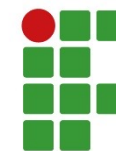


# JAVA: PLATAFORMA, TIPOS DE DADOS, OPERADORES E COMANDOS BÁSICOS

**Programação Orientada a Objetos – POOS3**

**Prof. Dr. Lucas Bueno R. de Oliveira**



**INSTITUTO FEDERAL**  
São Paulo  
Campus São Carlos

# O QUE JÁ VIMOS NAS AULAS ANTERIORES

Porque utilizaremos a linguagem Java para aprender Orientação a Objetos.

Uma breve introdução ao IntelliJ IDEA.

Nosso primeiro programa – “Olá mundo”.

# AO FINAL DESTA AULA, VOCÊ ESTARÁ APTO A...

Compreender, de forma básica, como funciona a plataforma por trás da linguagem Java.

Utilizar os tipos de dados básicos da linguagem Java.

Aplicar os diferentes tipos de operadores unários, binários e ternários.

Realizar leitura e escrita no console.

Utilizar funcionalidades para cálculos matemáticos disponíveis na linguagem.

Implementar estruturas de decisão em Java.

# PRIMEIRO, UM BREVE HISTÓRICO DO JAVA

O projeto da linguagem começou em 1990 e tinha foco na implementação de aparelhos eletrônicos, como TVs e eletrodomésticos.

A ideia por trás do Java era a criação de uma linguagem com interpretador único para diferentes plataformas. A linguagem não decolou!

Por sorte, em 1993, a WWW explodiu em popularidade e a Sun investiu no potencial do Java para adicionar conteúdo dinâmico às páginas Web.

Poucos anos depois, Java havia se tornado uma das linguagens mais populares do mundo.

Desde 2010, a Sun pertence a Oracle.

# PLATAFORMA JAVA

**Java Virtual Machine – JVM:** Programas em Java não são compilados diretamente para o sistema operacional, mas para uma camada intermediária, chamada de máquina virtual.

**Java Runtime Environment – JRE:** Bibliotecas da linguagem e a JVM formam o ambiente no qual programas Java são executados.

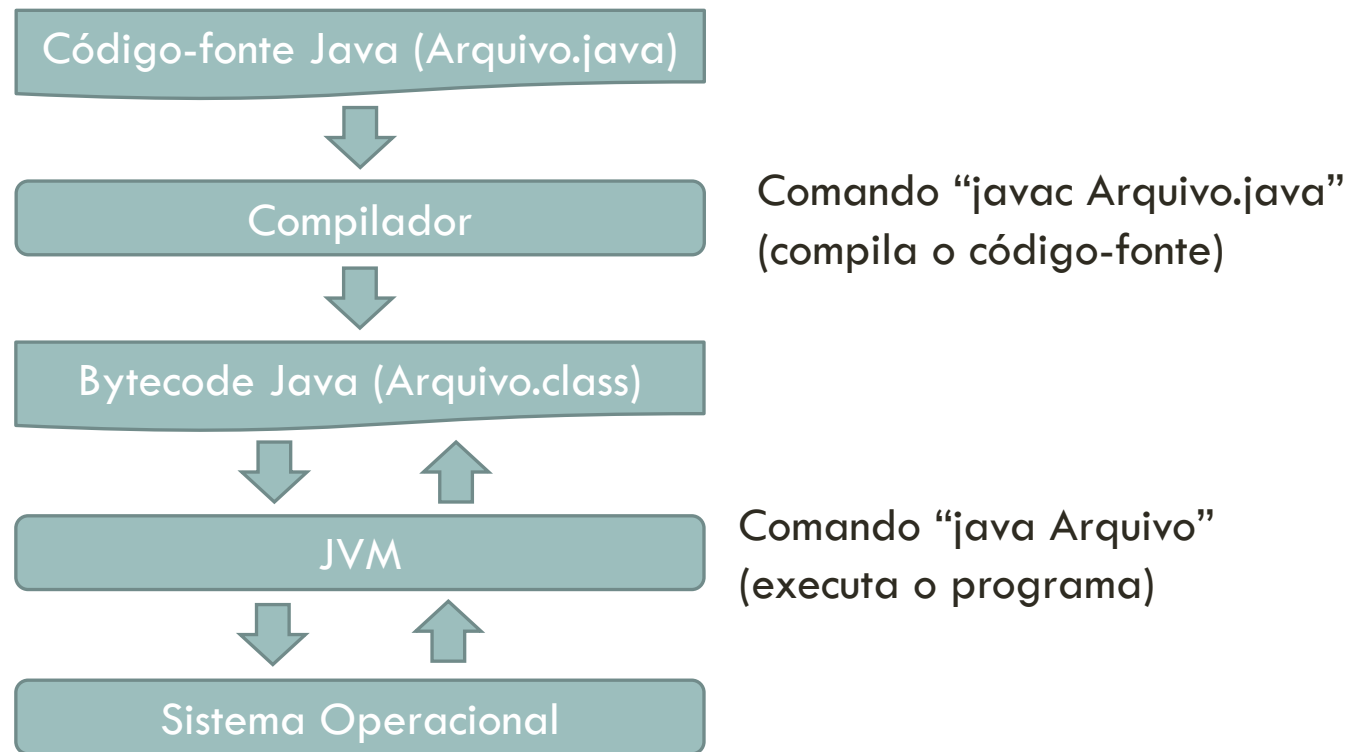
**Java Development Kit – JDK:** O JRE, em conjunto com o kit de desenvolvimento da plataforma, permitem a criação de programas em linguagem Java.

