  
{  
    write("Esse trecho de código NUNCA é executado!");  
}
```


Decisão e condições

Exemplo 2 que pode gerar confusão...

```
boolean booleanValue;  
booleanValue = false;
```

```
if (booleanValue)  if (false) { ... }  
{  
    write("Esse trecho de código NUNCA é executado!");  
}  
else  
{  
    write("Esse trecho de código SEMPRE é executado!");  
}
```



Switch-Case

Switch-case

Nestas decisões encadeadas, cada decisão compara se um identificador é igual a um valor específico.

```
if (<identificador> == <valor1>)
{
    <instruções quando identificador == valor1>
}
else if (<identificador> == <valor2>)
{
    <instruções quando identificador == valor2>
}
else if (<identificador> == <valorN>)
{
    <instruções quando identificador == valorN>
}
else
{
    <instruções quando todas decisões anteriores são falsas>
}
```

Switch-case

Quando isso ocorre, podemos usar o comando switch-case.

```
switch (<identificador>)  
{  
  case <valor1>:   
    <instruções quando identificador == valor1>  
    break;  
  
  case <valor2>:   
    <instruções quando identificador == valor2>  
    break;  
  
  case <valorN>:   
    <instruções quando identificador == valorN>  
    break;  
  
  default: // opcional   
    <instruções quando identificador é diferente de todos os outros casos>  
    break;  
}
```

Switch-case

Assumindo `identificador == valor2...`

```
switch (<identificador>)  
{  
  case <valor1>:  
    <instruções quando identificador == valor1>  
    break;  
  
  case <valor2>:  
    <instruções quando identificador == valor2>  
    break;  
  
  case <valorN>:  
    <instruções quando identificador == valorN>  
    break;  
  
  default:  
    <instruções quando identificador é diferente de todos os outros casos>  
    break;  
}
```

Switch-case

Assumindo `identificador == valor2...`

```
switch (<identificador>)
```

```
{
```

```
  case <valor1>:
```

```
    <instruções quando identificador == valor1>
```

```
    break;
```

Esse trecho de código é executado...

```
  case <valor2>:
```

```
    <instruções quando identificador == valor2>
```

```
    break;
```

...até encontrar a instrução `break`.

```
  case <valorN>:
```

```
    <instruções quando identificador == valorN>
```

```
    break;
```

```
  default:
```

```
    <instruções quando identificador é diferente de todos os outros casos>
```

```
    break;
```

```
}
```

Todos os outros casos e o caso `default` são ignorados.

Switch-case

E se esquecermos da instrução **break**?
Assumindo `identificador == valor2...`

```
switch (<identificador>)
```

```
{
```

```
  case <valor1>:
```

```
    <instruções quando identificador == valor1>
```

```
    break;
```

Esse trecho de código é executado...

```
  case <valor2>:
```

```
    <instruções quando identificador == valor2>
```

break?

```
  case <valorN>:
```

```
    <instruções quando identificador == valorN>
```

```
    break;
```

```
  default:
```

```
    <instruções quando identificador é diferente de todos os outros casos>
```

```
    break;
```

```
}
```

Switch-case

E se esquecermos da instrução **break**?
Assumindo `identificador == valor2...`

```
switch (<identificador>)
```

```
{
```

```
  case <valor1>:
```

```
    <instruções quando identificador == valor1>
```

```
    break;
```

```
  case <valor2>:
```

```
    <instruções quando identificador == valor2>
```

```
  case <valorN>:
```

```
    <instruções quando identificador == valorN>
```

```
    break;
```

```
  default:
```

```
    <instruções quando identificador é diferente de todos os outros casos>
```

```
    break;
```

```
}
```

...e esse também!

Continua sequencialmente até encontrar a
instrução **break** (ou acabar o bloco do **switch**).

Switch-case

Às vezes, queremos omitir o **break**...
“Casos em sequência” funcionam como
decisões que usam **|| (OR)**.

```
switch (keyPressed)
{
  case KEY_W:
  case KEY_I:
  case KEY_UP_ARROW:
    playerGoUp();
    break;

  case KEY_S:
  case KEY_K:
  case KEY_DOWN_ARROW:
    playerGoDown();
    break;

  // ...
}
```

playerGoUp() é chamado para esses casos.

```
if (keyPressed == KEY_W
    || keyPressed == KEY_I
    || keyPressed == KEY_UP_ARROW)
{
  playerGoUp();
}

else if (keyPressed == KEY_S
        || keyPressed == KEY_K
        || keyPressed == KEY_DOWN_ARROW)
{
  playerGoDown();
}

// ...
```



Linguagem Java

Java

A sintaxe das estruturas de decisão em Java é igual à apresentada nos slides anteriores.



Exemplo Java

Exemplo 4: Função polinomial do 2º grau

Uma função polinomial do 2º grau é qualquer função $f: \mathcal{R} \rightarrow \mathcal{R}$ dada por $y = f(x) = ax^2 + bx + c$, sendo $a, b, c \in \mathcal{R}$ e $a \neq 0$.

As raízes da função polinomial do 2º grau são dadas pela fórmula de Bhaskara: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Observação:

O total de raízes reais depende do valor obtido para o discriminante $\Delta = b^2 - 4ac$, a saber:

- $\Delta > 0 \rightarrow$ há duas raízes reais e distintas;
- $\Delta = 0 \rightarrow$ há só uma raiz real;
- $\Delta < 0 \rightarrow$ não há raiz real.

O seu gráfico é uma curva chamada parábola, sendo que, se $a > 0$, a parábola tem concavidade para cima e um ponto de mínimo $P = (-b/2a, -\Delta/4a)$ e, se $a < 0$, a parábola tem concavidade para baixo e um ponto de máximo $P = (-b/2a, -\Delta/4a)$. Tendo por base isso, elabore um programa em Java que leia a, b, c , calcule o Δ , as raízes da equação e o ponto de mínimo ou máximo P .

Exemplo 4: Função polinomial do 2º grau

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        // Entrada dos coeficientes a, b, c para uma equação do 2o grau.
        System.out.println("Equação do 2o Grau: ax^2 + bx + c = 0");
        System.out.print("Digite o valor de a: ");
        double a = s.nextDouble();
        if (a == 0)
            System.out.println("A equação não é de 2o grau.");
        else {
            System.out.print("Digite o valor de b: ");
            double b = s.nextDouble();
            System.out.print("Digite o valor de c: ");
            double c = s.nextDouble();

            // Continua no próximo slide...
```