**Java Bytecode:** Códigos-fonte em Java são compilados para uma linguagem independente de plataforma, que é interpretada pela JVM, responsável pela interação com o SO.

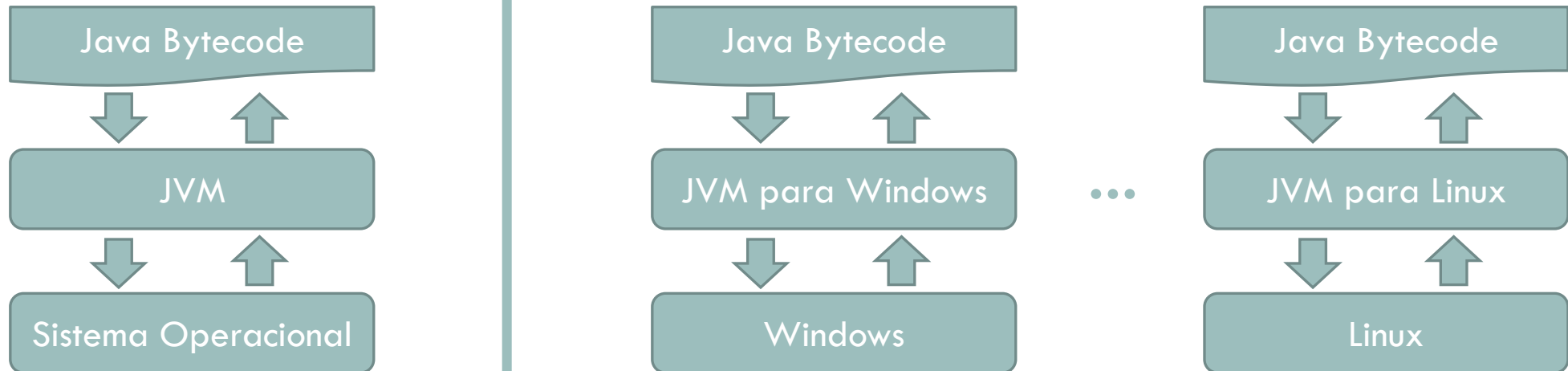
**JRE = JVM + Bibliotecas-padrão**

**JDK = JRE + Development Kit**

# PLATAFORMA JAVA

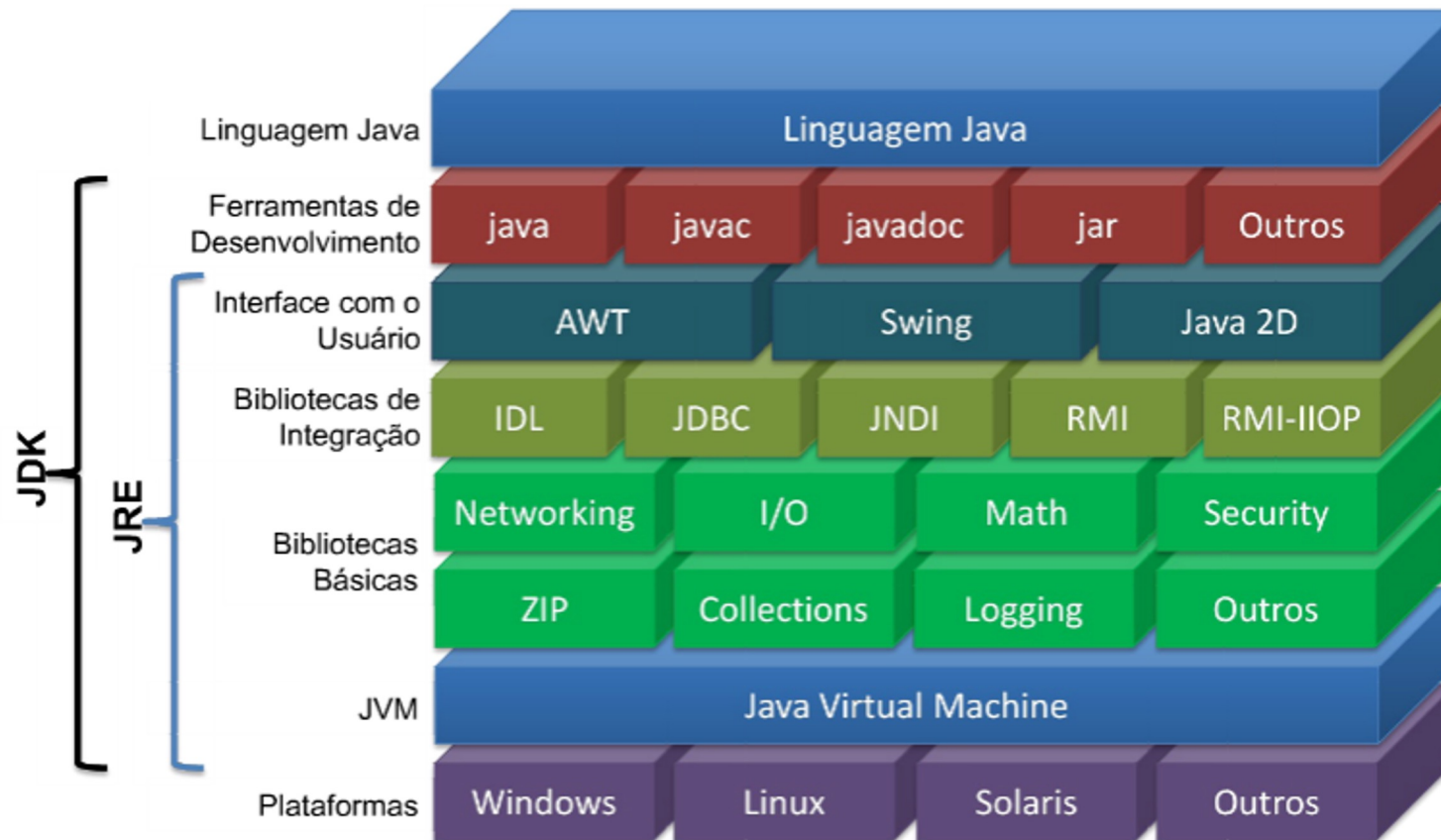


# PLATAFORMA JAVA



Princípio WORA (*Write Once, Run Anywhere*)