Exemplo 4: Função polinomial do 2º grau

```
// ...continuação do slide anterior.

// Calcula o delta e imprime.
double delta = Math.pow(b, 2) - 4 * a * c;
System.out.println("\nO valor do delta é: " + delta);

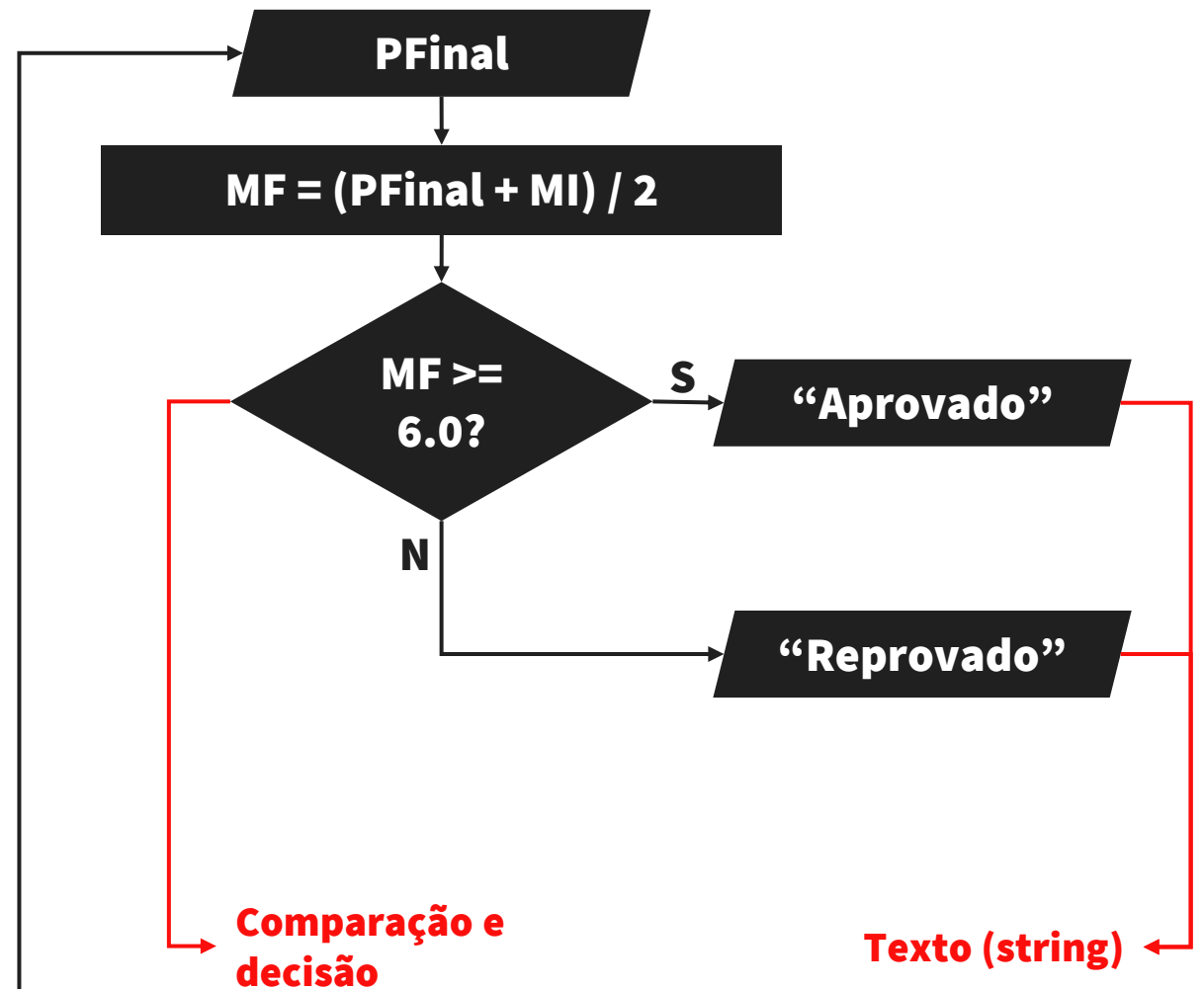
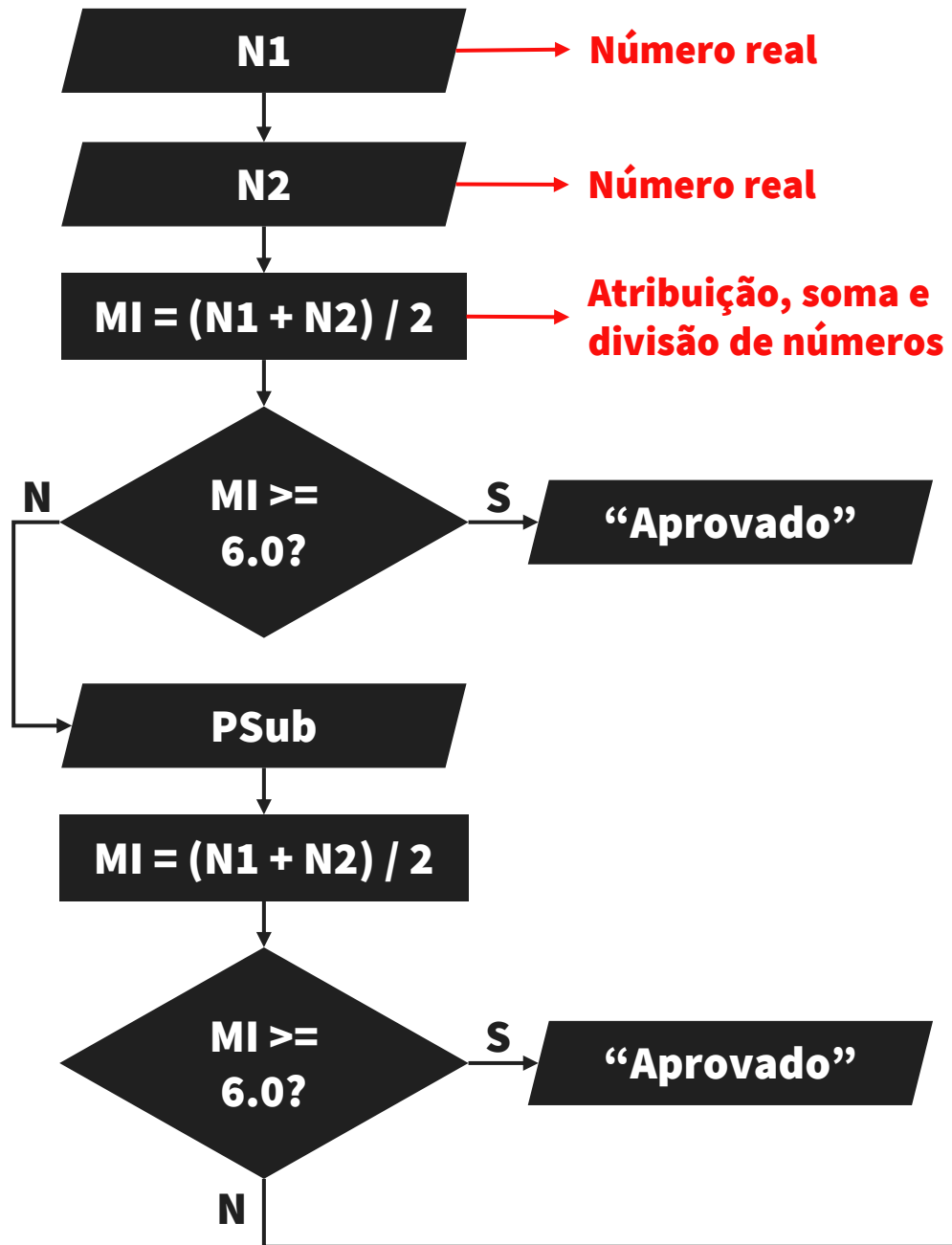
// Apresenta as raízes da equação.
if (delta > 0) {
    System.out.println("Duas raízes reais!");
    double x1 = (-b + Math.sqrt(delta)) / (2 * a);
    double x2 = (-b - Math.sqrt(delta)) / (2 * a);
    System.out.println("x1 = " + x1 + ", x2 = " + x2);
} else if (delta == 0) {
    System.out.println("Uma raiz real!");
    double x = -b / 2 * a;
    System.out.println("x = " + x);
} else
    System.out.println("Não há raízes reais!");

// Continua no próximo slide...
```

Exemplo 4: Função polinomial do 2º grau

```
// ...continuação do slide anterior.  
  
// Apresenta os pontos de máximo/mínimo.  
if (a > 0)  
    System.out.println("Ponto de mínimo:");  
else  
    System.out.println("Ponto de máximo:");  
double px = -b / 2 * a;  
double py = -delta / 4 * a;  
System.out.println("x = " + px + ", y = " + py + ".\n");  
}  
}  
}
```


Estruturas de Repetição



Estruturas de repetição

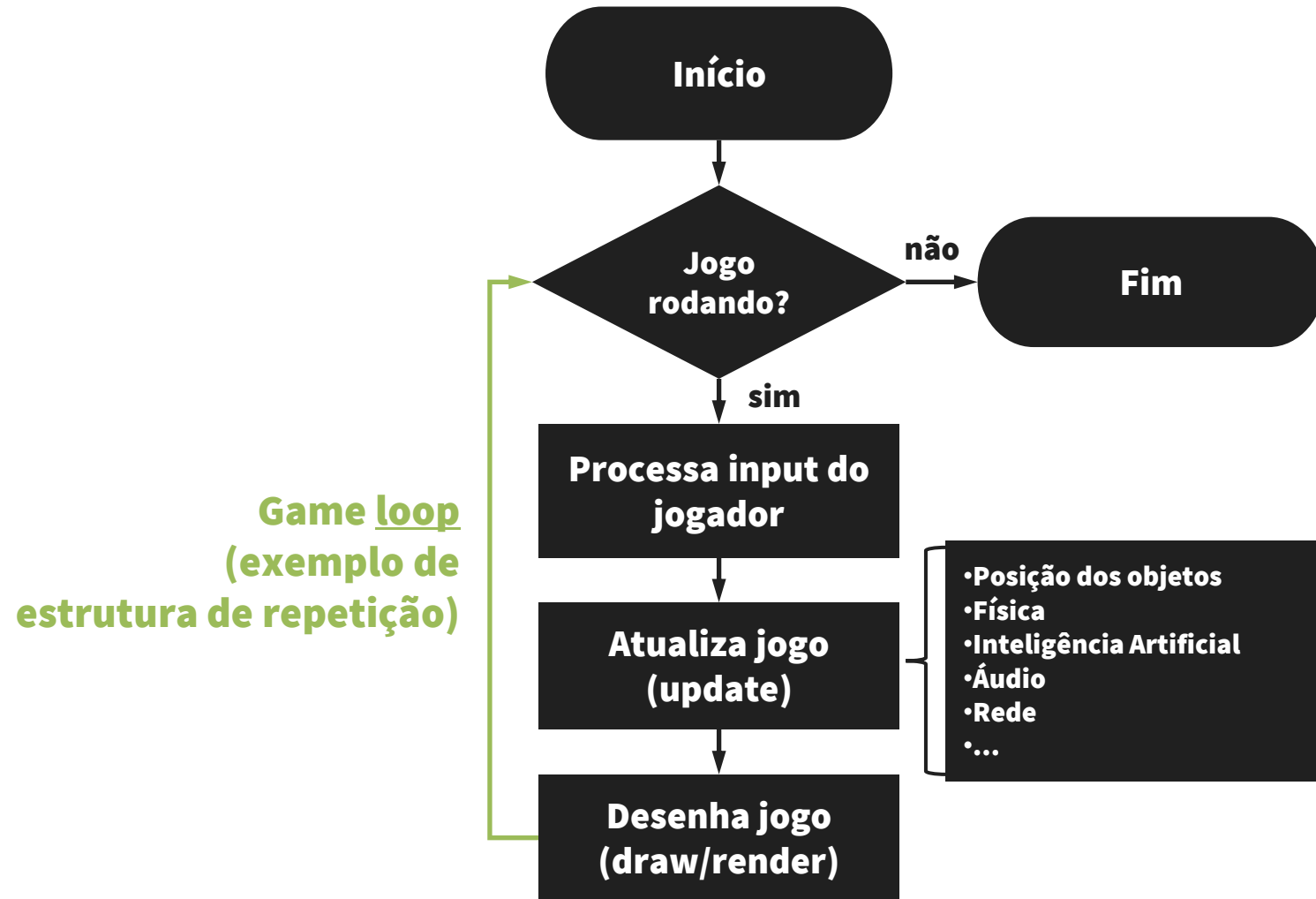
- Quero executar o algoritmo do slide anterior para 100 alunos.
- Como resolver?
 - Reescrevo o algoritmo 100 vezes.
 - Ou CTRL+C, CTRL+V (cuidado!)

Estruturas de repetição

- Quero executar o algoritmo do slide anterior para 100 alunos.
- Como resolver?
 - ~~Reescrevo o algoritmo 100 vezes.~~
 - ~~Ou CTRL+C, CTRL+V (cuidado!)~~
 - Uso estruturas de repetição!

Estruturas de repetição

- Repetir um bloco de instruções
 - N vezes ou
 - Enquanto uma condição for verdadeira
- Também conhecidas como
 - Laços
 - Loops



Estruturas de repetição

- while
- do-while
- repeat-until
- for
- Interrompendo e pulando um loop
- Repetições encadeadas

4a

**While
 (“Enquanto”)**

while

enquanto <condição> **faça**
 <instruções quando resultado da condição é verdadeira>
fim_enquanto

while (<condição>)
{
 <instruções quando resultado da condição é verdadeira>
}

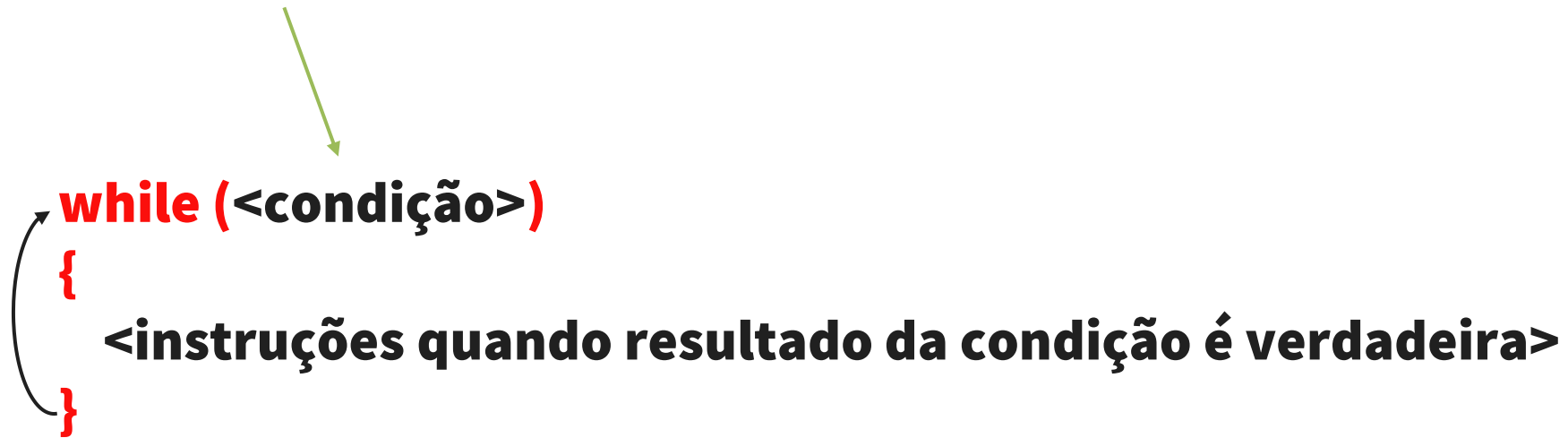
while

enquanto <condição> faça
 <instruções quando resultado da condição é verdadeira>
fim_enquanto

while (<condição>)
{
 <instruções quando resultado da condição é verdadeira>
}

while

Expressão que retorna true ou false, define se bloco de instruções da repetição deve ser executado.
Condição é verificada **ANTES** de executar o bloco de instruções.



```
while (<condição>)  
{  
    <instruções quando resultado da condição é verdadeira>  
}
```

while

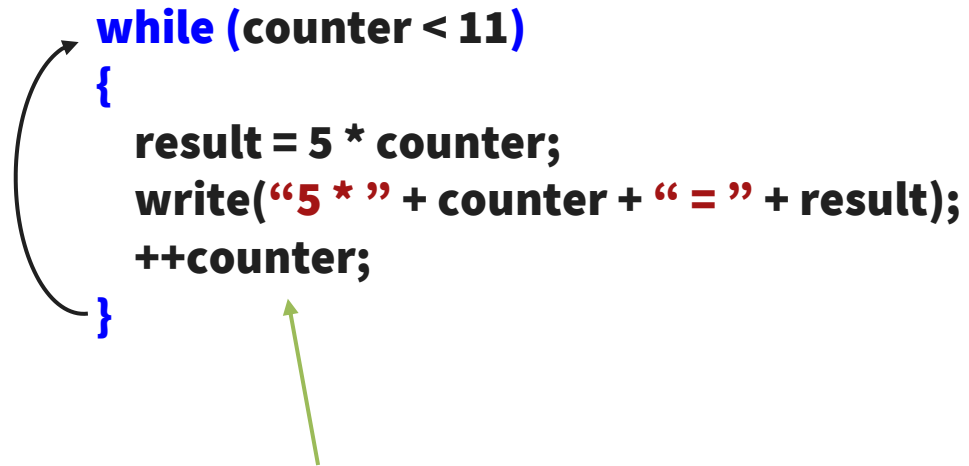
```
int counter, result;  
counter = 0;
```

```
while (counter < 11)  
{  
    result = 5 * counter;  
    write("5 * " + counter + " = " + result);  
    ++counter;  
}
```

while

```
int counter, result;  
counter = 0;
```

```
while (counter < 11)  
{  
    result = 5 * counter;  
    write("5 * " + counter + " = " + result);  
    ++counter;  
}
```



Repetição while com contador (não há necessidade do usuário entrar com algum valor para que o loop seja finalizado).

counter	result	saída
0	0	5 * 0 = 0
1	5	5 * 1 = 5
2	10	5 * 2 = 10
3	15	5 * 3 = 15
4	20	5 * 4 = 20
5	25	5 * 5 = 25
6	30	5 * 6 = 30
7	35	5 * 7 = 35
8	40	5 * 8 = 40
9	45	5 * 9 = 45
10	50	5 * 10 = 50

while

```
float n1, n2, finalScore;
```

```
char answer;
```

```
answer = 'S';
```

```
while (answer == 'S')
```

```
{
```

```
    write("Digite a primeira nota: ");
```

```
    read(n1);
```

```
    write("Digite a segunda nota: ");
```

```
    read(n2);
```

```
    finalScore = (n1 + n2) / 2;
```

```
    write("A média final é:" + finalScore);
```

```
    write("Calcular outra média? (S/N)");
```

```
    read(answer);
```

```
}
```

while

```
float n1, n2, finalScore;  
char answer;  
answer = 'S';
```

Define o valor de answer como 'S' antes do código do while, pois a repetição while verifica a condição de repetição antes do bloco de instruções.

```
while (answer == 'S')  
{  
    write("Digite a primeira nota: ");  
    read(n1);  
    write("Digite a segunda nota: ");  
    read(n2);  
    finalScore = (n1 + n2) / 2;  
    write("A média final é:" + finalScore);  
    write("Calcular outra média? (S/N)");  
    read(answer);  
}
```

O fim do loop depende do valor inserido pelo usuário, diferente do loop com contador.

4b

Do-While

(“Faça-Enquanto”)

do-while

faça

**<instruções quando resultado da condição é verdadeira,
a partir da segunda vez (o bloco de instruções sempre é
executado pelo menos uma vez)>**

enquanto <condição>

**do
{**

**<instruções quando resultado da condição é verdadeira,
a partir da segunda vez (o bloco de instruções sempre é
executado pelo menos uma vez)>**

} while (<condição>)

do-while

faça

<instruções quando resultado da condição é verdadeira,
a partir da segunda vez (o bloco de instruções sempre é
executado pelo menos uma vez)>

enquanto <condição>

do

{

**<instruções quando resultado da condição é verdadeira,
a partir da segunda vez (o bloco de instruções sempre é
executado pelo menos uma vez)>**

} while (<condição>)

do-while

do
{

**<instruções quando resultado da condição é verdadeira,
a partir da segunda vez (o bloco de instruções sempre é
executado pelo menos uma vez)>**

} while (<condição>)




**Expressão que retorna true ou false, define se bloco de
instruções da repetição deve ser executado.
Condição é verificada DEPOIS de executar o bloco de
instruções pelo menos uma vez.**

do-while

```
int counter, result;  
counter = 0;
```

```
do  
{  
    result = 5 * counter;  
    write("5 * " + counter + " = " + result);  
    ++counter;  
} while (counter < 11)
```




do-while

```
float n1, n2, finalScore;  
char answer;
```

```
do  
{  
    write("Digite a primeira nota: ");  
    read(n1);  
    write("Digite a segunda nota: ");  
    read(n2);  
    finalScore = (n1 + n2) / 2;  
    write("A média final é:" + finalScore);  
    write("Calcular outra média? (S/N)");  
    read(answer);  
} while (answer == 'S')
```

Repare que não definimos um valor inicial de answer para que a repetição do-while seja executada na primeira vez (compare com a repetição while).



4c

Repeat-Until (“Repita-Até”)

repeat-until

repita

<instruções quando resultado da condição é falsa, a partir da segunda vez (o bloco de instruções sempre é executado pelo menos uma vez)>

até <condição>

repeat

{

<instruções quando resultado da condição é falsa, a partir da segunda vez (o bloco de instruções sempre é executado pelo menos uma vez)>

} until (<condição>)

repeat-until

repita

<instruções quando resultado da condição é falsa, a partir da segunda vez (o bloco de instruções sempre é executado pelo menos uma vez)>
até <condição>

repeat

{

<instruções quando resultado da condição é falsa, a partir da segunda vez (o bloco de instruções sempre é executado pelo menos uma vez)>

} until (<condição>)

repeat-until

repeat

{

<instruções quando resultado da condição é falsa, a partir da segunda vez (o bloco de instruções sempre é executado pelo menos uma vez)>

} until (<condição>)




Expressão que retorna true ou false, define se bloco de instruções da repetição deve ser executado.