# PLATAFORMA JAVA



Fonte: Oracle



# TIPOS DE DADOS

Java é uma linguagem fortemente tipada, ou seja, antes de atribuir valor a uma variável, o tipo precisa ser declarado explicitamente.

```
int i; ... i = 10;  
String nome = "Homer";
```

Tipos de dados compatíveis podem ser convertidos:

```
i = 10;  
double d = i;    //Conversão implícita - Não há perda de informação  
  
d = 3.5;  
/*Conversão explícita ("cast")- Ocorre perda de informação por truncamento  
  Utilize comentários nesse formato para múltiplas linhas */  
i = (int) d;
```

# TIPOS PRIMITIVOS DE DADOS

Tipo	Descrição	Tamanho e Intervalo	Valor padrão
boolean	Valor lógico falso ou verdadeiro	---	false
char	Caractere unicode	0 e 65535 ( $2^{16}$ )	'\u0000'
byte	Inteiro de 8 bits	$-2^7 = -128$ a $2^7 - 1 = 127$	0
short	Inteiro de 16 bits	$-2^{15} = -32.768$ a $2^{15} - 1 = 32.767$	0
int	Inteiro de 32 bits	$-2^{31} = -2.147.483.648$ e $2^{31} - 1 = 2.147.483.647$	0
long	Inteiro de 64 bits	$-2^{31}$ a $2^{31}$	0L
float	Ponto flutuante de precisão simples	$1.40239846 \cdot 10^{-46}$ a $3.40282347 \cdot 10^{38}$	0.0f
double	Ponto flutuante de precisão dupla	$4.94065645841246544 \cdot 10^{-324}$ a $1.7976931348623157 \cdot 10^{+308}$	0.0d

# TIPOS PRIMITIVOS DE DADOS

```
byte varByte = -128;      varByte = 127;
short varShort = -32768;   varShort = 32767;
int varInt = -2147483648;   varInt = 2147483647;

// Perceba a letra L no final do número
long varLong = -9223372036854775808L;      varLong = 9223372036854775807L;

// Perceba a letra f no final do número
float varFloat = -100.4345f;      varFloat = 123243.4345f;

double varDouble = -3123.434354;      varDouble = 321321.3123435;
boolean varBoolean = false;      varBoolean = true;

// Valor deve estar entre aspas simples
char varChar = 'a';      varChar = 2; // ?? =o
```

# TIPOS PRIMITIVOS DE DADOS

Constantes são o oposto de variáveis: uma vez atribuído um valor, esse não pode ser alterado.

Para declarar um atributo como constante, utilize o modificador 'final'.

Por convenção, constantes são definidas em caixa alta, sendo que múltiplas palavras devem ser separadas com '\_'.

```
final boolean CONST_1;  
final boolean CONST_2 = true;  
CONST_1 = false;  
  
//CONST_2 = false;  
//Erro em tempo de compilação -> "Cannot assign a value to final variable 'CONST_2' "
```

# TIPO ENUMERAÇÃO (ENUM)

Enum é um tipo de dado especial que permite limitar o intervalo de valores aceitos por uma variável por meio de um conjunto de constantes.

Enums devem ser utilizados sempre que for necessário representar conjuntos finitos, como pontos cardeais, dias da semana, estados de pedidos, etc.

A variável do tipo enum deve ser igual a um dos valores predefinidos para ela.

Como os nomes possíveis de um enum são constantes, eles devem ser escritos em caixa alta.

```
enum <EnumName>{ <VALUE_1>, <VALUE_2>, ..., <VALUE_N> }
```

# TIPO ENUMERAÇÃO (ENUM)

Exemplo de definição e uso de enumeração:

```
public class Principal {  
    //declarado fora do main  
    enum EstadoCivil{SOLTEIRO, CASADO, SEPERADO, DIVORCIADO, VIUVO}  
  
    public static void main(String[] args) {  
        EstadoCivil ec = EstadoCivil.SOLTEIRO;  
        if (ec == EstadoCivil.CASADO)  
            System.out.println("Stay home");  
        else  
            System.out.printf("Go to the pub");  
    }  
}
```