Condição é verificada DEPOIS de executar o bloco de instruções pelo menos uma vez.

repeat-until

```
int counter, result;  
counter = 0;
```

```
repeat  
{  
    result = 5 * counter;  
    write("5 * " + counter + " = " + result);  
    ++counter;  
} until (counter > 10)
```




repeat-until

```
float n1, n2, finalScore;  
char answer;
```

```
repeat  
{  
    write(“Digite a primeira nota: ”);  
    read(n1);  
    write(“Digite a segunda nota: ”);  
    read(n2);  
    finalScore = (n1 + n2) / 2;  
    write(“A média final é:” + finalScore);  
    write(“Calcular outra média? (S/N)”);  
    read(answer);  
} until (answer != ‘S’)
```

Repare que não definimos um valor inicial de answer para que a repetição repeat-until seja executada na primeira vez (compare com a repetição while).



do-while x repeat-until

- Geralmente as linguagens de programação possuem somente uma das duas estruturas de repetição com condição após o bloco de instruções.
 - while, do-while, for
 - while, repeat-until, for
- Importante reparar que a condição do do-while é diferente da condição do repeat-until.

do-while x repeat-until

```
int counter, result;  
counter = 0;
```

```
do  
{  
    result = 5 * counter;  
    write("..." + result);  
    ++counter;  
} while (counter < 11)
```



Condição: counter < 11

**“Faça (execute) esse
bloco de instruções
enquanto counter for
menor que 11”**

```
int counter, result;  
counter = 0;
```

```
repeat  
{  
    result = 5 * counter;  
    write("..." + result);  
    ++counter;  
} until (counter > 10)
```



Condição: counter > 10

**“Repita (execute) esse
bloco de instruções até
que counter seja maior
que 10”**

do-while x repeat-until

```
do
{
    write("Calcular outra média?");
    read(answer);
} while (answer == 'S')
```

↑
Condição: `answer == 'S'`

“Faça (execute) esse
bloco de instruções
enquanto `answer` for
igual à `'S'`”

```
repeat
{
    write("Calcular outra média?");
    read(answer);
} until (answer != 'S')
```

↑
Condição: `answer != 'S'`

“Repita (execute) esse
bloco de instruções até
que `answer` seja
diferente de `'S'`”



**For
("Para")**

for

para <variável> **de** <valor inicial> **até** <valor limite> **passo** <incremento>
faça
 <instruções enquanto variável está dentro do valor limite>
fim_para

for (<valor inicial>; <condição>; <passo>)
{
 <instruções quando resultado da condição é verdadeira>
}

for

para <variável> de <valor inicial> até <valor limite> passo <incremento>
faça
 <instruções enquanto variável está dentro do valor limite>
fim_para

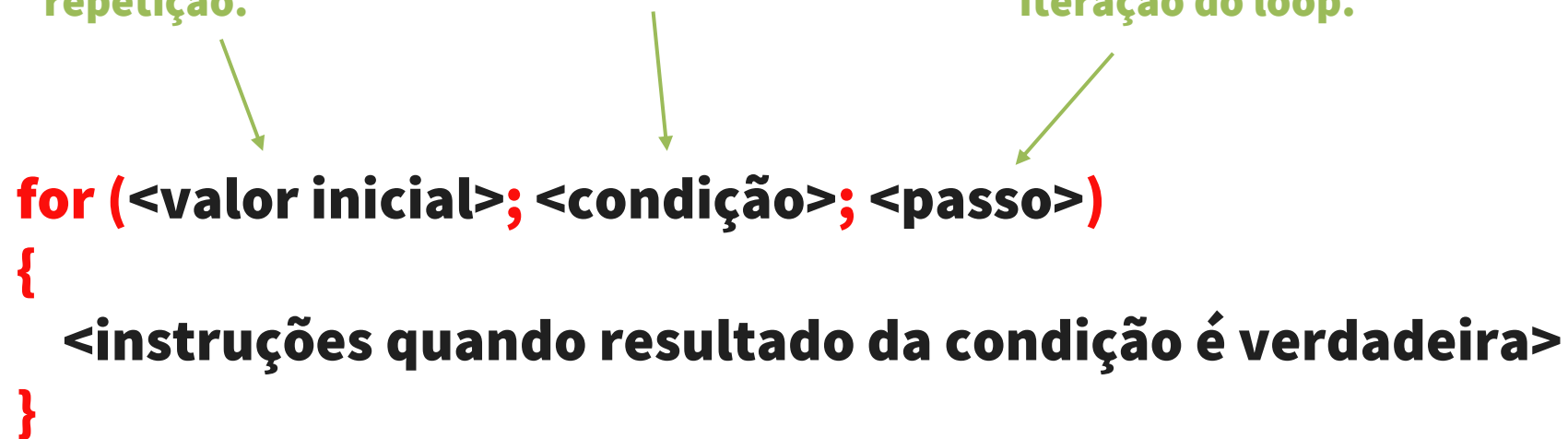
for (<valor inicial>; <condição>; <passo>)
{
 <instruções quando resultado da condição é verdadeira>
}

for

Valor inicial da
variável usada para
controlar a
repetição.

Condição para que
loop seja executado
ou encerrado.

Como a variável de
controle é
atualizada a cada
iteração do loop.



```
for (<valor inicial>; <condição>; <passo>)  
{  
    <instruções quando resultado da condição é verdadeira>  
}
```

for

Ordem de execução do loop for:
Passo 1

**<valor inicial> é
executado somente
na primeira iteração
do loop.**

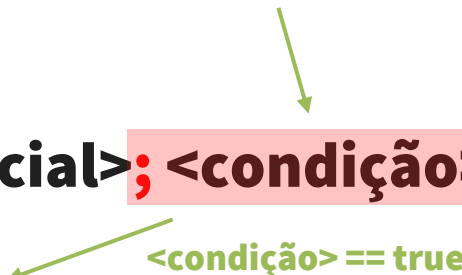


```
for (<valor inicial>; <condição>; <passo>)  
{  
    <instruções quando resultado da condição é verdadeira>  
}
```

for

Ordem de execução do loop for:
Passo 2

Verifica <condição>.
Caso seja true,
executa o bloco de
instruções.

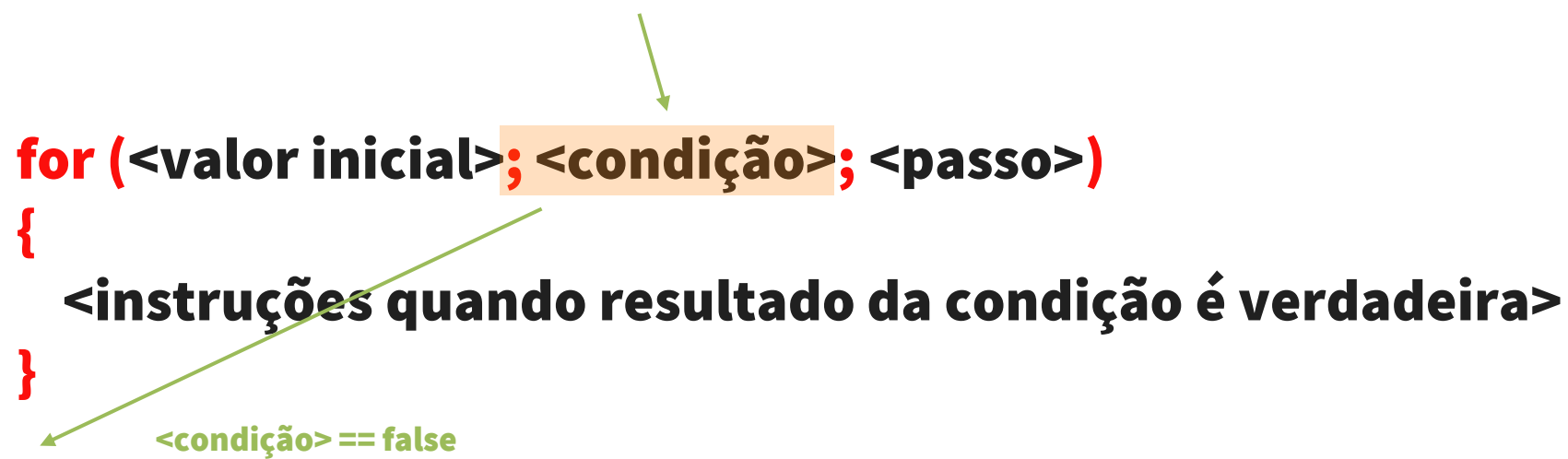


```
for (<valor inicial>; <condição>; <passo>)  
{  
    <instruções quando resultado da condição é verdadeira>  
}
```

for

Ordem de execução do loop for:
Passo 2

Verifica <condição>.
Caso seja false, sai
do loop.



```
for (<valor inicial>; <condição>; <passo>)  
{  
    <instruções quando resultado da condição é verdadeira>  
}
```

<condição> == false

for

Ordem de execução do loop for:
Passo 3

Após executar bloco
de instruções,
executa <passo> e
volta para passo 2.



```
for (<valor inicial>; <condição>; <passo>)  
{  
    <instruções quando resultado da condição é verdadeira>  
}
```

for

```
int counter, result;
```

```
for (counter = 0; counter < 11; ++counter)  
{  
    result = 5 * counter;  
    write("5 * " + counter + " = " + result);  
}
```

for

```
int counter, result;
```

```
for (counter = 0; counter < 11; ++counter)
```

```
{
```

```
    result = 5 * counter;
```

```
    write("5 * " + counter + " = " + result);
```

```
}
```



4e

Interrompendo e pulando um loop

Interrompendo um loop

- Caso necessário, é possível interromper um loop antes do seu encerramento normal.
- Via instrução `break`;
 - A mesma do comando `switch-case`.

Interrompendo um loop

- Exemplo:
 - Ler números inteiros até que 0 seja informado;
 - Quando 0 for inserido, encerrar o loop;
 - Após loop terminar, exibir a média dos números.

Interrompendo um loop

```
int total, number, counter;  
total = 0;  
counter = 0;
```

```
while (true) // loop infinito.
```

```
{
```

```
    read(number);
```

```
    if (number == 0)
```

```
    {
```

```
        break; // interrompe o loop!
```

```
    }
```

```
    total += number;
```

```
    ++counter;
```

```
}
```

```
write("Média dos números inseridos: " + (total / counter));
```

Instruções ignoradas
quando break é chamado.



“Pulando” um loop

- Também é possível ignorar um trecho de um loop, sem interrompê-lo.
- Via instrução `continue`;
- Todas as instruções após `continue`; são ignoradas e loop volta para o início, começando uma nova iteração.

“Pulando” um loop

- Exemplo:
 - Ler números inteiros até que 0 seja informado;
 - Quando 0 for inserido, encerrar o loop;
 - Ignorar números negativos;
 - Após loop terminar, exibir a média dos números.

“Pulando” um loop

```
int total, number, counter;  
total = 0;  
counter = 0;  
while (true) // loop infinito.
```

```
{  
    read(number);  
    if (number == 0)  
    {  
        break;  
    }  
    else if (number < 0)  
    {  
        continue; // “pula” o loop  
    }  
    total += number;  
    ++counter;  
}
```

Instruções ignoradas caso
number < 0.

```
write(“Média dos números inseridos: ” + (total / counter));
```

4f

Repetições encadeadas

Repetições encadeadas

- Assim como estruturas de decisão, é possível encadear repetições.
- Podem ser escritas com quaisquer estruturas de repetição:
 - for dentro de while
 - do-while dentro de while dentro de for
 - Etc.
- Depende do problema a ser resolvido e de como definimos o algoritmo.

Repetições encadeadas

// Tabuada de multiplicação de 1 a 9, nos intervalos de 0 a 10.

```
int number, multiplier, result;
```

```
for (number = 1; number < 10; ++number)
```

```
{
```

```
    for (multiplier = 0; multiplier < 11; ++multiplier)
```

```
    {
```

```
        result = number * multiplier;
```

```
        write(number + " * " + multiplier + " = " + result);
```

```
    }
```

```
}
```

Repetições encadeadas

// Tabuada de multiplicação de 1 a 9, nos intervalos de 0 a 10.

```
for (number = 1; number < 10; ++number)
```

```
for (multiplier = 0; multiplier < 11; ++multiplier)
```

number	multiplier	result	saída
1			

for externo

Repetições encadeadas

// Tabuada de multiplicação de 1 a 9, nos intervalos de 0 a 10.

for (number = 1; number < 10; ++number)

for (multiplier = 0; multiplier < 11; ++multiplier)

for interno

for externo

number	multiplier	result	saída
1	0	0	1 * 0 = 0
	1	1	1 * 1 = 1
	2	2	1 * 2 = 2
	3	3	1 * 3 = 3
	4	4	1 * 4 = 4
	5	5	1 * 5 = 5
	6	6	1 * 6 = 6
	7	7	1 * 7 = 7
	8	8	1 * 8 = 8
	9	9	1 * 9 = 9
	10	10	1 * 10 = 10

Repetições encadeadas

// Tabuada de multiplicação de 1 a 9, nos intervalos de 0 a 10.

```
for (number = 1; number < 10; ++number)
```

```
for (multiplier = 0; multiplier < 11; ++multiplier)
```

number	multiplier	result	saída
2			

for externo

Repetições encadeadas

// Tabuada de multiplicação de 1 a 9, nos intervalos de 0 a 10.

for (number = 1; number < 10; ++number)

for (multiplier = 0; multiplier < 11; ++multiplier)

for interno

for externo

number	multiplier	result	saída
2	0	0	2 * 0 = 0
	1	2	2 * 1 = 2
	2	4	2 * 2 = 4
	3	6	2 * 3 = 6
	4	8	2 * 4 = 8
	5	10	2 * 5 = 10
	6	12	2 * 6 = 12
	7	14	2 * 7 = 14
	8	16	2 * 8 = 16
	9	18	2 * 9 = 18
	10	20	2 * 10 = 20



Linguagem Java

Java

- Possui três estruturas de repetição
 - while
 - do-while
 - for

Java

- Há também uma estrutura de repetição conhecida como for-each (*range-based loop*), usada para percorrer coleções de dados (ex. arrays).
- Algumas regras para o for-each:
 - Somente leitura (um elemento da coleção não pode ser modificado).
 - Uma única coleção de dados (não é possível percorrer duas coleções de dados de uma vez – por ex., comparar dois arrays).
 - Um único elemento por vez (acesso apenas ao elemento atual do loop).
 - Apenas para frente (a coleção de dados é percorrida do primeiro ao último elemento, um por vez).
- Um exemplo usando for-each pode ser visto na seção 5 (Vetores e Matrizes).

Java – while

Expressão que retorna true ou false, define se bloco de instruções da repetição deve ser executado. Condição é verificada **ANTES** de executar o bloco de instruções.



```
while (<condição>) {  
    <instruções quando resultado da condição é verdadeira>  
}
```

Java – do-while

do {

**<instruções quando resultado da condição é verdadeira,
a partir da segunda vez (o bloco de instruções sempre é
executado pelo menos uma vez)>**

} while (<condição>);



**Expressão que retorna true ou false, define se bloco de
instruções da repetição deve ser executado.**

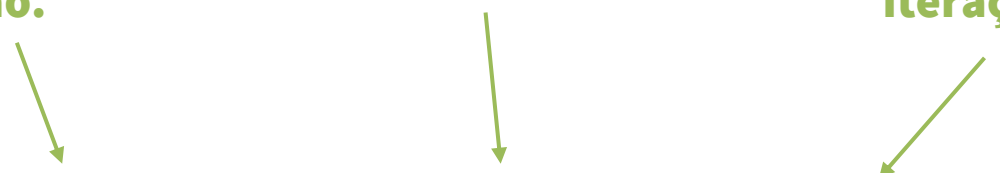
**Condição é verificada DEPOIS de executar o bloco de
instruções pelo menos uma vez.**

Java – for

Valor inicial da
variável usada para
controlar a
repetição.

Condição para que
loop seja executado
ou encerrado.

Como a variável de
controle é
atualizada a cada
iteração do loop.



```
for (<valor inicial>; <condição>; <passo>) {  
    <instruções quando resultado da condição é verdadeira>  
}
```

Java – for-each

Tipo de dado da coleção de dados a ser percorrida.

Variável local do loop for (elemento atual da coleção de dados).

Variável que contém a coleção de dados.



```
for (<tipo> <variável> : <array/coleção de dados>) {  
    <instruções a serem executadas para o elemento atual da  
    coleção de dados>  
}
```



Exemplo Java

Exemplo 5: Repetição while

```
import java.util.Scanner;
class Main {
    static final int QTDE_NUMEROS = 5;
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        float soma = 0.0f, valor = 0.0f;
        int i = 0;
        while (i < QTDE_NUMEROS) {
            System.out.print("Digite o " + (i + 1) + "o valor: ");
            valor = s.nextFloat();
            soma = soma + valor;
            i++;
        }
        System.out.println("\nSoma = " + soma);
    }
}
```