# TIPO ENUMERAÇÃO (ENUM)

Exemplo de enumeração já disponível na linguagem:

```
//LocalDate e DayOfWeek foram introduzidos no Java 8
LocalDate date = LocalDate.now();
DayOfWeek dow = date.getDayOfWeek(); // enum para dias da semana

if(dow == DayOfWeek.SATURDAY || dow == DayOfWeek.SUNDAY)
    System.out.println("Party time!!! 🎉");
else
    System.out.println("Working time! 🕒");
```

# OPERADORES

Categoria	Operadores
Unário	<code>expr++</code> <code>expr--</code> <code>++expr</code> <code>--expr</code> <code>+expr</code> <code>-expr</code> <code>!</code>
Binário aditivo	<code>+</code> <code>-</code>
Binário multiplicativo	<code>*</code> <code>/</code> <code>%</code>
Binário relacional	<code>==</code> <code>!=</code> <code>&lt;</code> <code>&gt;</code> <code>&lt;=</code> <code>&gt;=</code> <code>instanceof</code>
Binário lógico	<code>&amp;&amp;</code> <code>  </code>
Lógico bit a bit (bitwise)	<code>&amp;</code> <code>^</code> <code> </code>
Deslocamento	<code>&lt;&lt;</code> <code>&gt;&gt;</code> <code>&gt;&gt;&gt;</code>
Ternário	<code>?:</code>
Atribuição	<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code> <code>&amp;=</code> <code>^=</code> <code> =</code> <code>&lt;&lt;=</code> <code>&gt;&gt;=</code> <code>&gt;&gt;&gt;=</code>



# OPERADORES

Em operações aritméticas, a variável de ‘tipo menor’ é promovida para a de maior ‘tipo’ antes do cálculo.

O resultado da operação é sempre do mesmo tipo das variáveis utilizadas na operação.

É possível “forçar” a conversão para um tipo compatível usando cast.

```
int x = 10 + 1; // Soma
long y = x - 7; // Subtração
long z = x * y; // Multiplicação
float r = (float) z / y; // Divisão. A conversão explícita é necessária. Por quê?
z = y % x; // Módulo
```

# OPERADORES

O operador de atribuição simples, se combinado aos operadores aritméticos, permite realizar uma operação e, logo na sequência, uma atribuição.

```
int num = 20; // Atribuição simples
num += 20; // É o mesmo que num = num + 20, logo, num vale 40.
num -= 10; // É o mesmo que num = num - 10, logo, num num vale 30
num *= 2; // É o mesmo que num = num * 2, logo, num num vale 60
num /= 6; // É o mesmo que num = num / 6, logo, num num vale 10
num %= 3; // É o mesmo que num = num % 3, logo, num num vale 1
```

# OPERADORES

Exemplo de operadores binários relacionais:

```
if(num1 == num2){...} // Verdadeiro se num1 for igual a num2
if(num1 != num2){...} // Verdadeiro se num1 for diferente a num2
if(num1 < num2){...} // Verdadeiro se num1 for menor a num2
if(num1 > num2){...} // Verdadeiro se num1 for maior a num2
if(num1 <= num2){...} // Verdadeiro se num1 for menor igual a num2
if(num1 >= num2){...} // Verdadeiro se num1 for maior igual a num2
```

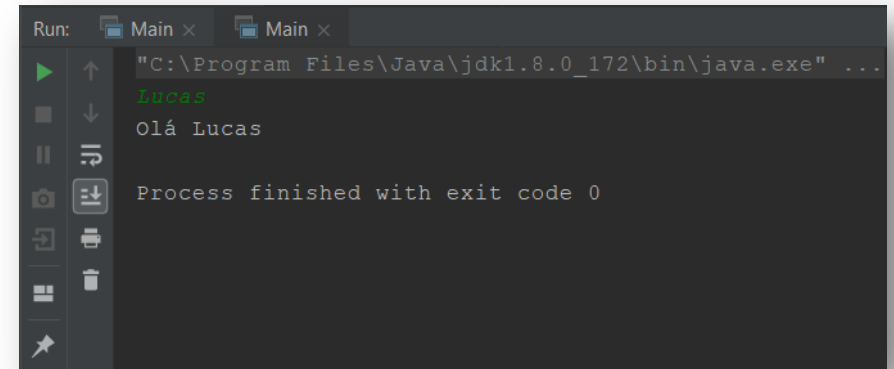
Exemplo de operadores binários condicionais:

```
if(a || b){...} // Verdadeiro se 'a' OU 'b' forem verdadeiros
if(a && b){...} // Verdadeiro se 'a' E 'b' forem verdadeiros
```

# COMANDOS BÁSICOS

Leitura e escrita do console:

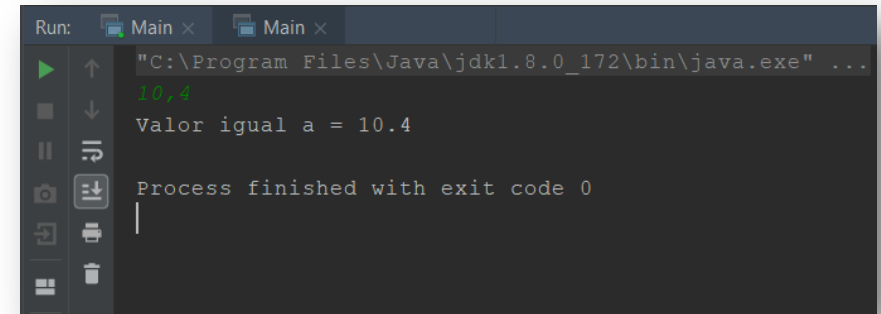
```
public class Principal {  
    public static void main(String[] args) {  
        // Cria um objeto para ler da entrada padrão  
        Scanner leitor = new Scanner(System.in);  
  
        String value = leitor.nextLine(); // Lê uma linha digitada no console  
        System.out.println("Olá " + value); // Escreve na saída padrão  
    }  
}
```



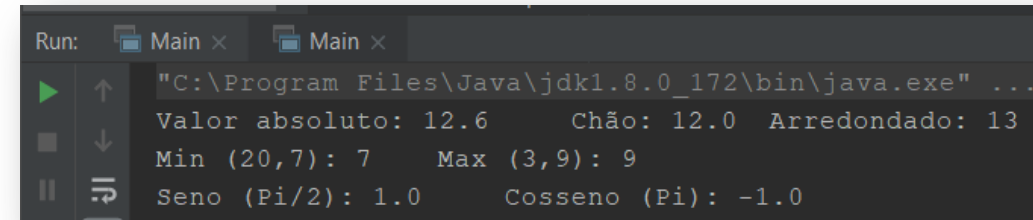
# COMANDOS BÁSICOS

Leitura e escrita de valores numéricos do console:

```
public class Principal {  
    public static void main(String[] args) {  
        Scanner leitor = new Scanner(System.in);  
  
        double value = leitor.nextDouble(); // Lê um double do console  
        // O mesmo vale para outros primitivos:  
        // boolean, byte, short, long e float  
  
        // Escreve na saída padrão. A variável 'value' é convertida para String  
        System.out.println("Valor igual a = " + value);  
    }  
}
```



# COMANDOS BÁSICOS



```
Run: Main x Main x
"C:\Program Files\Java\jdk1.8.0_172\bin\java.exe" ...
Valor absoluto: 12.6      Chão: 12.0  Arredondado: 13
Min (20,7): 7      Max (3,9): 9
Seno (Pi/2): 1.0      Cosseno (Pi): -1.0
```

A classe Math:

```
double v = 12.6;
System.out.println("Valor absoluto: " + Math.abs(-v) + "\t Chão: " +
Math.floor(v) + "\t Arredondado: " + Math.round(v)); // \t imprime um tab

System.out.println("Min (20,7): " + Math.min(20,7) + "\t Max (3,9): " +
Math.max(3,9));

System.out.println("Seno (Pi/2): " + Math.sin(Math.PI/2) + "\t Cosseno (Pi): "
+ Math.cos(Math.PI));
```