Exemplo 6: Repetição do-while

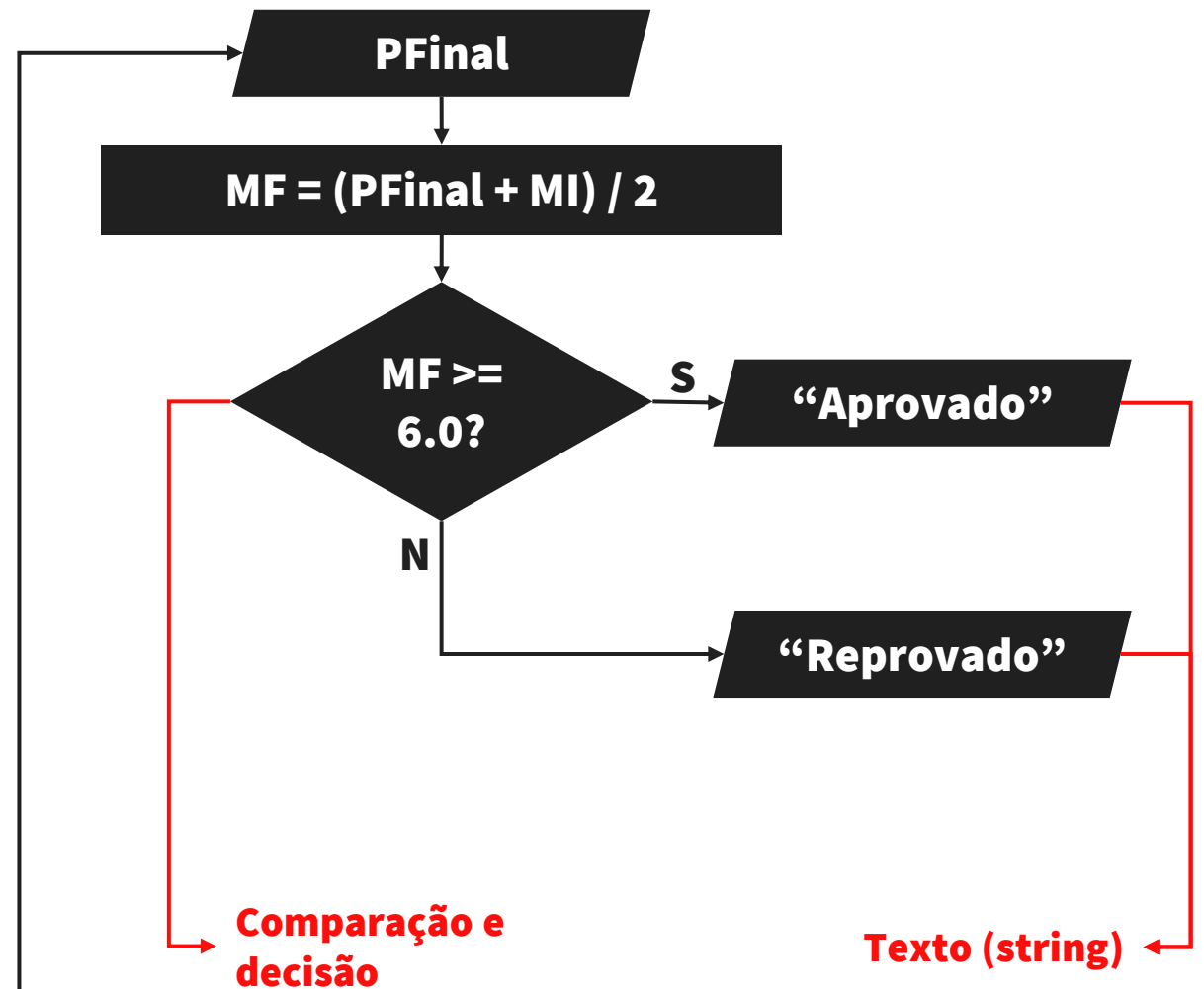
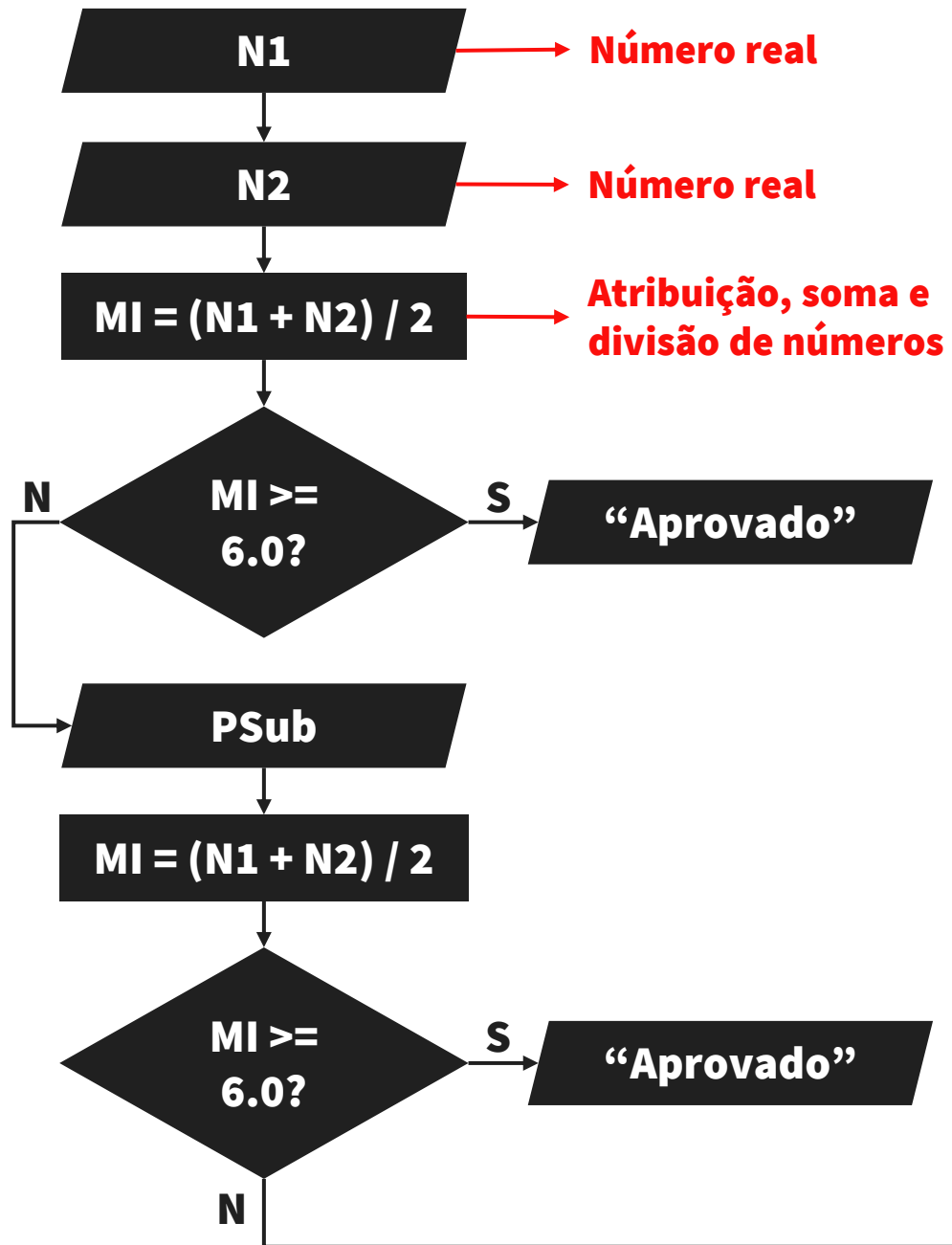
```
import java.util.Scanner;
class Main {
    static final int QTDE_NUMEROS = 5;
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        float soma = 0.0f, valor = 0.0f;
        int i = 0;
        do {
            System.out.print ("Digite o " + (i + 1) + "o valor: ");
            valor = s.nextFloat();
            soma = soma + valor;
            i++;
        } while(i < QTDE_NUMEROS);
        System.out.println("\nSoma = " + soma);
    }
}
```


Exemplo 7: Repetição for

```
import java.util.Scanner;
class Main {
    static final int QTDE_NUMEROS = 5;
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        float soma = 0.0f, valor = 0.0f;
        int i = 0;
        for (i = 0; i < QTDE_NUMEROS; i++) {
            System.out.print("Digite o " + (i + 1) + "o valor: ");
            valor = s.nextFloat();
            soma = soma + valor;
        }
        System.out.println("\nSoma = " + soma);
    }
}
```

5

Vetores e Matrizes



Vetores e matrizes

- Na seção anterior (4 – Estruturas de Repetição)...
 - Quero executar o algoritmo do slide anterior para 100 alunos.
 - Como resolver?
 - ~~Reescrevo o algoritmo 100 vezes.~~
 - ~~Ou CTRL+C, CTRL+V (cuidado!)~~
 - Uso estruturas de repetição!

Vetores e matrizes

- Agora...
 - Quero executar o algoritmo do slide anterior para 100 alunos && manter as notas N1, N2 e média de cada aluno na memória.
 - Na seção anterior, as notas eram “perdidas” a cada iteração do loop.
 - Como resolver o novo problema?

Vetores e matrizes

- Até o momento: variáveis unitárias.
 - Armazenam apenas um valor de cada vez.
- Solução: criar 100 variáveis para N1, 100 variáveis para N2 e 100 variáveis para média
 - n1_aluno1, n1_aluno2, ..., n1_aluno100.
- E se, ao invés de 100 alunos, quisermos mil alunos? 3 mil variáveis?

Vetores e matrizes

- Solução: variáveis compostas.
 - Representam um conjunto de dados de mesmo tipo.
 - Armazenam diversos valores na memória, de forma sequencial.
 - Também conhecidas como vetores e matrizes, ou ainda, arrays (arranjos).

5a

Vetores

Variáveis (1a)

- Usadas para armazenar e acessar dados
- Região (divisão) da memória do computador
 - Possui endereço único
 - Armazena um dado por vez
 - Enquanto programa está sendo executado

...	101	102	103	104	105	106	107	108	109
110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	...