# COMANDOS BÁSICOS

Estrutura condicional IF:

```
if(<expressao_logica>){  
    <comandos>;  
}else{  
    <comandos>;  
}
```

# COMANDOS BÁSICOS

Exemplo de uso do IF:

```
public static void main(String[] args) {  
  
    boolean condition = true;  
  
    if (condition){ // Podemos omitir as chaves quando há apenas um comando  
        System.out.println("A variável condition possui valor verdadeiro.");  
    }  
    else{ // o mesmo vale para o else  
        System.out.println("A variável condition possui valor falso.");  
    }  
}
```



# COMANDOS BÁSICOS

Outro exemplo de uso do IF:

```
public static void main(String[] args) {  
  
    int val = 15;  
  
    if (val > 0)  
        System.out.println("A variável val é positiva.");  
    else if (val < 0)  
        System.out.println("A variável val é negativa.");  
    else  
        System.out.println("A variável val é igual a zero.");  
}
```

# COMANDOS BÁSICOS

Estrutura condicional IF com operador ternário:

```
<expressao_logica>? <comando_se_verdadeiro> : <comando_se_falso>;
```

Veja o exemplo:

```
double i = (x != 0.0)? Math.sin(x)/x : 1.0;
```

Logo, 'i' recebe  $\text{Math.sin}(x)/x$  se  $(x \neq 0.0)$ . Caso contrário, 'i' recebe 1.0.

# COMANDOS BÁSICOS

Estrutura condicional Switch-Case:

```
switch (<seletor>) {  
    case <value_1>:  
        <comandos>  
        break;  
    case <value_n>:  
        <comandos>  
        break;  
    default:  
        <comandos>  
        break;  
}
```

# COMANDOS BÁSICOS

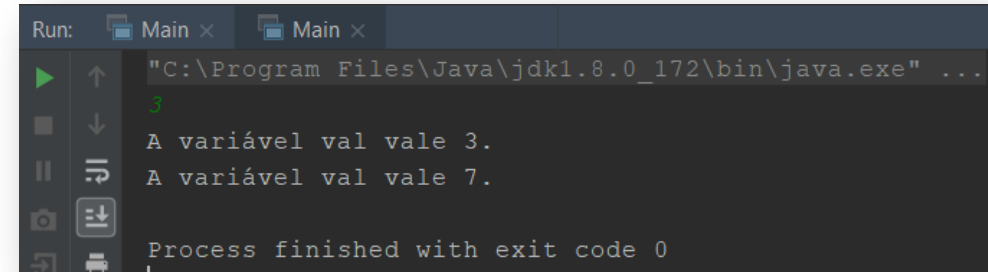
Estrutura condicional Switch-Case pode ser utilizada com os tipos byte, short, char, int, Enum, String ou para tipos variados de objetos (desde o Java 17, com *pattern matching*).

○ comando break é opcional, porém fortemente recomendado:

- Em um case, a ausência do break faz a execução continuar, independente das próximas expressões, até encontrar um comando de parada ou o fim do switch-case; e
- No default, o break é redundante, pois switch-case termina logo na sequência.

○ bloco default é opcional.

# COMANDOS BÁSICOS



```
Run: Main x Main x
"C:\Program Files\Java\jdk1.8.0_172\bin\java.exe" ...
3
A variável val vale 3.
A variável val vale 7.
Process finished with exit code 0
```

Exemplo de Switch-Case com break ausente:

```
switch (val){
    case 1: System.out.println("A variável val vale 1.");
        break;
    case 2: System.out.println("A variável val vale 2.");
        break;
    case 3: System.out.println("A variável val vale 3.");
        //break; Continua até encontra um break ou o fim do switch-case
    case 7: System.out.println("A variável val vale 7.");
        break;
    default: System.out.println("val é diferente de 1, 2, 3 ou 7.
        \nA seção Default é opcional.");
        break; // Esse break é opcional
}
```

# COMANDOS BÁSICOS

Exemplo de Switch-Case com enum:

```
public static void main(String[] args) {  
    LocalDate date = LocalDate.now();  
    switch (date.getDayOfWeek()) {  
        case MONDAY: // não é possível quantificar assim: DayOfWeek.MONDAY  
            System.out.println("Segunda-feira. ='("); break;  
        case TUESDAY:  
            System.out.println("Terça-feira. =("); break;  
        case WEDNESDAY:  
            System.out.println("Quarta-feira. =|"); break;  
        ...  
    }
```

# COMANDOS BÁSICOS

Exemplo de Switch-Case com enum:

```
...
case THURSDAY:
    System.out.println("Quinta-feira. =)"); break;
case FRIDAY:
    System.out.println("Sexta-feira! =D"); break;
case SATURDAY:
    System.out.println("Sábado! \\o/" ); break;
case SUNDAY:
    System.out.println("Domingo. <o/"); break;
}
}
```

# COMANDOS BÁSICOS

Estruturas Switch-Case podem também funcionar como expressões (Java 13):

```
switch (day) {  
    case MONDAY: case FRIDAY: case SUNDAY:  
        numLetters = 6; break;  
    case TUESDAY:  
        numLetters = 7; break;  
    case THURSDAY:  
        numLetters = 8; break;  
    case WEDNESDAY:  
        numLetters = 9; break;  
}  
System.out.println(numLetters);
```



```
System.out.println(  
    switch (day) {  
        case MONDAY, FRIDAY, SUNDAY -> 6;  
        case TUESDAY -> 7;  
        case THURSDAY -> 8;  
        case WEDNESDAY -> 9;  
    }  
);
```



# COMANDOS BÁSICOS

Uma Switch Expression pode conter mais de um comando. Nesse caso, é necessário delimitar o escopo em um bloco de chaves e indicar o retorno usando a declaração `yield`.

```
int numLetters = switch (day) {  
    case MONDAY, FRIDAY, SUNDAY -> {  
        System.out.println(6);  
        yield 6;  
    }  
    case TUESDAY -> {  
        System.out.println(7);  
        yield 7;  
    }  
    ... // O bloco default é opcional e funciona segundo a mesma lógica  
}
```

# RESUMO DA AULA

Programas Java são compilados para bytecodes e executados em uma máquina virtual.

Java possui oito tipos primitivos: boolean, char, byte, short, int, long, float e double.

Constantes são indicadas pelo modificador final.

Enumerações são tipos de dados especiais que possuem um conjunto finito de valores.

A classe Scanner permite ler do console tanto Strings quanto valores numéricos.

A classe Math reúne um conjunto de funcionalidades para cálculos matemáticos.

Estruturas condicionais if convencionais podem ser simplificadas com operadores ternários.

Estruturas switch-case podem ser utilizadas com enums, Strings ou para selecionar tipos.

Switch Expression é uma estrutura switch capaz retornar valores.

# EXERCÍCIOS

Uma livraria está fazendo uma promoção para pagamento a vista. O comprador pode escolher entre dois critérios de pagamento:

- Critério A: R\$ 0,25 por livro + R\$ 7,50 fixo
- Critério B: R\$ 0,50 por livro + R\$ 2,50 fixo

Faça um programa que o usuário informa a quantidade de livros e o programa retorna qual o melhor critério de pagamento.

Exemplos de entrada e saída esperada:

Entrada	Saída
100	Critério A
5	Critério B
20	Indiferente
-5	Erro

# INFORMAÇÕES ADICIONAIS

Tipos de dados: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

Operadores: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html>

Conversões e promoções: <https://docs.oracle.com/javase/specs/jls/se7/html/jls-5.html#jls-5.1>

Enumerações: <https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>

LocalDate: <https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html>

*Pattern Matching* em Switch: <https://www.baeldung.com/java-switch-pattern-matching>