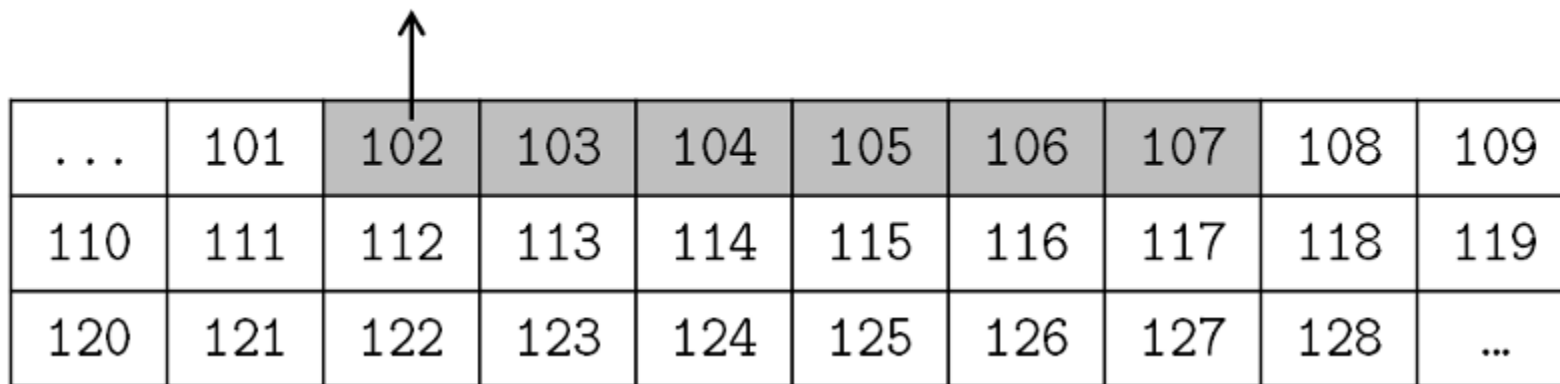
Vetores

- Nome
 - Segue regras de nomenclatura de variáveis
- Tipo
 - Qual o tipo de dado
- Tamanho
 - Quantos elementos devem ser armazenados na memória

Vetores

- Valores são armazenados sequencialmente na memória.
- Exemplo: vetor com 6 elementos (vetor de tamanho 6).

Início do vetor (primeiro elemento do vetor)



The diagram illustrates a memory layout with a grid of 10 columns and 3 rows. The first row contains addresses from 101 to 109, with an ellipsis at the beginning. The second row contains addresses from 110 to 119. The third row contains addresses from 120 to 128, with an ellipsis at the end. An arrow points from the text 'Início do vetor (primeiro elemento do vetor)' to the cell containing the address 102, which is highlighted in gray. The cells containing 102, 103, 104, 105, 106, and 107 in the first row are also highlighted in gray, representing a vector of 6 elements.

...	101	102	103	104	105	106	107	108	109
110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	...

Vetores

- Geralmente, a declaração de variável é feita usando [] após o nome da variável
 - [] e <tamanho> dependem da linguagem.

<tipo> <nomeDaVariavel>[<tamanho>];

variável chamada meuVetor

int meuVetor[6];



Armazena 6 elementos do tipo int (ou... vetor tem tamanho 6)

Vetores

- Cada elemento do vetor é identificado por um índice.
 - Número que indica a posição do elemento no vetor.
 - Geralmente começa em zero.
 - Usado para acessar conteúdo armazenado na memória por cada elemento (como uma variável unitária).

Vetores

- Acesso a um elemento do vetor também é feito usando [] após o nome da variável.

<nomeDaVariavel>[<indiceElemento>] = <valor>;
write(“Conteúdo:” + <nomeDaVariavel>[<indiceElemento>]);

Associa o valor 99 ao primeiro elemento do vetor chamado meuVetor

meuVetor[0] = 99;
write(“Conteúdo:” + meuVetor[0]);



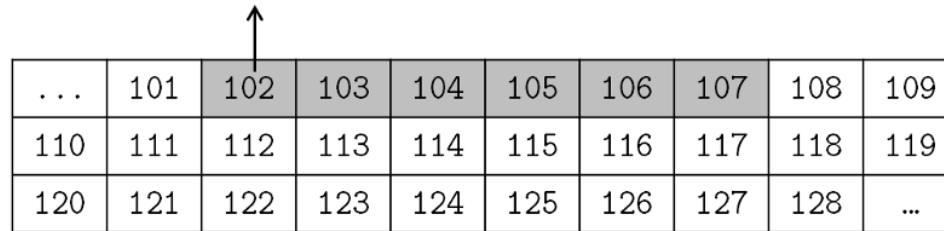
Acessa o valor armazenado no primeiro elemento do vetor chamado meuVetor

Vetores

- Dado um índice n de um vetor, o computador busca o conteúdo da memória no endereço $E[n] = E[0] + n$
 - n é o índice do elemento a ser acessado
 - $E[n]$ é o endereço do elemento de índice n
 - $E[0]$ é o endereço do primeiro elemento

Vetores

Início do vetor (primeiro elemento do vetor)



...	101	102	103	104	105	106	107	108	109
110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	...

Elemento	Índice	Endereço de memória
1º	0	$E[0] = E[0] + 0 = 102 + 0 = 102$
2º	1	$E[1] = E[0] + 1 = 102 + 1 = 103$
3º	2	$E[2] = E[0] + 2 = 102 + 2 = 104$
4º	3	$E[3] = E[0] + 3 = 102 + 3 = 105$
5º	4	$E[4] = E[0] + 4 = 102 + 4 = 106$
6º	5	$E[5] = E[0] + 5 = 102 + 5 = 107$
-	6	$E[6] = E[0] + 6 = 102 + 6 = 108$ (erro)

Vetores

```
1 import java.util.Scanner;
2
3 public class Program {
4
5     public static void main(String[] args) {
6         Scanner s = new Scanner(System.in);
7
8         float n1_aluno1, n2_aluno1, media_aluno1;
9         float n1_aluno2, n2_aluno2, media_aluno2;
10        // Outras 98 * 3 variáveis para as notas aqui.
11
12        int atual = 0; // Indica o aluno atual, começando em zero.
13
14        System.out.println("Aluno " + atual);
15        System.out.print("Nota 1: ");
16        n1_aluno1 = s.nextFloat();
17        System.out.print("Nota 2: ");
18        n2_aluno1 = s.nextFloat();
19        media_aluno1 = (n1_aluno1 + n2_aluno1) * 0.5f;
20        System.out.println("Média: " + media_aluno1);
21
22        ++atual;
23
24        System.out.println("Aluno " + atual);
25        System.out.print("Nota 1: ");
26        n1_aluno2 = s.nextFloat();
27        System.out.print("Nota 2: ");
28        n2_aluno2 = s.nextFloat();
29        media_aluno2 = (n1_aluno2 + n2_aluno2) * 0.5f;
30        System.out.println("Média: " + media_aluno2);
31
32        ++atual;
33        // E assim por diante, para as outras 98 * 3 variáveis...
34    }
35
36 }
```

Solução (parcial) do problema inicial da aula, sem usar vetores.

Vetores

```
1  import java.util.Scanner;
2
3  public class Program {
4
5      public static void main(String[] args) {
6          Scanner s = new Scanner(System.in);
7
8          final int ALUNOS = 100;
9
10         float n1[] = new float[ALUNOS];
11         float n2[] = new float[ALUNOS];
12         float media[] = new float[ALUNOS];
13
14         for (int atual = 0; atual < ALUNOS; ++atual) {
15             System.out.println("Aluno " + atual);
16             System.out.print("Nota 1: ");
17             n1[atual] = s.nextFloat();
18             System.out.print("Nota 2: ");
19             n2[atual] = s.nextFloat();
20             media[atual] = (n1[atual] + n2[atual]) * 0.5f;
21             System.out.println("Média: " + media[atual]);
22         }
23     }
24 }
25
```

Solução (completa) do problema inicial da aula, usando vetores.

Vetores

```
1  import java.util.Scanner;
2
3  public class Program {
4
5      public static void main(String[] args) {
6          Scanner s = new Scanner(System.in);
7
8          final int ALUNOS = 1000;
9
10         float n1[] = new float[ALUNOS];
11         float n2[] = new float[ALUNOS];
12         float media[] = new float[ALUNOS];
13
14         for (int atual = 0; atual < ALUNOS; ++atual) {
15             System.out.println("Aluno " + atual);
16             System.out.print("Nota 1: ");
17             n1[atual] = s.nextFloat();
18             System.out.print("Nota 2: ");
19             n2[atual] = s.nextFloat();
20             media[atual] = (n1[atual] + n2[atual]) * 0.5f;
21             System.out.println("Média: " + media[atual]);
22         }
23     }
24 }
25
```

Solução (completa) do problema para mil alunos, usando vetores.



Matrizes

Matrizes

- Vetores da seção anterior: vetores unidimensionais
- Matrizes: vetores com duas (bidimensionais) ou mais dimensões (multidimensionais)
 - 2D: linha x coluna

Matrizes

- Declaração parecida com vetores, porém usando [] para cada dimensão.
- Matriz bidimensional:

<tipo> <nomeDaVariavel>[<linhas>][<colunas>];

variável chamada minhaMatriz

int minhaMatriz[3][3];

É uma matriz 3x3 e armazena 9 elementos do tipo int


Matrizes

- Acesso a um elemento da matriz também é feito usando [] após o nome da variável.

```
<nomeDaVariavel>[<linha>][<coluna>] = <valor>;  
write("Conteúdo:" + <nomeDaVariavel>[<linha>][<coluna>]);
```

Associa o valor 99 ao elemento na linha 1 e coluna 0 [1][0] da matriz chamada minhaMatriz

```
minhaMatriz[1][0] = 99;  
write("Conteúdo:" + minhaMatriz[1][0]);
```



Acessa o valor armazenado em [1][0] da matriz chamada minhaMatriz



Matrizes

- Matriz N-dimensional:

<tipo> <nomeDaVariavel>[<tamanho dimensão 1>][<tamanho dimensão 2>]...[<tamanho dimensão N>];

int minhaMatriz[9][9][9][9];

variável chamada minhaMatriz é uma matriz 9x9x9x9



Matrizes

Exemplo: Jogo da Velha (variáveis)

char M[3][3]; // ‘’, ‘O’ e ‘X’

int linha;

int coluna;

M[0][0]	M[0][1]	M[0][2]
M[1][0]	M[1][1]	M[1][2]
M[2][0]	M[2][1]	M[2][2]

Matrizes

Exemplo: Jogo da Velha (iniciar matriz com valores vazios)

```
for (linha = 0; linha < 3; ++linha) {  
  for (coluna = 0; coluna < 3; ++coluna) {  
    M[linha][coluna] = '';  
  }  
}
```

''	''	''
''	''	''
''	''	''

Matrizes

Exemplo: Jogo da Velha (primeira jogada: 'X' no centro)

```
if (M[1][1] == '') {  
    M[1][1] = 'X';  
}
```

''	''	''
''	'X'	''
''	''	''

Matrizes

Exemplo: Jogo da Velha (segunda jogada: 'O' no centro)

```
if (M[1][1] == 'O') { // false  
    M[1][1] = 'O'; // não é executado  
}
```

“	“	“
“	‘X’	“
“	“	“

Matrizes

Exemplo: Jogo da Velha (segunda jogada: 'O' no último quadrado)

```
if (M[2][2] == ' ') {  
    M[2][2] = 'O';  
}
```

' '	' '	' '
' '	'X'	' '
' '	' '	'O'

Matrizes

Exemplo: Jogo da Velha (verificar se jogador 'X' ganhou pela primeira linha)

```
if (M[0][0] == 'X' && M[0][1] == 'X' && M[0][2] == 'X') {  
  // Jogador 'X' venceu!  
}
```

'X'	'X'	'X'
“	“	“
“	“	“

Matrizes

Exemplo: Jogo da Velha (verificar se jogador 'O' ganhou pela diagonal principal)

```
if (M[0][0] == 'O' && M[1][1] == 'O' && M[2][2] == 'O') {  
    // Jogador 'O' venceu!  
}
```

'O'	'	'
'	'O'	'
'	'	'O'

5c

Linguagem Java

Java

- Vetores em Java podem ser de tipos primitivos e objetos.
- Primeiro elemento começa no índice 0.
 - Último índice é $n - 1$.
 - n = tamanho do vetor (quantidade de elementos).
- Vetores são declarados usando [], que podem ser inseridos logo após o tipo de dado ou logo após o nome da variável.
- É possível criar um vetor em Java de algumas maneiras, conforme os próximos slides.

Java

```
// Declaramos um vetor de chars.  
char a[];  
// Depois alocamos memória para armazenar N chars.  
// Nesse caso, 5 chars, de a[0] até a[4].  
a = new char[5];  
  
// Declaramos outro vetor de chars.  
char[] b;  
// Depois alocamos memória para armazenar N chars.  
// Nesse caso, 5 chars, de b[0] até b[4].  
b = new char[5];  
  
// Atribuimos valores para os elementos do vetor de chars b.  
b[0] = '?';  
b[1] = '@';  
b[2] = 'A';  
b[3] = '9';
```

Java

```
// Declaramos um vetor de inteiros e alocamos memória para 10 elementos.  
// Vetor com capacidade 10, de vetor[0] até vetor[9].  
int[] vetor = new int[10];  
  
// Declaramos outro vetor de inteiros e alocamos memória para  
// 20 elementos. Vetor com capacidade 20.  
int outroVetor[] = new int[20];  
  
// Declaramos um vetor de inteiros e já atribuímos os valores do vetor.  
// Nesse caso, a memória é alocada conforme a quantidade de valores atribuídos.  
int[] c = { 10, 20, 30, 40, 50 };  
  
// Outro exemplo, com double.  
double[] x = { 1.2, 3.4, 5.6, 7.8 };
```

Java

Atenção!

- Em Java, vetores (arrays) são objetos.
 - Instâncias da classe `java.lang.Object`.
- Para obter o tamanho de um vetor, acessamos o atributo `length`:

```
int[] vetor = new int[10];  
System.out.println("Tamanho do vetor: " + vetor.length);
```

Java

▪ Exemplo de matrizes em Java

```
// Declaramos, alocamos memória e iniciamos uma matriz bidimensional de inteiros.  
int x[][] = {  
    { 1, 2, 3 },  
    { 4, 5, 6 }  
};
```

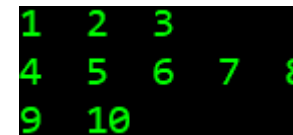
```
// Com a declaração anterior, os elementos da matriz x são:  
// c[0][0] = 1 c[0][1] = 2 c[0][2] = 3  
// c[1][0] = 4 c[1][1] = 5 c[1][2] = 6
```

Java

▪ Exemplo de matrizes em Java

// É possível declarar uma matriz com linhas de tamanhos diferentes,
// ainda que isto não seja muito usado:

```
int[][] y = {  
    { 1, 2, 3 },  
    { 4, 5, 6, 7, 8 },  
    { 9, 10 }  
};  
  
for (int row = 0; row < y.length; ++row) {  
    for (int col = 0; col < y[row].length; ++col) {  
        System.out.print(y[row][col] + " ");  
    }  
    System.out.println();  
}
```



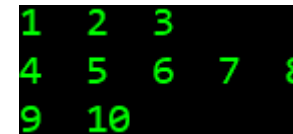
1	2	3		
4	5	6	7	8
9	10			

Java

▪ Exemplo de for-each em Java

// É possível declarar uma matriz com linhas de tamanhos diferentes,
// ainda que isto não seja muito usado:

```
int[][] y = {  
    { 1, 2, 3 },  
    { 4, 5, 6, 7, 8 },  
    { 9, 10 }  
};  
  
for (int row : y) {  
    for (int col : row) {  
        System.out.print(col + " ");  
    }  
    System.out.println();  
}
```



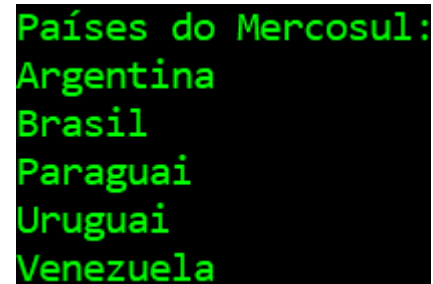
```
1 2 3  
4 5 6 7 8  
9 10
```

Java

▪ Exemplo de for-each e var em Java

```
System.out.println("Países do Mercosul:");
```

```
String mercosul[] = { "Argentina", "Brasil", "Paraguai", "Uruguai", "Venezuela" };  
for (var pais : mercosul) {  
    System.out.println(pais);  
}
```



```
Países do Mercosul:  
Argentina  
Brasil  
Paraguai  
Uruguai  
Venezuela
```


5d

Exemplo Java

Exemplo 8: Array

```
import java.util.Scanner;

class Exemplo8 {
    static final int QTDE_NUMEROS = 5;

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        float soma = 0.0f;
        int vetor[] = new int[QTDE_NUMEROS];

        // Leitura e soma dos números.
        for (int i = 0; i < QTDE_NUMEROS; ++i) {
            System.out.print("Digite o " + (i + 1) + "o valor: ");
            vetor[i] = s.nextInt();
            soma = soma + vetor[i];
        }

        // Continua no próximo slide...
```

Exemplo 8: Array

```
// ...continuação do slide anterior.
```

```
// Calcula a média.
```

```
float media = soma / QTDE_NUMEROS;
```

```
int qtdeMaiorMedia = 0;
```

```
// Encontra a quantidade maior do que a média.
```

```
for (int i = 0; i < QTDE_NUMEROS; ++i) {
```

```
    if (vetor[i] > media)
```

```
        ++qtdeMaiorMedia;
```

```
}
```

```
System.out.println("Média = " + media + "\nQtde. Maior Média = " +  
qtdeMaiorMedia);
```

```
}
```

```
}
```

Referências

Referências



Programação de Computadores
Desenvolvimento de Jogos Digitais com
GameMaker: Studio
André Kishimoto
ISBN 978-85-906129-2-6

Referências

Estrutura de Dados I – Conceitos Básicos da Linguagem de Programação Java

Notas de aula da disciplina Estrutura de Dados I, 2023.

Prof. Dr. Ivan Carlos Alcântara de Oliveira

Estrutura de Dados I – Conceitos Básicos da Linguagem de Programação Java

Exemplos Java (código-fonte), 2023.

Prof. Dr. Ivan Carlos Alcântara de Oliveira



CC BY-SA 4.0 DEED

Atribuição-Compartilhagual 4.0 Internacional

Canonical URL : <https://creativecommons.org/licenses/by-sa/4.0/>

[See the legal code](#)


Você tem o direito de:


Compartilhar — copiar e redistribuir o material em qualquer suporte ou formato para qualquer fim, mesmo que comercial.

Adaptar — remixar, transformar, e criar a partir do material para qualquer fim, mesmo que comercial.

O licenciante não pode revogar estes direitos desde que você respeite os termos da licença.

De acordo com os termos seguintes:

 **Atribuição** — Você deve dar o [crédito apropriado](#), prover um link para a licença e [indicar se mudanças foram feitas](#). Você deve fazê-lo em qualquer circunstância razoável, mas de nenhuma maneira que sugira que o licenciante apoia você ou o seu uso.

 **Compartilhagual** — Se você remixar, transformar, ou criar a partir do material, tem de distribuir as suas contribuições sob a [mesma licença](#) que o original.

Sem restrições adicionais — Você não pode aplicar termos jurídicos ou [medidas de caráter tecnológico](#) que restrinjam legalmente outros de fazerem algo que a licença permita.



CC BY-SA 4.0 DEED

Attribution-ShareAlike 4.0 International

Canonical URL : <https://creativecommons.org/licenses/by-sa/4.0/>

[See the legal code](#)


You are free to:


Share — copy and redistribute the material in any medium or format for any purpose, even commercially.

Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

 **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

 **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.

No additional restrictions — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.

Notices: